Registered Functional Encryption for Pseudorandom Functionalities from Lattices:

Registered ABE for Unbounded Depth Circuits and Turing Machines, and More

Tapas Pal[1], Robert Schädlich[2], and Erkan Tairi[2]

[1]Karlsruhe Institute of Technology, KASTEL Security Research Labs
tapas.pal@kit.edu
[2]DIENS, École normale supérieure, PSL University, CNRS, Inria
{robert.schaedlich,erkan.tairi}@ens.fr

Abstract

Registered functional encryption (RFE) is a generalization of public-key encryption that enables computation on encrypted data (like classical FE), but without needing a central trusted authority. Concretely, the users choose their own public keys and register their keys together with a function with an (untrusted) key curator. The key curator aggregates all of the individual public keys into a short master public key, which serves as the public key of the FE scheme.

Currently, we only know RFE constructions for restricted functionalities using standard assumptions, or for all circuits using powerful tools such as indistinguishability obfuscation, and only in the non-uniform model. In this work, we make progress on this front by providing the first lattice-based constructions of RFE for *pseudorandom* functionalities, where the model of computation is either non-uniform (unbounded depth circuits) or uniform (Turing machines). Intuitively, we call a functionality pseudorandom if the output of the circuit is indistinguishable from uniform for every input seen by the adversary. Security relies on LWE and a recently introduced primitive called *pseudorandom FE* (prFE), which currently can be instantiated from evasive LWE.

We illustrate the versatility of these new functionalities for RFE by leveraging them to achieve key-policy and ciphertext-policy registered attribute-based encryption and registered predicate encryption schemes (KP-RABE, CP-RABE and RPE) for both unbounded depth circuits and Turing machines. Existing RABE constructions support only bounded depth circuits, and prior to our work there neither existed RABE for uniform models of computation nor RPE. As an appealing feature, all our constructions enjoy asymptotic optimality in the sense that their parameters depend neither on the length of public attributes nor the size of policies.

Along the way, we can also improve on the state-of-the-art for classical attribute-based encryption (ABE) and predicate encryption (PE). Specifically, we obtain new constructions for KP-ABE, CP-ABE and PE for Turing machines with optimal asymptotic parameters. For KP-ABE, this is an in improvement in terms of efficiency, whereas for CP-ABE and PE we are not aware of any prior purely lattice-based construction supporting Turing machines.

Contents

1 Introduction

Registered Functional Encryption. Registered functional encryption (RFE) [FFM+23, DPY24] has recently emerged as a promising public-key cryptographic paradigm designed to address the longstanding key-escrow issue prevalent in the existing functional encryption (FE) schemes [BSW11, GGH+13, WW21, JLS21]. In RFE, a key curator initializes a common reference string (CRS) and publicly distributes it to all users. Once the CRS is generated, the random coins used in its creation can be discarded. Importantly, the curator does not possess any secret information but merely facilitates user registration. This is a key distinction from traditional FE, where a central authority, acting as the key curator, possesses a master secret key capable of decrypting any ciphertext. Using the CRS, a new user can independently generate a public-private key pair $(\mathsf{pk}, \mathsf{sk})$ and submit a function f along with pk to the curator of RFE for registration. Upon receiving (f, pk), the curator updates the master public key mpk and the helper secret key hsk for the user. To encrypt data, a provider utilizes mpk to generate a ciphertext associated with private input x. The user, in turn, can decrypt the ciphertext using $(\mathsf{sk}, \mathsf{hsk})$ to obtain $f(x)$ while learning no additional information about x. Fundamental requirements of this framework are that the registration process remains *deterministic* to guarantee transparency and that mpk and hsk are *compact*, i.e., they have polylogarithmic size in the total number of users, and can be efficiently updated.

A special case of RFE is the notion of registered attribute-based encryption (RABE) [HLWW23], where an unchanging message μ is encrypted along with a public attribute x, public keys are registered with respect to a public policy f and decryption returns μ if and only if $f(x) = 0$. Security is similar to the case of RFE except that x must not be hidden. RABE comes in two flavors – key-policy RABE (KP-RABE) where the policy is encoded in the public key, or ciphertext-policy RABE (CP-RABE) where the policy is encoded in the ciphertext. An interesting strengthening of KP-RABE is the so-called registered predicate encryption (RPE) where ciphertexts hide not only the message μ but also the attribute x.

Prior Work. Substantial progress has been made in constructing RFE for specific classes of functionalities. In particular, early work by Garg et al. [GHMR18] introduced the first registration-based encryption (RBE) scheme focused on identity-based policies, which laid the foundation for subsequent advancements [GV20, DKL+23, GKMR23] in security and efficiency. Constructions for more general forms of RABE have been developed based on standard assumptions in bilinear groups [HLWW23, ZZGQ23, AT24, GLWW24] and lattice-based cryptography [FWW23, CHW25, ZZC+25, YZGC25]. These developments demonstrate the potential of RFE in achieving fine-grained access control while mitigating the trust-related limitations of traditional FE schemes.

Moving beyond *all-or-nothing* type encryption paradigm, several works proposed group-based RFE constructions for specific function classes such as (attribute-based) linear functions [FFM+23, DPY24], quadratic functions [ZLZ+24, BLM+24] and (attribute-based) attribute-weighted sums [PS25]. On the other hand, RFEs for all polynomial-sized circuits are currently based on indistinguishability obfuscation (iO) [FFM+23, DPY24]. Despite significant efforts to construct iO from well-founded and standard assumptions [JLS21, JLS22, RVV24], the existing approaches remain dependent on pairings. Thus, all these RFE schemes, including the group-based ones mentioned above, are inherently insecure against quantum attacks. Recently, Zhang et al. [ZCHZ24] built RFE for all circuits providing security only in the bounded collusion model, where the total number of compromised users, Q, must be predetermined and fixed during the setup phase. The sizes of both the master public key and the helper key in [ZCHZ24] scale with Q, which diminishes the original compactness of RFE. Although their bounded collusion RFE guarantees adaptive simulation security under the post-quantum secure learning with errors (LWE) and evasive LWE [Wee22] assumptions, the scheme is proven secure in the random oracle model (ROM). Therefore, a natural open question is:

Can we build fully collusion-resistant and compact RFE, for any functionality beyond ABE, from lattice assumptions (preferably, in the standard model)?

Moreover, prior works consider non-uniform models of computation, where the function f is represented

as a circuit, and this is regardless of whether the RFE (including the special case of RABE) operates in the fully collusion-resistant or bounded collusion setting [FWW23, FFM+23, DPY24, ZCHZ24, CHW25, ZZC+25, YZGC25]. The circuit model imposes inherent limitations as it necessitates that the input size be fixed in advance and enforces a worst-case runtime for every input. Such constraints are undesirable from both theoretical and practical perspectives. While there has been remarkable progress in developing plain FE or ABE schemes that support uniform models of computation [Wat12, GKP+13, AS16, AS17, AM18, AMY19a, KNTY19, GWW19, AMY19b, LL20, GW20, AMVY21, AKM+22, AKY24a], to the best of our knowledge, no existing work has explored RFE or RABE in this setting. This leads to another fundamental open question in the field:

Can we construct compact RFE and RABE for any uniform model of computation?

Pseudorandom Functionalities. A recent work by Agrawal, Kumari and Yamada [AKY24b] initiates the study of so-called pseudorandom functionalities in the context of (classical) FE. Intuitively, a functionality is said to be pseudorandom if the output is (pseudo)random for any given input that is seen by the adversary in the security game.[1] [AKY24b] considers pseudorandom functionalities in the setting, where the underlying computational model is the circuit model. More general, we observe that pseudorandom functionalities can also be studied with respect to other models of computations, e.g., Turing machines. For notational convenience, we adopt the use of the acronym prFE from [AKY24b] for the particular case of FE for pseudorandom functionalities, where the underlying computational model captures *bounded depth circuits*, and we refer to Section 1.2 for a detailed discussion about existing instantiations of prFE.

Besides other computational models, the notion of pseudorandom functionalities can also be adapted from classical to registered functional encryption. In this case, the requirement for pseudorandomness is not imposed on all functions (as is the case in classical FE) but only on those registered by *corrupt* users whose secret keys are known by the adversary. Moreover, as in the classical setting [AKY24b], we can refine the security notion by considering *partially-hiding* pseudorandom functionalities. In this case, an input x to the function f is divided into a public part x_{pub} and a private part x_{pri}. While the correctness requirement remains the same as in standard RFE, we adopt a relaxed security notion where the public input is visible to the adversary. The key advantage of this relaxation is that it enables shorter ciphertexts whose size does not depend on the length of x_{pub}, following the convention of disregarding the public part when measuring ciphertext size.

1.1 Our Results

In this work, we make significant progress in answering the above two questions by designing RFE schemes based on lattice assumptions for the following functionalities and models of computation. Our results are summarized and compared with the state of the art in Table 1.

Unbounded-Depth Circuit Model. We achieve RFE for partially-hiding pseudorandom functionalities for unbounded depth circuits with (nearly) optimal[2] parameter sizes. We refer to Section 4 for the details, and summarize our result with the following informal theorem:

Theorem 1.1 (Pseudorandom RFE for Circuits – Informal)**.** *Assuming LWE and prFE, there exists a selectively secure compact RFE scheme for partially-hiding pseudorandom functionalities supporting unbounded depth*

[1] As we will see later, the formal definition of "pseudorandom functionality" establishes a particular security notion: a FE scheme for some function class $\mathcal{F}$ is said to be secure against pseudorandom functionalities if ciphertexts for challenge inputs x^* cannot be distinguished from random bit strings as long as the function values $f(x^*)$ (of functions $f \in \mathcal{F}$ for which the adversary possesses a secret key) cannot be distinguished from random elements of the codomain. Importantly, the condition of pseudorandom function values is only imposed on *challenge* inputs x^*, while nothing is required for other elements x of the domain.

[2] Our parameters are optimal in the sense that they depend neither on the public input length nor the function size. However, there is a logarithmic dependency on the number L of users. Since the optimal parameters for RFE in terms of L are unclear, we refer to our scheme as "nearly" optimal. Known lower bounds only relate the parameter sizes to the number of updates [MQR22].

circuits and unbounded number of users with the following parameters:

$$|\mathsf{crs}| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log L \cdot \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}| = (\log L)^2 \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = (\log L)^2 \cdot \mathsf{poly}(\lambda) + |x_{\mathsf{pri}}|$$

where L denotes the total number of registered users in the system.

Note that the parameter sizes are optimal up to a factor of $\log L$, in particular they do not depend on the circuit size or depth. By leveraging the RFE from Theorem 1.1, we instantiate the first KP-RABE, CP-RABE and RPE for unbounded depth circuits with (nearly) optimal parameters, as detailed in Section 5.

Theorem 1.2 (RABEs and RPE for Unbounded Depth Circuits – Informal)**.** *Assuming LWE and prFE, there exist selectively secure compact KP-RABE, CP-RABE and RPE schemes supporting unbounded depth circuits and having message space $\{0,1\}^\ell$ with the following parameters:*

$$|\mathsf{crs}| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log L \cdot \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = (\log L)^2 \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = (\log L)^2 \cdot \mathsf{poly}(\lambda) + \ell + \ell_{\mathsf{att}}$$

where ℓ_{att} denotes the length of an attribute.

Uniform Model. We study a uniform model of computation in the area of registration-based encryption for the first time in the literature. More precisely, we consider pseudorandom functionalities where functions are expressed as Turing machines allowing unbounded length public and private inputs. We construct RFE for partially-hiding pseudorandom functionalities supporting Turing machines with (nearly) optimal parameters. Our scheme is optimal in the sense that the parameters do not restrict the size of the Turing machine and input, nor the time/space complexity. Moreover, it obeys the standard compactness requirement of RFE, i.e., the parameters only grow logarithmically in the total number of users in the system. This is detailed in Section 4 and summarized in the following theorem:

Theorem 1.3 (Pseudorandom RFE for Turing Machines – Informal)**.** *Assuming LWE and prFE, there exists a selectively secure compact RFE scheme for partially hiding pseudorandom functionalities expressed as Turing machines and supporting an unbounded number of users with the following parameters:*

$$|\mathsf{crs}| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log L \cdot \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = (\log L)^2 \cdot \mathsf{poly}(\lambda) + |x_{\mathsf{pri}}|$$

where L denotes the total number of registered users in the system.

Similar to the constructions in the circuit model, we can directly leverage the RFE for pseudorandom functionalities to realize KP-RABE, CP-RABE[3] and RPE for Turing machines. This gives rise to the first RABE/RPE with unbounded length attributes from a plausibly post-quantum assumption.

Theorem 1.4 (RABEs and RPE for Turing machines – Informal)**.** *Assuming LWE and prFE, there exist selectively secure compact KP-RABE, CP-RABE and RPE schemes for Turing machines having message space $\{0,1\}^\ell$ with the following parameters:*

$$|\mathsf{crs}| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log L \cdot \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = (\log L)^2 \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = (\log L)^2 \cdot \mathsf{poly}(\lambda) + \ell + \ell_{\mathsf{att}}$$

where ℓ_{att} denotes the length of the attribute.

[3] More precisely, the construction of CP-ABE for some policy class $\mathcal{F}$ also requires a (classical) KP-ABE for $\mathcal{F}$ with certain properties. In the case of unbounded depth circuits, a suitable KP-ABE scheme was presented in [AKY24b] and proven secure assuming LWE and prFE. For the case of Turing machines, we provide a similar result in this work (see Theorem 1.5 below).

Scheme	Type	Policy	Security	Assumption	$\|\mathsf{crs}\|$	$\|\mathsf{mpk}\|$	$\|\mathsf{hsk}_i\|$	$\|\mathsf{ct}\|$
[FFM+23]	KP/CP/PE	UD-Cir	Adp-IND	iO + SSB	$\mathsf{poly}(\log L)$	$\mathsf{poly}(\log L)$	$\mathsf{poly}(\log L)$	$\|\mathsf{iO}(f)\|$
[HLWW23]	CP	LSSS	Adp-IND	composite-order	$L^2 \cdot \mathsf{poly}(\log L, \|x\|)$	$\|x\| \cdot \mathsf{poly}(\log L)$	$\|x\| \cdot \mathsf{poly}(\log L)$	$\|f\| \cdot \mathsf{poly}(\log L)$
[FWW23]	CP	BD-Cir	Sel-IND	witness encryption	$\mathsf{poly}(\log L)$	$\mathsf{poly}(\log L)$	$\mathsf{poly}(\log L, d, \|x\|)$	$\|\mathsf{WE.ct}\|$
[ZZGQ23]	KP/CP	ABP	Adp-IND	prime-order	$L^2 \cdot \mathsf{poly}(\log L, \|x\|)$	$\|x\| \cdot \mathsf{poly}(\log L)$	$\|x\| \cdot \mathsf{poly}(\log L)$	$\|f\| \cdot \mathsf{poly}(\log L)$
[AT24]	CP	LSSS	Adp-IND	prime-order	$L^2 \cdot \mathsf{poly}(\log L)$	$\|x\|^2 \cdot \mathsf{poly}(\log L)$	$\|x\|^2 \cdot \mathsf{poly}(\log L)$	$\|x\| \cdot \|f\| \cdot \mathsf{poly}(\log L)$
[GLWW24]	CP	LSSS	Adp-IND	composite-order	$L^{1+o(1)} \cdot \mathsf{poly}(\log L, \|x\|)$	$\mathsf{poly}(\log L, \|x\|)$	$\mathsf{poly}(\log L, \|x\|)$	$\|f\| \cdot \mathsf{poly}(\log L)$
[CHW25]	KP	BD-Cir	Sel-IND	succinct LWE + RO	$(L^2 + \|x\|^2) \cdot \mathsf{poly}(d)$	$\mathsf{poly}(d)$	$\mathsf{poly}(d)$	$\mathsf{poly}(d)$
[ZZC+25]	KP	BD-Cir	Sel-IND	LWE + (private-coin) evasive LWE	$\mathsf{poly}(\log L, d)$	$\mathsf{poly}(\log L, d)$	$\mathsf{poly}(\log L, d)$	$\mathsf{poly}(\log L, d)$
[YZGC25]	KP	BD-Cir	Sel-IND	(public-coin) evasive LWE + tensor LWE	$(L(\log L)^2 + \|x\|) \cdot \mathsf{poly}(d)$	$(\log L + \|x\|) \cdot \mathsf{poly}(d)$	$(\log L)^2 \cdot \mathsf{poly}(d)$	$(\log L + \|x\|) \cdot \mathsf{poly}(d)$
this work	KP/CP/PE	UD-Cir + TM	Sel-IND	LWE + prFE	$\log L$	$\log L$	$(\log L)^2$	$(\log L)^2$

Table 1: Comparison with existing RABE schemes. Here, "KP", "CP" and "PE" stand for key-policy, ciphertext-policy and predicate encryption, respectively. The policy classes "LSSS", "ABP", "BD-Cir", "UD-Cir" and "TM" refer to linear secret sharing schemes, arithmetic branching programs, bounded depth circuits, unbounded depth circuits and Turing machines, respectively. By "Adp" and "Sel" we denote adaptive and selective security, and by "RO" we indicate the use of a random oracle. L denotes the number of users, d denotes the circuit depth, $|x|$ denotes the attribute length and $|f|$ denotes the policy size. We denote by $|\mathsf{iO}(f)|$ and $|\mathsf{WE.ct}|$ the size of the obfuscated circuit and the size of the witness encryption ciphertext, respectively. We ignore factors polynomial in the message length ℓ and security parameter λ.

Our techniques developed in the registration-based setting prove useful to also improve the state of the art in the classical setting. Specifically, we provide the first *plain* FE for partially-hiding pseudorandom functionalities supporting Turing machines. While in the circuit model this was already provided by [AKY24b], we do not have such a scheme for Turing machines yet. By further observing that existing compilers for circuits [AKY24b] can also be leveraged in the case of Turing machines, we then obtain KP-ABE, CP-ABE and PE for TMs all with asymptotically optimal parameter (in Section 7).

Theorem 1.5 (ABEs and PE for Turing Machines – Informal)**.** *Assuming LWE and prFE, there exist very selectively secure KP-ABE, CP-ABE and RPE schemes for Turing machines having message space* $\{0,1\}^\ell$ *with the following parameters:*

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda), \qquad |\mathsf{sk}| = \mathsf{poly}(\lambda), \qquad |\mathsf{ct}| = \mathsf{poly}(\lambda) + \ell + \ell_{\mathsf{att}}.$$

Here, very selective security means that the adversary must commit to the challenge query and key queries before seeing the public parameters. We compare our KP-ABE for TM with the previous constructions of [HLL24, AKY24a, AKY24b, CW25] in Table 2. We remark that we are not aware of any previous work in the literature constructing CP-ABE or PE for TMs.

As can be observed from Table 2, ours is the first construction that has ciphertext size independent of the running time t. To achieve this we follow the approach of the KP-ABE scheme for *circuits* in [AKY24b], where the authors achieve optimal parameters by using decomposable garbled circuits to deal with each input bit and gate independently. Concretely, we generalize their approach to uniform models of computation by using two "layers" of decomposable garbled circuits where the first circuit receives the attribute length ℓ_{att} and runtime t encoded in binary using bitstrings of fixed length λ (but not the actual values of ℓ_{att} and t) and computes a garbling of the second circuit that performs the actual evaluation of the Turing machine on inputs of length ℓ_{att} and runtime t. This allows us to also get rid of the $|M|$ dependence on $|\mathsf{sk}|$ from [AKY24b]. For more details about this technique we refer the reader to Section 2.3.

Scheme	Assumption	$\|\mathsf{mpk}\|$	$\|\mathsf{sk}\|$	$\|\mathsf{ct}\|$
[HLL24]	LWE + (public-coin) evasive LWE with structured errors	$O(1)$	$\|M\|$	$t \cdot 2^s$
[AKY24a]	LWE + (private-coin) evasive LWE + circular tensor LWE	$O(1)$	$\|M\|^2$	t^2
[AKY24b]	LWE + prFE	$O(1)$	$\|M\|$	t
[CW25]	(public-coin) circular evasive LWE	$O(1)$	$O(1)$	t
this work	LWE + prFE	$O(1)$	$O(1)$	$O(1)$

Table 2: Comparison with existing KP-ABE schemes for TM. Here, $|M|$ denotes the size of the Turing machine M as a circuit, which is roughly linear in the number of states, t denotes the running time and s denotes the space of M. We ignore factors polynomial in the message length ℓ and attribute length ℓ_{att} as they are swallowed by the running time t. We additionally ignore polynomial factors in the security parameter λ. The assumption "prFE" is currently implied by (private-coin) evasive LWE.

1.2 On prFE as an Assumption

Our schemes rely on prFE [AKY24b] (i.e., pseudorandom FE for bounded depth circuits) and a primitive called poly-domain pseudorandom iO (pPRIO) [AKY24c] which can be built generically from prFE and one-way functions. The security of prFE and pPRIO currently rely on LWE and *private-coin* evasive LWE [AKY24b]. Even though evasive LWE has seen several recent counterexamples [BÜW24, DJM+25, AMYY25, HJL25], we believe that in the post-quantum setting the use of primitives for *pseudorandom* functionalities that rely on evasive LWE is still preferable over general purpose tools such as FE and iO for all circuits. More precisely, while in the non-post-quantum setting we can construct FE and iO for general circuits under well-studied assumptions [JLS21, JLS22, RVV24], in the post-quantum setting the situation is more grim. The existing (plausibly) post-quantum iO candidates can be roughly classified into two categories. The first category involves iO constructions based on (constant-degree) graded encoding schemes [LV16, Lin17, LT17], which rely on ad-hoc lattice-based assumptions and are subject to so-called "zeroizing attacks" [CGH17, Pel18, CHKL18, CVW18, CCH+19]. The second category involves constructions based on some form of "circular LWE with leakage" assumptions [BDGM20, GP21, WW21, DQV+21, BDGM22, HJL25], where subsequent cryptanalysis [HJL21, JLLS23, BDJ+24] has demonstrated either counterexamples or attacks against these assumptions (with the exception of the newly proposed candidate from [HJL25]).

In a bit more detail, the main hurdle frequently faced in the aforementioned constructions is hiding the randomness leakage, i.e., the adversary must not obtain low norm polynomial equations over the integers that can be solved to obtain harmful leakage via zeroizing attacks. Addressing this state of affairs in a formal manner, the original intention behind evasive LWE was to define an assumption that completely avoids the class of zeroizing attacks. Even though the recent works [BÜW24, DJM+25, AMYY25, HJL25] have shown that this is not entirely true, it indicates that any plausible new assumption in the future will only be *more stringent* with respect to this class of attacks. Therefore, despite the attacks on evasive LWE, the challenge in designing schemes that avoid zeroizing attacks remains. One example here is the construction of prFE from [AKY24b], which explicitly prevents such harmful leakage by smudging it with a large PRF value and the *pseudorandom* decryption result, making the applicability of zeroizing attacks in our opinion unlikely. To us, this intuition is in line with the previously mentioned works which, despite breaking certain regimes of the (private-coin) evasive LWE assumption, do not seem to come close to breaking the actual prFE scheme or other constructions based on the same assumption, e.g., the ABE in [AKY24a]. We therefore believe it to be a plausible scenario that—after the attacks on evasive LWE have stabilized—a new stricter (maybe even falsifiable) assumption could be formalized for which no counterexamples would exist and yet still would be sufficient for the proof of prFE.

This would immediately imply prFE and all our results from the same assumption.

Lastly, we remark that, analogous to evasive LWE, it is known that prFE and pPRIO cannot be secure for arbitrary samplers [BDJ+24, AMYY25]. However, for this work we only need to consider *specific* samplers along the lines of [AKY24b, AKY24c], which include samplers with natural auxiliary information (as opposed to samplers with potentially harmful auxiliary information used in the recent counterexamples of [BDJ+24, AMYY25]). Hence, when we invoke the security of prFE or pPRIO, we always invoke it with respect to a specific sampler class induced by the respective application. However, for ease of exposition, we sometimes omit the reference to the specific sampler.

2 Technical Overview

We start with a high-level overview illustrating the modular structure of our results. Figure 1 depicts the connections between our constructions in the registration-based setting.

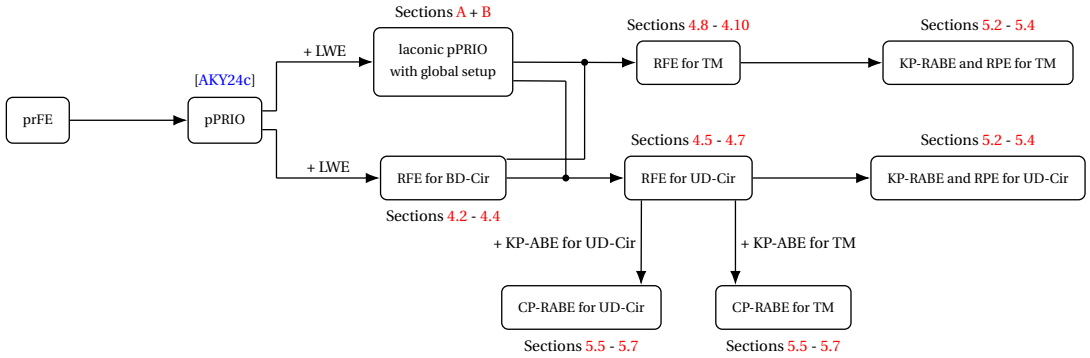

Figure 1: Roadmap for the technical part (registration-based setting). Here, "BD-Cir", "UD-Cir" and "TM" refer to bounded depth circuits, unbounded depth circuits and Turing machines, respectively. For clarity, we do not mention the use of one-way functions as additional building block.

Our starting point is the abovementioned variant of obfuscation called pPRIO which can be instantiated from prFE [AKY24b, AKY24c]. Using pPRIO, we can construct RFE for bounded depth circuits as an intermediate step which can subsequently be leveraged to obtain pseudorandom RFE for both unbounded depth circuits and Turing machines. For this, we need another tool that we call laconic pPRIO with global setup which can also be obtained from (plain) pPRIO. Being equipped with pseudorandom RFE for some function class $\mathcal{F}$, we show how to generically obtain KP-RABE and RPE for policy class $\mathcal{F}$. Furthermore, we show how RFE for unbounded depth circuits together with a (classical) KP-ABE for a policy class $\mathcal{F}$ can be used to obtain CP-RABE for $\mathcal{F}$. In the case of unbounded depth circuits, [AKY24b] have provided a suitable KP-ABE construction based on prFE. For Turing machines, we will establish a similar result in this work (see below).

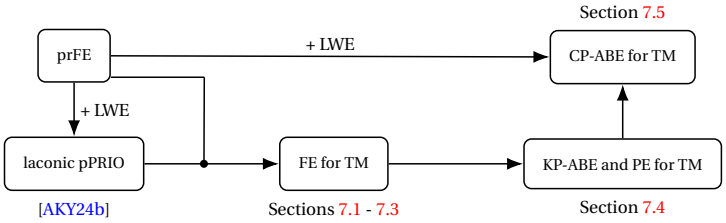

Figure 2: Roadmap for the technical part (classical setting)

Figure 2 provides an overview of our results in the classical FE setting. Using prFE and another tool called laconic pPRIO, which is known to exist assuming prFE [AKY24b, AKY24c], we construct pseudorandom FE for

Turing machines which can in turn be leveraged to obtain KP-ABE and PE for Turing machines. Furthermore, using prFE and the KP-ABE from the previous step, we can also obtain CP-ABE for Turing machines.

In the remainder of the technical overview, we start with a discussion of the basic RFE for bounded depth circuits (Section 2.1). Subsequently, we show how to lift this scheme to unbounded depth circuits, where we pay particular attention to the challenge of achieving optimal asymptotic parameters. We also present a simplified version of the generic compilers to obtain KP-RABE, RPE and CP-RABE (Section 2.2). Finally, we discuss the main ideas behind our construction of pseudorandom RFE for Turing machines which can subsequently be plugged into the same generic compilers for RABE and RPE (Section 2.3).

2.1 Pseudorandom RFE for Bounded Depth Circuits

Thanks to the powers-of-two compiler developed in [GHMR18, HLWW23, FFM+23], it suffices to design constructions for the simpler *slotted* RFE (sRFE) primitive. In a sRFE scheme, the number L of slots[4] is fixed, hence it does not incorporate the complex update mechanism of RFE. In a bit more detail, each user $i \in [L]$ can generate their own key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ and register pk_i along with a function f_i of their choice. After receiving all tuples $\{(\mathsf{pk}_i, f_i)\}_{i\in[L]}$, the transparent *key aggregator* generates a short master public key mpk and a set of helper secret keys $\{\mathsf{hsk}_i\}_{i\in[L]}$. Encryption of an input x uses mpk to generate a ciphertext ct, and each user i can decrypt ct using $(\mathsf{sk}_i, \mathsf{hsk}_i)$ to learn $f_i(x)$. Security requires that an adversary possessing sk_i cannot learn anything about x beyond $f_i(x)$.

In addition, sRFE is subject to a strong efficiency requirement: a scheme is called *compact* if master public keys, helper secret keys and ciphertexts scale at most polylogarithmically in the number L of users. We note that compactness is a crucial requirement without which the sRFE primitive would be completely trivial. Indeed, dropping compactness would allow us to pick any public key encryption (PKE) scheme PKE, sample key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, define $\mathsf{mpk} = \{\mathsf{pk}_i\}_{i\in[L]}$ and generate a ciphertext for an input x as $\mathsf{ct} = \{\mathsf{ct}_i \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_i, f_i(x))\}_{i\in[L]}$, i.e., mpk and ct were simply a collection of independent PKE public keys and ciphertexts. With the compactness requirement, however, constructing sRFE schemes is a highly challenging task requiring complex techniques to compress master public keys and ciphertexts.

The Registration-Based IBE of [DKL+23]. The scheme of Döttling et al. [DKL+23], henceforth DKL^+, chooses the dual-Regev PKE [GPV08] as their starting point and observe that its structure synergizes remarkably well with a variant of Ajtai's collision-resistant hash function [Ajt96, GGH96]. Specifically, they consider the Merkle tree with respect to this particular hash function with the leaves being the (dual-Regev) public keys of the users. Intuitively, they show that knowledge of the tree's root is sufficient to generate ciphertexts for any identity $i \in [L]$, i.e., it can serve as master public key. Furthermore, the nodes along the path from the root to an identity's leaf are sufficient to decrypt ciphertexts with respect to this identity, making them a suitable helper secret key. Since the length of this path equals the depth of the hash tree which is $\ell := \log L$, the size of mpk and hsk_i meet the compactness condition.

In a bit more detail, let us fix public parameters $\mathbf{A}, \mathbf{B}_0, \mathbf{B}_1 \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and let $(\mathsf{pk}_i := \mathbf{y}_{\mathsf{bits}(i)} = \mathbf{A}\mathbf{k}_i, \mathsf{sk}_i := \mathbf{k}_i)$ be the user's key pairs, where $\mathbf{k}_i \xleftarrow{\$} \{0,1\}^m$ for all $i \in [L]$. Here, $\mathsf{bits}(i) \in \{0,1\}^\ell$ denotes the binary decomposition of i. Given additionally $j \in [0;\ell]$, $\mathsf{bits}(i)_{1:j}$ denotes the substring consisting of the first j bits of $\mathsf{bits}(i)$. In particular, if $j = 0$ we obtain the empty string ε of length 0. For all $j = \ell-1, \ldots, 0$ and $\mathsf{str} \in \{0,1\}^j$, the aggregator computes

$$\mathbf{y}_{\mathsf{str}} = \mathbf{B}_0\mathbf{G}^{-1}(\mathbf{y}_{\mathsf{str}\|0}) + \mathbf{B}_1\mathbf{G}^{-1}(\mathbf{y}_{\mathsf{str}\|1}) \,.$$

Then it outputs $\mathsf{mpk} = \mathbf{y}_\varepsilon$ and $\mathsf{hsk}_i = \{\mathbf{y}_{\mathsf{bits}(i)_{1:j-1}\|b}\}_{j\in[\ell],b\in\{0,1\}}$ for each $i \in [L]$. To generate a ciphertext of a message $\mu \in \{0,1\}$ with respect to an identity $\mathsf{bits}(i^*) = (\beta_1, \ldots, \beta_\ell)$, the encryptor samples $\mathbf{s}_0, \ldots, \mathbf{s}_\ell \xleftarrow{\$} \mathbb{Z}_q^n$ and

[4] In this work, we use the terms "slots" and "users" interchangeably.

computes

$$\begin{aligned}
\mathbf{c}_0^\top &= \mathbf{s}_0^\top \mathbf{y}_\varepsilon + \lfloor q/2 \rceil \cdot \mu \\
\mathbf{c}_j^\top &= -\mathbf{s}_{j-1}^\top (\mathbf{B}_0 \,\|\, \mathbf{B}_1) + \mathbf{s}_j^\top \left(\bar{\beta}_j \cdot \mathbf{G} \,\|\, \beta_j \cdot \mathbf{G}\right)^\top \qquad \text{for all } j \in [\ell] \\
\mathbf{c}_{\ell+1}^\top &= -\mathbf{s}_\ell^\top \mathbf{A} \,,
\end{aligned} \tag{1}$$

where $\bar{\beta}_j$ is shorthand for $(1-\beta_j)$ and wavy underlines are used in place of noise terms. For decryption, we observe that

$$\begin{bmatrix} \mathbf{B}_0 & \mathbf{B}_1 \\ \bar{\beta}_j \cdot \mathbf{G} & \beta_j \cdot \mathbf{G} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{G}^{-1}(\mathbf{y}_{\mathsf{bits}(i^*)_{1:j-1}\|0}) \\ \mathbf{G}^{-1}(\mathbf{y}_{\mathsf{bits}(i^*)_{1:j-1}\|1}) \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{\mathsf{bits}(i^*)_{1:j-1}} \\ \mathbf{y}_{\mathsf{bits}(i^*)_{1:j}} \end{bmatrix} \tag{2}$$

which uses the fact that $\bar{\beta}_j \mathbf{y}_{\mathsf{bits}(i^*)_{1:j-1}\|0} + \beta_j \mathbf{y}_{\mathsf{bits}(i^*)_{1:j-1}\|1} = \mathbf{y}_{\mathsf{bits}(i^*)_{1:j}}$. Hence, given $\mathsf{sk}_{i^*} = \mathbf{k}_{i^*}$ and $\mathsf{hsk}_{i^*} = \{\mathbf{y}_{\mathsf{bits}(i^*)_{1:j-1}\|b}\}_{j\in[\ell],b\in\{0,1\}}$, we can recover

$$\mathbf{c}_0 + \sum_{j\in[\ell]} \mathbf{c}_j^\top \begin{bmatrix} \mathbf{G}^{-1}(\mathbf{y}_{\mathsf{bits}(i^*)_{1:j-1}\|0}) \\ \mathbf{G}^{-1}(\mathbf{y}_{\mathsf{bits}(i^*)_{1:j-1}\|1}) \end{bmatrix} + \mathbf{c}_{\ell+1}^\top \mathbf{k}_{i^*} \approx \left\lfloor \frac{q}{2} \right\rceil \cdot \mu \,. \tag{3}$$

Beyond Identity-Based Encryption. In the next step, we attempt to equip the above scheme with more powerful access policies than the identity function. For this, we recall that given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times \ell_{\mathsf{in}} m}$ and a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$, we can derive a matrix $\mathbf{H}_{\mathbf{A},C} \in \mathbb{Z}_q^{n\times m}$ such that for any $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{in}}}$, we can compute a low-norm matrix $\mathbf{H}_{\mathbf{A},C,\mathbf{x}} \in \mathbb{Z}_q^{\ell_{\mathsf{in}} m\times m}$ satisfying the homomorphic relation

$$(\mathbf{A} - \mathbf{x}^\top \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathbf{A},C,\mathbf{x}} = \mathbf{A}\mathbf{H}_{\mathbf{A},C} - C(\mathbf{x}) \cdot \mathbf{G} \,. \tag{4}$$

The KP-ABE scheme by Boneh et al. [BGG+14] for bounded depth circuits, henceforth referred to as BGG^+, embeds $(\mathbf{A}-\mathbf{x}^\top\otimes\mathbf{G})$ and $\mathbf{A}\mathbf{H}_{\mathbf{A},C}$ in the ciphertexts and secret keys, respectively, and multiplication of a ciphertext by $\mathbf{H}_{\mathbf{A},C,\mathbf{x}}$ enables decryption. Our goal is to lift the scheme of DKL^+ to a slotted Registered ABE (sRABE), where each user $i \in [L]$ registers a policy $C_i\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$. To do so, we need to bind a user's policy C_i to their public key pk_i during the aggregation process. Looking at DKL^+, we can observe that pk_i is embedded in the term $\mathbf{y}_{\mathsf{bits}(i)}$ corresponding to the i-th leaf of the Merkle tree. We therefore try to extend $\mathbf{y}_{\mathsf{bits}(i)}$ such that it additionally hardwires the term $\mathbf{A}\mathbf{H}_{\mathbf{A},C_i}$.

First of all, we split the matrix $\mathbf{A}$ into two independent matrices $\mathbf{A}_{\mathsf{user}} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and $\mathbf{A}_{\mathsf{att}} \xleftarrow{\$} \mathbb{Z}_q^{n\times \ell_{\mathsf{in}} m}$, taking into account the different $\mathbf{A}$ matrices from the DKL^+ and BGG^+ scheme. User $i \in [L]$ generates a key pair of the form $(\mathsf{pk}_i := \mathbf{u}_i = \mathbf{A}_{\mathsf{user}}\mathbf{k}_i, \mathsf{sk}_i := \mathbf{k}_i)$, where $\mathbf{k}_i \xleftarrow{\$} \{0,1\}^m$, and the term to be placed at the i-th leaf of the Merkle tree becomes

$$\mathbf{y}_{\mathsf{bits}(i)} = \mathbf{A}_{C_i}\mathbf{r} + \mathbf{D}\mathbf{G}^{-1}(\mathbf{u}_i) \,,$$

where $\mathbf{D} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and $\mathbf{r} \xleftarrow{\$} \{0,1\}^m$ are part of the public parameters and $\mathbf{A}_{C_i} := \mathbf{A}_{\mathsf{att}}\mathbf{H}_{\mathbf{A}_{\mathsf{att}},C_i}$.[5] Corresponding to the split of the matrix $\mathbf{A}$ into $\mathbf{A}_{\mathsf{att}}$ and $\mathbf{A}_{\mathsf{user}}$, we also divide the ciphertext component $\mathbf{c}_{\ell+1}$ into

$$\begin{aligned}
\mathbf{c}_{\mathsf{att}}^\top &= -\mathbf{s}_\ell^\top (\mathbf{A}_{\mathsf{att}} - \mathbf{x}^\top \otimes \mathbf{G} \,\|\, \mathbf{D}) + \mathbf{s}_{\ell+1}^\top (\mathbf{0}_{n\times \ell_{\mathsf{in}} m} \,\|\, \mathbf{G}) \\
\mathbf{c}_{\mathsf{user}}^\top &= -\mathbf{s}_{\ell+1}^\top \mathbf{A}_{\mathsf{user}} \,,
\end{aligned} \tag{5}$$

where $\mathbf{s}_\ell, \mathbf{s}_{\ell+1} \xleftarrow{\$} \mathbb{Z}_q^n$. For decryption, we observe that

$$\begin{bmatrix} \mathbf{A}_{\mathsf{att}} - \mathbf{x}^\top \otimes \mathbf{G} & \mathbf{D} \\ \mathbf{0}_{n\times \ell_{\mathsf{in}} m} & \mathbf{G} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{H}_{\mathbf{A}_{\mathsf{att}},C_{i^*},\mathbf{x}} \cdot \mathbf{r} \\ \mathbf{G}^{-1}(\mathbf{u}_{i^*}) \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{C_{i^*}}\mathbf{r} - C_{i^*}(\mathbf{x}) \cdot \mathbf{G}\mathbf{r} + \mathbf{D}\mathbf{G}^{-1}(\mathbf{u}_{i^*}) \\ \mathbf{u}_{i^*} \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{\mathsf{bits}(i^*)} - C_{i^*}(\mathbf{x}) \cdot \mathbf{G}\mathbf{r} \\ \mathbf{u}_{i^*} \end{bmatrix}$$

[5] A similar idea was explored in concurrent works [ZZC+25, YZGC25] building KP-RABE for bounded depth circuits.

Thus, in the decryption relation (Equation (3)) we replace the term $\mathbf{c}_{\ell+1}^\top \mathbf{k}_{i^*}$ by

$$\mathbf{c}_{\mathsf{att}}^\top \begin{bmatrix} \mathbf{H}_{\mathbf{A}_{\mathsf{att}}, C_{i^*}, \mathbf{x}} \cdot \mathbf{r} \\ \mathbf{G}^{-1}(\mathbf{u}_i) \end{bmatrix} + \mathbf{c}_{\mathsf{user}}^\top \mathbf{K}_{i^*} \approx -\mathbf{s}_\ell^\top \mathbf{y}_{\mathsf{bits}(i^*)} + C_{i^*}(\mathbf{x}) \cdot \mathbf{s}_\ell^\top \mathbf{G}\mathbf{r} \,.$$

If $C_{i^*}(\mathbf{x}) = 0$, then the term $C_{i^*}(\mathbf{x}) \cdot \mathbf{s}_\ell^\top \mathbf{G}\mathbf{r}$ vanishes and decryption is possible as in the DKL^+ scheme. If $C_{i^*}(\mathbf{x}) = 1$, we note that the term $\mathbf{G}\mathbf{r}$ has the same distribution as a vector $\mathbf{b} \xleftarrow{\$} \mathbb{Z}_q^n$ chosen uniformly at random and, skipping several details, we can rely on the LWE assumption with respect to $(\mathbf{A}_{\mathsf{att}}, \mathbf{b})$ to prove that ciphertexts are pseudorandom.

Towards Functional Encryption. A sequence of recent works [HLL23, AKY24a, AKY24b] explores a technique that embeds ciphertexts of a fully homomorphic encryption (FHE) scheme into the attributes of BGG^+ encodings. This approach has lead to a variety of new lattice-based constructions such as ABE for unbounded depth circuits and Turing machines or FE for pseudorandom functionalities.

We briefly recall the core idea. Let $\mathsf{hct}_{\mathbf{t}}(\mathbf{x})$ denote a FHE ciphertext of a message $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{in}}}$ decryptable by secret key $\mathbf{t}$. We consider a BGG^+-style encoding $\mathbf{E} = \mathbf{s}^\top(\mathbf{A}_{\mathsf{att}} - \mathsf{hct}_{\mathbf{t}}(\mathbf{x}) \otimes \mathbf{G})$ that embeds the FHE ciphertext as its attribute. Let $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ be a (bounded depth) circuit and denote by Eval_C the FHE evaluation circuit with respect to the function C. Using knowledge of the "attribute" $\mathsf{hct}_{\mathbf{t}}(\mathbf{x})$, we can homorphically evaluate the encoding $\mathbf{E}$ and obtain

$$\mathbf{s}^\top(\mathbf{A}_{\mathsf{Eval}_C} - \mathsf{Eval}_C(\mathsf{hct}_{\mathbf{t}}(\mathbf{x})) \otimes \mathbf{G}) = \mathbf{s}^\top(\mathbf{A}_{\mathsf{Eval}_C} - \mathsf{hct}_{\mathbf{t}}(C(\mathbf{x})) \otimes \mathbf{G}) \,,$$

where $\mathbf{A}_{\mathsf{Eval}_C} = \mathbf{A}_{\mathsf{att}} \mathbf{H}_{\mathbf{A}_{\mathsf{att}}, \mathsf{Eval}_C}$. Brakerski et al. [BTVW17] suggested a "vectorization" of the BGG^+ ciphertext evaluation procedure so that homomorphic evaluation yields an encoding of the form $\mathbf{s}^\top(\mathbf{A}_{\mathsf{Eval}_C} - \mathsf{hct}_{\mathbf{t}}(C(\mathbf{x})))$, i.e., without the gadget matrix $\mathbf{G}$. Furthermore, they noticed that choosing the secrets $\mathbf{s}$ and $\mathbf{t}$ to be equal — a.k.a. *reusing* (or making *dual use* of) the LWE secret $\mathbf{s}$ in the BGG^+ encoding as secret of the FHE ciphertext $\mathsf{hct}_{\mathbf{s}}(C(\mathbf{x}))$ — can lead to *automatic decryption* of the FHE ciphertext as described next. Recall that in the GSW FHE scheme [GSW13], the secret key is $\mathbf{s}^\top$, a ciphertext of a message $C(\mathbf{x})$ is a matrix $\mathbf{C}$ and decryption computes $\mathbf{s}^\top \mathbf{C}$ to recover (a noisy variant of) $C(\mathbf{x})$. Then we can observe that if the above encoding uses GSW as its FHE scheme and makes dual use of $\mathbf{s}$, then it is equal to the following

$$\mathbf{s}^\top(\mathbf{A}_{\mathsf{Eval}_C} - \mathsf{hct}_{\mathbf{s}}(C(\mathbf{x}))) = \mathbf{s}^\top \mathbf{A}_{\mathsf{Eval}_C} - C(\mathbf{x}) \,,$$

which follows from the fact that multiplication of $\mathbf{s}^\top$ and $\mathsf{hct}_{\mathbf{s}}(\mathbf{x}^\top)$ causes GSW decryption to occur automatically. To summarize, the new relation of the homomorphic evaluation is

$$\mathbf{s}^\top(\mathbf{A}_{\mathsf{att}} - \mathsf{hct}_{\mathbf{s}}(\mathbf{x}) \otimes \mathbf{G}) = \mathbf{s}^\top \mathbf{A}_{\mathsf{Eval}_C} - C(\mathbf{x}) \,. \tag{6}$$

Our next goal is to incorporate this interleaving of GSW ciphertexts and BGG^+ encodings into the sRABE scheme described in the previous paragraph, thereby turning it into an sRFE scheme. Each user $i \in [L]$ registers a circuit $C_i\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ and is supposed to learn $C_i(\mathbf{x})$ when decrypting an encryption of a message $\mathbf{x}$. We discuss the changes in the scheme to adapt it from the classical BGG^+ homomorphic evaluation (recalled in Equation (4)) to the new homomorphic relation in Equation (6).

1. As mentioned above, the "attribute" in the attribute encoding of our sRFE scheme becomes a GSW ciphertext encrypting the input $\mathbf{x}$. Thus, the ciphertext component $\mathbf{c}_{\mathsf{att}}$ in Equation (5) is changed to

$$\mathbf{c}_{\mathsf{att}}^\top = -\mathbf{s}_\ell^\top(\mathbf{A}_{\mathsf{att}} - \mathsf{hct}_{\mathbf{s}_\ell}(\mathbf{x}) \otimes \mathbf{G} \,\|\, \mathbf{D}) + \mathbf{s}_{\ell+1}^\top(\mathbf{0} \,\|\, \mathbf{G}) \,.$$

   As the message is now embedded in $\mathbf{c}_{\mathsf{att}}$, we remove it from $\mathbf{c}_0$ and compute $\mathbf{c}_0^\top = \mathbf{s}_0^\top \mathbf{Y}\varepsilon + \cancel{\lfloor q/2 \rfloor \cdot \mu}$.

2. The term $\mathbf{y}_{\mathsf{bits}(i)}$ placed at the leaves of the Merkle tree is changed to

$$\mathbf{Y}_{\mathsf{bits}(i)} = \mathbf{A}_{C_i}\,\cancel{\mathbf{r}} + \mathbf{D}\mathbf{G}^{-1}(\mathbf{U}_i) \in \mathbb{Z}_q^{m\times \ell_{\mathsf{out}}} . \tag{7}$$

Note that we remove the vector $\mathbf{r}$ whose only purpose in the sRABE scheme was to provide a random mask in case decryption is not authorized, i.e., $C_i(\mathbf{x}) = 1$. In contrast, we now want to learn the actual result $C_i(\mathbf{x})$ of the homomorphic evaluation. We further note that circuits C_i in our sRFE scheme have ℓ_{out}-bits outputs. Therefore, $\mathbf{y}_{\mathsf{bits}(i)}$ is replaced by a matrix $\mathbf{Y}_{\mathsf{bits}(i)}$ with ℓ_{out} columns.

3. As already indicated in Equation (7), the user public keys are also extended to have ℓ_{out} columns now, which is necessary to maintain valid matrix operations. Hence, we let users generate

$$\left(\mathsf{pk}_i := \mathbf{U}_i = \mathbf{A}_{\mathsf{user}}\,\mathbf{K}_i \in \mathbb{Z}_q^{n\times \ell_{\mathsf{out}}},\ \mathsf{sk}_i := \mathbf{K}_i\right) \qquad \text{for } \mathbf{K}_i \xleftarrow{\$} \{0,1\}^{m\times \ell_{\mathsf{out}}} .$$

Compressing the Ciphertext. At first glance, it might seem that we already built an sRFE scheme for all bounded depth circuits in the last paragraph. However, there is one issue that we glossed over so far. Recall that in a registered IBE, each ciphertext can only be decrypted by one user. Looking again at the form of a ciphertext in the DKL^+ scheme (Equation (1)), we can see that it crucially relies on this fact. Indeed, a DKL^+ ciphertext can only be decrypted by the user i^* whose identity $\mathsf{bits}(i^*) = (\beta_1, \ldots, \beta_\ell)$ was used during the generation of the components $\{\mathbf{c}_j\}_{j\in[\ell]}$. For more general primitives like sRABE and sRFE, we run into the problem that ciphertexts may be decrypted by several (or even all) users. Naively, we could generate an independent ciphertext for each user. But then the total size of ciphertexts would be $O(L)$ which contradicts the compactness requirement allowing ciphertexts to grow only logarithmically in L.

We therefore need to develop a different method to make our above construction decryptable by multiple users. As a starting point, it is instructive to consider the use of indistinguishability obfuscation (iO) and observe that it would immediately solve our problem. Indeed, we could define a circuit C_{enc} that on input $i \in [L]$ performs the following steps:

1. If $i \notin [L]$, return $\bot$.
2. Let $\mathsf{PRF}\colon \{0,1\}^\lambda \times [L] \to \{0,1\}^\lambda$ be a PRF and $\mathsf{sd} \in \{0,1\}^\lambda$ be a seed hardwired in the description of C_{enc}. Compute $R_i := \mathsf{PRF}(\mathsf{sd}, i)$.
3. Sample LWE secrets $\mathbf{s}_0, \ldots, \mathbf{s}_{\ell+1} \xleftarrow{\$} \mathbb{Z}_q^n$ and noise vectors using random coins R_i.
4. Generate the ciphertext components $\mathsf{ct}_i = (\mathbf{c}_0, \ldots, \mathbf{c}_\ell, \mathbf{c}_{\mathsf{att}}, \mathbf{c}_{\mathsf{user}})$ with respect to user i and return ct_i.

For security, we can argue that an obfuscation $\widehat{C}_{\mathsf{enc}}$ leaks nothing beyond the set of outputs $\{C_{\mathsf{enc}}(i)\}_{i\in[L]}$ which is exactly the set of ciphertexts for all users. So security can basically be reduced to the security of the previous scheme where ciphertexts can only be decrypted by a single user. For compactness, we observe that C_{enc}'s only dependency on L is the fact that it generates ciphertext components $\{\mathbf{c}_j\}_{j\in[\ell]}$, where $\ell = \log L$. Hence, the size of C_{enc} grows logarithmically in L satisfying the compactness condition.

As discussed in Section 1.2, iO is a heavyweight tool whose usage we would like to avoid when aiming for lattice-based constructions. Hence, as an alternative we look into primitives that are weaker than iO but can be constructed from lattices. One such primitive is the recently introduced *pseudorandom iO* (PRIO) [MPV24, AKY24c, BDJ+24]. Consider a circuit $C\colon [L] \to [M]$ for $L, M \in \mathbb{N}$. Intuitively, PRIO guarantees that obfuscated circuits are pseudorandom under the condition that the image of C, i.e., $\{C(i)\}_{i\in[L]}$, cannot be efficiently distinguished from a set $\{\delta_i \xleftarrow{\$} [M]\}_{i\in[L]}$ of uniform samples from the co-domain of the circuit. In particular, if $L = \mathsf{poly}(\lambda)$, then the primitive is called *poly-input* PRIO (pPRIO) and can be instantiated from prFE and one-way functions [AKY24c]. The poly-sized domain does not pose a restriction in our case since our domain $[L]$ has polynomial size anyway. On the other hand, the requirement for pseudorandom outputs is less obvious and, jumping ahead, the condition is not satisfied for general circuits. We discuss this aspect in more detail below. For clarity, we first summarize the sRFE scheme that we have constructed so far.

Construction 2.1 (sRFE for Bounded Depth Circuits – Insecure)**.**

$\mathsf{Setup}(1^\lambda)$**:** The setup algorithm samples matrices $\mathbf{A}_{\mathsf{att}}, \mathbf{A}_{\mathsf{user}}, \mathbf{B}_0, \mathbf{B}_1$ and $\mathbf{D}$ of appropriate dimensions and outputs $\mathsf{crs} := (\mathbf{A}_{\mathsf{att}}, \mathbf{A}_{\mathsf{user}}, \mathbf{B}_0, \mathbf{B}_1, \mathbf{D})$.

$\mathsf{Gen}(\mathsf{crs})$**:** To generate a key pair, user $i \in [L]$ samples a random matrix $\mathbf{K}_i$, computes $\mathbf{U}_i = \mathbf{A}_{\mathsf{user}}\mathbf{K}_i$ and sets ($\mathsf{pk} := \mathbf{U}_i, \mathsf{sk} := \mathbf{K}_i$).

$\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, C_i)\}_{i\in[L]})$**:** On input L pairs of a public key pk_i and a circuit C_i, the key aggregator defines the encoding circuit $E[C_i](\mathbf{x}) = C_i(\mathbf{x}) \cdot \lfloor q/2 \rceil$, computes the corresponding matrix $\mathbf{A}_{E[C_i]} = \mathbf{A}_{\mathsf{att}}\mathbf{H}_{\mathbf{A}_{\mathsf{att}},\mathsf{Eval}_{E[i,C_i]}}$ and sets

$$\mathbf{Y}_{\mathsf{bits}(i)} = \mathbf{A}_{E[C_i]} + \mathbf{D}\mathbf{G}^{-1}(\mathbf{U}_i) \ .$$

For $j = \ell - 1, \ldots, 0$ and $\mathsf{str} \in \{0,1\}^j$, it computes

$$\mathbf{Y}_{\mathsf{str}} = \mathbf{B}_0\mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{str}\|0}) + \mathbf{B}_1\mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{str}\|1})$$

and outputs $\mathsf{mpk} := (\mathsf{crs}, \mathbf{Y}_\varepsilon, \ell)$ and $\mathsf{hsk}_i := \{\mathbf{Y}_{\mathsf{bits}(i)_{1:j-1}\|b}\}_{j\in[\ell],b\in\{0,1\}}$ for all $i \in [L]$.

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{x})$**:** Given the master public key mpk and an input $\mathbf{x}$, the encryptor samples a PRF seed $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$ and defines a circuit $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}](i)$[6] as follows:

1. Compute $R_i = \mathsf{PRF}(\mathsf{sd}, i)$.
2. Use R_i to sample LWE secrets $\mathbf{s}_0, \ldots, \mathbf{s}_{\ell+1}$ and noise vectors (we continue to indicate noise vectors by wavy underlines).
3. Compute a GSW ciphertext $\mathbf{X} = \mathsf{hct}_{\mathbf{s}_\ell}(\mathbf{x})$.
4. Let $(\beta_1, \ldots, \beta_\ell) = \mathsf{bits}(i) \in \{0,1\}^\ell$ be the bit representation of i and denote $\bar{\beta}_j = 1 - \beta_j$ for $j \in [\ell]$. Compute

$$\begin{aligned}
\mathbf{c}_0^\top &= \underset{\sim}{\mathbf{s}_0^\top \mathbf{Y}_\varepsilon} \\
\mathbf{c}_j^\top &= -\underset{\sim}{\mathbf{s}_{j-1}^\top (\mathbf{B}_0 \,\|\, \mathbf{B}_1)} + \mathbf{s}_j^\top \left(\bar{\beta}_j \cdot \mathbf{G} \,\|\, \beta_j \cdot \mathbf{G}\right) \qquad \text{for } j \in [\ell] \\
\mathbf{c}_{\mathsf{att}}^\top &= -\underset{\sim}{\mathbf{s}_\ell^\top (\mathbf{A}_{\mathsf{att}} - \mathbf{X} \otimes \mathbf{G} \,\|\, \mathbf{D})} + \mathbf{s}_{\ell+1}^\top (\mathbf{0} \,\|\, \mathbf{G}) \\
\mathbf{c}_{\mathsf{user}}^\top &= -\underset{\sim}{\mathbf{s}_{\ell+1}^\top \mathbf{A}_{\mathsf{user}}} \ .
\end{aligned}$$

5. Output $(\mathbf{X}, \{\mathbf{c}_j\}_{j\in[0;\ell]}, \mathbf{c}_{\mathsf{att}}, \mathbf{c}_{\mathsf{user}})$.

Then the encryptor outputs $\mathsf{ct} := \widehat{C}_{\mathsf{enc}} \leftarrow \mathsf{pPRIO.Obf}(1^\lambda, C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}])$.

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, C_i, \mathsf{ct})$**:** The decryptor evaluates

$$\left(\mathbf{X}, \{\mathbf{c}_j\}_{j\in[0;\ell]}, \mathbf{c}_{\mathsf{att}}, \mathbf{c}_{\mathsf{user}}\right) \leftarrow \mathsf{pPRIO.Eval}(\widehat{C}_{\mathsf{enc}}, i^*) \ ,$$

and computes the matrix $\mathbf{H}_{\mathbf{A}_{\mathsf{att}},E[C_{i^*}],\mathbf{X}}$ using $\mathbf{A}_{\mathsf{att}}$ and $\mathbf{X}$. Then it computes

$$\mathbf{z} = \mathbf{c}_0 + \sum_{j\in[\ell]} \mathbf{c}_j^\top \begin{bmatrix} \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|0}) \\ \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|1}) \end{bmatrix} + \mathbf{c}_{\mathsf{att}}^\top \begin{bmatrix} \mathbf{H}_{\mathbf{A}_{\mathsf{att}},E[C_{i^*}],\mathbf{X}} \\ \mathbf{G}^{-1}(\mathbf{U}_{i^*}) \end{bmatrix} + \mathbf{c}_{\mathsf{user}}^\top \mathbf{K}_{i^*} \ , \tag{8}$$

rounds $\mathbf{z}$ coordinate-wise and outputs the most significant bits.

[6] The notation $C[\alpha](\beta)$ indicates that a circuit C has the value α hardwired in its description and takes β as input.

For correctness, we observe that

$$\begin{bmatrix} \mathbf{A}_{\mathsf{att}} - \mathbf{x}^\top \otimes \mathbf{G} & \mathbf{D} \\ \mathbf{0}_{n \times \ell_{\mathsf{in}} m} & \mathbf{G} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{H}_{\mathbf{A}_{\mathsf{att}}, E[C_{i^*}], \mathbf{x}} \\ \mathbf{G}^{-1}(\mathbf{U}_{i^*}) \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_{\mathsf{bits}(i^*)} - E[C_{i^*}](\mathbf{x}) \\ \mathbf{U}_{i^*} \end{bmatrix}$$

Combining this with our analysis from Equation (2) for the other ciphertext components, we conclude that $\mathbf{z} \approx E[C_{i^*}](\mathbf{x}) = C_{i^*} \cdot \lfloor q/2 \rceil$. In particular, note that the homomorphic evaluation yields a noisy version of the output. Therefore, we use the circuit $E[C_i]$ to encode the (binary) result $C_{i^*}(\mathbf{x})$ in a way that protects it against small errors.

Achieving Pseudorandom Ciphertexts. Let us try to argue that the output of the circuit C_{enc} is pseudorandom. As the circuit generates independent random coins R_i for each $i \in [L]$, outputs for different i's are independent and we can restrict our analysis to some fixed i^*. Loosely speaking, we can start by randomizing the components $(\mathbf{X}, \{\mathbf{c}_j\}_{j \in [0;\ell]}, \mathbf{c}_{\mathsf{att}}, \mathbf{c}_{\mathsf{user}})$ output by $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}](i^*)$ by relying on LWE, but there is one relation between the ciphertext components which ensures decryption correctness (Equation (8)) and, hence, cannot be changed by relying on any computational assumption. Let us therefore restrict our analysis to only this relation and disregard all other details of the proof. We can observe that $\mathbf{c}_{\mathsf{user}}^\top \mathbf{K}_{i^*}$ is the only term in $\mathbf{z}$ that cannot be computed without knowledge of $\mathsf{sk}_{i^*} = \mathbf{K}_{i^*}$. Hence, the adversary can perform the homomorphic evaluation on $\mathbf{c}_{\mathsf{att}}$ and compute $\mathbf{z}$ up to this last term which yields $\mathbf{z} - \mathbf{c}_{\mathsf{user}}^\top \mathbf{K}_{i^*} \approx E[C_{i^*}](\mathbf{x}) - \mathbf{c}_{\mathsf{user}}^\top \mathbf{K}_{i^*}$. If slot i^* is honest, i.e., $\mathbf{K}_{i^*}$ is not known to the adversary, we can conclude from an application of the leftover hash lemma that

$$\begin{bmatrix} \mathbf{A}_{\mathsf{user}} \\ \mathbf{c}_{\mathsf{user}}^\top \end{bmatrix} \cdot \mathbf{K}_{i^*} \approx_s \begin{bmatrix} \mathbf{U}_{i^*} \\ \mathbf{v}_i^\top \end{bmatrix}$$

for some random vector $\mathbf{v}_i \xleftarrow{\$} \mathbb{Z}_q^{\ell_{\mathsf{out}}}$. Thus, the term $\mathbf{c}_{\mathsf{user}}^\top \mathbf{K}_{i^*}$ can serve as a pseudorandom masking term for the output value $E[C_{i^*}](\mathbf{x})$.

Conversely, if the adversary has knowledge of the secret key $\mathbf{K}_{i^*}$, then by the correctness of the scheme she is able to decrypt and learn $C_{i^*}(\mathbf{x})$. This makes it impossible for general circuits C_{i^*} to replace the output of $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}](i^*)$ with a pseudorandom string since this would destroy decryptability and could be observed by the adversary. Hence, if i^* is a corrupted slot, then we cannot prove security of our scheme without imposing further restrictions on C_{i^*}.

Let us analyze these necessary restrictions for pseudorandomness in more detail. Clearly, we cannot prevent the adversary from running the decryption algorithm in an honest way and leveraging the correctness of the scheme. Therefore, a minimal assumption is the fact that the real decryption result $C_{i^*}(\mathbf{x})$ must not be distinguishable from the result of decrypting a random string. Inspecting the decryption algorithm of our scheme, we can observe that if $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}](i^*) = (\mathbf{X}, \{\mathbf{c}_j\}_{j \in [0;\ell]}, \mathbf{c}_{\mathsf{att}}, \mathbf{c}_{\mathsf{user}})$ is replaced with a random string, then the decryption result is also a random value. This translates into the following necessary condition on the set $\{C_i\}_{i \in \mathcal{C}}$ of circuits which are registered in a corrupted slot $i \in \mathcal{C}$ for which the adversary knows sk_i:

$$\{C_i(\mathbf{x})\}_{i \in \mathcal{C}} \approx_c \{\mathbf{y}_i \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{out}}}\}_{i \in [c] C} \,. \tag{9}$$

Having established this necessary condition, we may hope that it is also sufficient to prove pseudorandomness of $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}](i^*)$. Unfortunately, it turns out that this is not the case (yet). The problem is that Equation (9) only requires the final decryption result to be random, i.e., *after* rounding the vector $\mathbf{z}$ in the last step of our decryption algorithm. However, the distribution of $\mathbf{z}$ itself may not be pseudorandom which can be noticed by the adversary. Taking inspiration from the construction of prFE in [AKY24b], we fix this issue by smudging $\mathbf{z}$ with a large PRF value. Specifically, we extend the encoding circuit $E[C_i]$ as follows:

$$E[i, C_i](\mathbf{x}, \mathsf{sd}') = C_i(\mathbf{x}) \cdot \lfloor q/2 \rceil + \mathsf{PRF}'(\mathsf{sd}', i) \,, \tag{10}$$

where $\mathsf{PRF}'\colon \{0,1\}^\lambda \times [L] \to [-q/4+B; q/4-B]^{\ell_{\mathsf{out}}}$ is another pseudorandom function. For sufficiently large parameters, we can choose B such that (1) the decryption noise is smaller than B with high probability, and (2) B is exponentially smaller than $q/4$. The first condition makes sure that decryption correctness is preserved. The second condition together with the pseudorandomness of C_{i^*} implies that the output of $E[i^*, C_{i^*}]$ (and thus $\mathbf{z}$) are pseudorandom.

Defining Security. Our next step consists in turning the above intuition into a formal security definition. As mentioned above, the idea of considering functionalities with pseudorandom outputs was recently explored in the context of (plain) FE by Agrawal, Kumari and Yamada [AKY24b], henceforth referred to as AKY. Let us therefore start by translating their definition of very selective *pseudorandom-ciphertext* (prCT) security into the registration-based setting. For simplicity, we only consider the case of honest and corrupted slots, where a slot $i \in [L]$ is said to be corrupted if the secret key sk_i corresponding to the i-th registered public key pk_i is known to the adversary. For simplicity, we ignore so-called malicious slots which allow the adversary to generate a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ herself and register a potentially malformed public key pk_i.

Let Samp be a PPT algorithm that on input 1^λ outputs a tuple of the form

$$\left(C_1, \ldots, C_L, \mathbf{x}, \mathcal{C} \subseteq [L], \mathsf{aux} \in \{0,1\}^*\right),$$

where $\mathcal{C}$ denotes the set of corrupted slots. We say that the sRFE scheme is very selectively prCT secure if for all PPT samplers Samp, we have

$$\left(\mathsf{aux}, \mathsf{crs}, \{C_i, \mathsf{pk}_i\}_{i\in[L]}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}}, \boxed{\mathsf{ct}^*}\right) \approx_c \left(\mathsf{aux}, \mathsf{crs}, \{C_i, \mathsf{pk}_i\}_{i\in[L]}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}}, \boxed{\mathsf{ct}^\$}\right)$$

assuming that

$$\left(\mathsf{aux}, \{C_i\}_{i\in[L]}\{\boxed{\delta_i^* := C_i(\mathbf{x}_i)}\}_{i\in\mathcal{C}}\right) \approx_c \left(\mathsf{aux}, \{C_i\}_{i\in[L]}\{\boxed{\delta_i^* \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{out}}}}\}_{i\in[\mathcal{C}]}\right).$$

Here, the challenger computes

$$\begin{aligned} \mathsf{crs} &\leftarrow \mathsf{Setup}(1^\lambda), & \mathsf{ct}^* &\leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}), \\ \{(\mathsf{pk}_i, \mathsf{sk}_i) &\leftarrow \mathsf{Gen}(\mathsf{crs})\}_{i\in[L]}, & \mathsf{ct}^\$ &\xleftarrow{\$} \mathcal{CT}, \\ (\mathsf{mpk}, \{\mathsf{hsk}_i\}_i) &\leftarrow \mathsf{Agg}(\mathsf{crs}, \{\mathsf{pk}_i, C_i\}), \end{aligned}$$

where $\mathcal{CT}$ denotes the ciphertext space. Note that the Agg algorithm is deterministic, so the adversary can derive the master public key and the helper secret keys herself and there is no need to explicitly include it into the distributions. We further note that this definition reflects the case distinction in the security proof sketched above. That is, the pre-condition applies only to *corrupted* slots while it does not impose any restrictions on the functions of *honest* slots.

The security of the AKY construction relies on evasive LWE, so it seems natural that they can only prove a very selective security notion. On the other hand, the only explicit lattice assumption in our construction is plain LWE.[7] It is therefore an interesting question to analyze if the restriction to very selective security is strictly necessary or if it is possible to prove security with respect to a stronger security notion. Intuitively speaking, it seems unlikely that we can handle adaptive challenge queries since our construction uses a BGG^+-like attribute encoding which usually requires programming of the public parameters (in our case $\mathbf{A}_{\mathsf{att}}$) during the security proof. At the same time, this seems to be the only obvious constraint and an upfront commitment to the sets of challenge functions, keys and corrupted slots might be unnecessary for the security proof.

While this intuition turns out to be true, it arises the question of how a "less selective" version of prCT security might actually look like. To find a useful generalization of the very selective version, it is instructive to

[7] Of course, our construction makes use of pPRIO as a building block which currently is also instantiated from evasive LWE. But since pPRIO does not have a global setup that fixes public parameters upfront, its security assumption seems rather irrelevant for the security level of our sRFE scheme.

recall in which fashion AKY use their FE construction in applications. E.g., they show how a prCT secure FE scheme can be used to build ABE schemes. During the security proof, they argue that each admissible adversary against the ABE scheme translates into a sampler against the FE scheme which satisfies the pre-condition. Then, exploiting the prCT security, all FE ciphertexts in the scheme can be replaced by pseudorandom strings which can in turn be used to prove security of the ABE scheme. The goal of our security notion is to enable similar arguments in the registration-based setting. In particular, if we want our final sRABE scheme to achieve selective security, where the adversary can make adaptive key-generation and corruption queries before committing to the set of challenge functions, then such an adversary must be convertible into a sampler against our sRFE scheme. Hence, our definition of *selective* prCT security considers samplers that have access to key-generation and corruption oracles before submitting the challenge functions.

We formalize this idea by defining pre- and post-games (rather than static pre- and post-distributions) which take into account the interaction between the sampler and the challenger. During these games, the challenger constructs a *record* of the communication containing either the real values or random strings. At the end of the game, this record is given to the adversary who must distinguish the real from the random case. Intuitively, we can think of these records as the pre- and post-distributions in the very selective case. The difference is however that the records are constructed dynamically depending on the sampler's oracle queries, whereas the distributions in the very selective case are the result of samplers outputting all queries upfront in one shot. For the formal definition, please see Section 4.1. Casting the above intuition about the security of our scheme into a formal proof (including the fix in Equation (10)), we can conclude that our scheme satisfies *selective* prCT security. We omit the details at this point and refer the reader to Section 4.4.

2.2 Pseudorandom RFE and RABE for Unbounded Depth Circuits

Being equipped with a prCT secure sRFE for bounded depth circuits, we first show how to extend the construction to unbounded depth circuits, followed by applications to sRABE and sRPE. These constructions bear similarities with AKY who presented analogous transformations for classical FE. We will therefore focus on challenges that are unique to the registration-based setting. Looking ahead, these results will also form the foundation of our constructions for Turing machines discussed in Section 2.3.

Dealing with Circuits of Unbounded Depth. Our first step consists in upgrading the sRFE scheme for bounded depth circuits from the previous section to unbounded depth circuits. For this, we make use of a garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$ satisfying two structural properties.

- *Decomposability.* Let $C = \{C^{(k)}\}_{k\in|C|}$ be a circuit where $C^{(k)}$ denotes the k-th gate described by a string of fixed polynomial length. We require that both the garbling for each circuit and the labels of each input bit $\{\mathbf{x}[k]\}_{k\in[\ell_{\mathsf{in}}]}$ can be generated by a circuit of fixed depth, i.e., there exist efficient algorithms

$$\begin{aligned}(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) &\leftarrow \mathsf{Garble}_{\mathsf{inp},k}(1^\lambda) \qquad \text{for } k \in [\ell_{\mathsf{in}}]\\ \widetilde{C}^{(k)} &\leftarrow \mathsf{Garble}_k(1^\lambda, C^{(k)}) \qquad \text{for } k \in [|C|]\end{aligned}$$

  such that $\mathsf{Eval}(\{\widetilde{C}^{(k)}\}_{k\in[|C|]}, \{\mathsf{lab}_{k,\mathbf{x}[k]}\}_{k\in[\ell_{\mathsf{in}}]}) = C(\mathbf{x})$.

- *Blindness.* If the result of the evaluation is pseudorandom, then the garbling looks also pseudorandom.

A garbling scheme with these properties exists assuming one-way functions [BLSV18]. The construction of our sRFE scheme for unbounded depth circuits combines a sRFE scheme $\mathsf{sRFE} = \mathsf{sRFE}.(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$ for bounded depth circuits and a decomposable and blind garbling scheme $\mathsf{bGC} = \mathsf{bGC}.(\mathsf{Garble}, \mathsf{Eval})$. Intuitively speaking, we cannot use sRFE to perform the actual evaluation of the circuit since the depth may be too high. To circumvent this bound, we let sRFE generate the garbled gates and the input labels, while the actual evaluation of the circuit happens only at the time of decryption. Here, we leverage the decomposability of bGC to ensure that all computations performed by sRFE can be implemented by circuits of bounded depth, so the sRFE can handle them. We summarize the construction as follows.

Construction 2.2 (sRFE for Unbounded Depth Circuits – Informal)**.**

$\mathsf{Setup}(1^\lambda)$**:** Run and output $\mathsf{crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda)$.

$\mathsf{Gen}(\mathsf{crs})$**:** Run and output $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{sRFE.Gen}(\mathsf{crs})$.

$\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, C_i)\}_{i\in[L]})$**:** Run and output

$$\big(\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]}\big) \leftarrow \mathsf{sRFE.Agg}\big(\mathsf{crs}, \{(\mathsf{pk}_i, C_{\mathsf{reg}}[C_i])\}_{i\in[L]}\big)\,,$$

where $C_{\mathsf{reg}}[C_i](\mathbf{x})$ does the following.

1. Run $(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) \leftarrow \mathsf{Garble}_{\mathsf{inp},k}(1^\lambda)$ for $k \in [|\mathbf{x}|]$
2. Run $\widetilde{C}_i^{(k)} \leftarrow \mathsf{Garble}_k(1^\lambda, C_i^{(k)})$ for $k \in [|C_i|]$.
3. Output $(\widetilde{C}_i = \{\widetilde{C}_i^{(k)}\}_{k\in[|C_i|]}, \{\mathsf{lab}_{k,\mathbf{x}[k]}\}_{k\in[|\mathbf{x}|]})$.

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{x})$**:** Run and output $\mathsf{ct} \leftarrow \mathsf{sRFE.Enc}(\mathsf{mpk}, \mathbf{x})$.

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, C_{i^*}, \mathsf{ct})$**:** Compute and output $\mathbf{y}$ as follows:

$$\begin{aligned}\big(\widetilde{C}_{i^*}, \{\mathsf{lab}_{k,\mathbf{x}[k]}\}_{k\in[\ell_{\mathsf{in}}]}\big) &\leftarrow \mathsf{sRFE.Dec}\big(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, C_{\mathsf{reg}}[C_{i^*}], \mathsf{ct}\big)\\ \mathbf{y} &\leftarrow \mathsf{bGC.Eval}\big(\widetilde{C}_{i^*}, \{\mathsf{lab}_{k,\mathbf{x}[k]}\}_{k\in[\ell_{\mathsf{in}}]}\big)\,.\end{aligned}$$

The security proof proceeds in three steps.

- First, invoking the prCT security of sRFE, it suffices to show that the output $(\widetilde{C}, \{\mathsf{lab}_{i,k,\mathbf{x}[k]}\}_{k\in[\ell_{\mathsf{in}}]})$ of the registered circuit $C_{\mathsf{reg}}[\mathsf{dig}_{C_i}](\mathbf{x})$ is pseudorandom for corrupted $i \in \mathcal{C}$.
- Second, we generate the garbling using the bGC simulator

$$\big(\widetilde{C}_{i^*}, \{\mathsf{lab}_{k,\mathbf{x}[k]}\}_{k\in[\ell_{\mathsf{in}}]}\big) \leftarrow \mathsf{bGC.Sim}\big(1^\lambda, C_{i^*}(\mathbf{x})\big)\,.$$

- Finally, we use the condition from the pre-game to conclude that $C_{i^*}(\mathbf{x})$ is pseudorandom. Hence, we can rely on the blindness of bGC to replace the output of the simulator with random strings.

Application to KP-sRABE and sRPE. Next, we show that a prCT secure sRFE scheme sRFE immediately gives rise to KP-sRABE and sRPE. In particular, instantiating sRFE with our scheme from Construction 2.2 yields KP-sRABE and sRPE for unbounded depth circuits. The idea behind the compiler is simple. Given L pairs (pk_i, C_i) of a public key pk_i and a policy C_i, the aggregator registers pk_i along with a circuit $C_{\mathsf{reg}}[i, C_i]$ which, on input an attribute $\mathbf{x}$, a message μ and a PRF seed sd, evaluates the policy on input $\mathbf{x}$ and outputs μ if $C_i(\mathbf{x}) = 0$ or a PRF value $\mathsf{PRF}(\mathsf{sd}, i)$ if $C_i(\mathbf{x}) = 1$. In a bit more detail, the scheme works as follows.

Construction 2.3 (KP-sRABE and sRPE for Unbounded Depth Circuits – Informal)**.**

$\mathsf{Setup}(1^\lambda)$**:** Run and output $\mathsf{crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda)$.

$\mathsf{Gen}(\mathsf{crs})$**:** Run and output $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{sRFE.Gen}(\mathsf{crs})$.

$\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, C_i)\}_{i\in[L]})$**:** Run and output

$$\big(\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]}\big) \leftarrow \mathsf{sRFE.Agg}\big(\mathsf{crs}, \{(\mathsf{pk}_i, C_{\mathsf{reg}}[i, C_i])\}_{i\in[L]}\big)\,,$$

where $C_{\mathsf{reg}}[i, C_i]$ is defined as

$$C_{\mathsf{reg}}[i, C_i](\mathbf{x}, \mu, \mathsf{sd}) = \begin{cases} \mu & \text{if } C_i(\mathbf{x}) = 0\\ \mathsf{PRF}(\mathsf{sd}, i) & \text{if } C_i(\mathbf{x}) = 1\,. \end{cases}$$

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu)$**:** Sample $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$ and output $\mathsf{ct} \leftarrow \mathsf{sRFE.Enc}(\mathsf{mpk}, (\mathbf{x}, \mu, \mathsf{sd}))$.

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, C_{i^*}, \mathsf{ct})$**:** Output $\mu \leftarrow \mathsf{sRFE.Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, C_{\mathsf{reg}}[i^*, C_{i^*}], \mathsf{ct})$.

Correctness easily follows from the correctness of sRFE and the fact that $C_{\mathsf{reg}}[i, C_i](\mathbf{x}, \mu, \mathsf{sd}) = \mu$ if $C_i(\mathbf{x}) = 0$. For security, we note that admissible adversaries satisfy $C_i(\mathbf{x}) = 1$ for all corrupted slots $i \in \mathcal{C}$. In this case, the output $C_{\mathsf{reg}}[i, C_i](\mathbf{x}, \mu, \mathsf{sd}) = \mathsf{PRF}(\mathsf{sd}, i)$ is pseudorandom. Hence, we can invoke the prCT security of sRFE to conclude that ct is also pseudorandom and, thus, hides all information about the attribute $\mathbf{x}$ and the message μ.

Application to CP-sRABE. Moreover, a prCT secure sRFE scheme can also be used to construct CP-sRABE. The construction additionally uses a *classical* KP-ABE scheme $\mathsf{kpABE} = \mathsf{kpABE}.(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with INDr security, i.e., ciphertexts not authorized for decryption are pseudorandom. The idea of the construction is as follows. Given L key-attribute pairs $(\mathsf{pk}_i, \mathbf{x}_i)$, the aggregator registers pk_i along with a circuit $C_{\mathsf{reg}}[i, \mathbf{x}_i]$ which, on input a message μ, a PRF seed sd and a master public key kpABE.msk generates a kpABE ciphertext of μ with respect to kpABE.mpk and $\mathbf{x}_i$, using random coins $\mathsf{PRF}(\mathsf{sd}, i)$. Correspondingly, given a policy C and a message μ as input, the encryptor samples a PRF seed sd and a fresh key pair $\mathsf{kpABE}.(\mathsf{mpk}, \mathsf{msk})$, and generates a sRFE ciphertext encrypting $(\mu, \mathsf{sd}, \mathsf{kpABE.mpk})$. Moreover, it uses kpABE.msk to generate a kpABE secret key with respect to the policy C. The decryptor first decrypts the sRFE ciphertext to obtain a kpABE ciphertext which can subsequently be decrypted using the kpABE secret key for C. We summarize the construction as follows.

Construction 2.4 (CP-sRABE for Unbounded Depth Circuits – Informal)**.**

$\mathsf{Setup}(1^\lambda)$**:** Run and output $\mathsf{crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda)$.

$\mathsf{Gen}(\mathsf{crs})$**:** Run and output $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{sRFE.Gen}(\mathsf{crs})$.

$\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathbf{x}_i)\}_{i\in[L]})$**:** Run and output

$$\big(\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]}\big) \leftarrow \mathsf{sRFE.Agg}\big(\mathsf{crs}, \{(\mathsf{pk}_i, C_{\mathsf{reg}}[i, \mathbf{x}_i])\}_{i\in[L]}\big)\,,$$

where $C_{\mathsf{reg}}[i, \mathbf{x}_i]$ is defined as

$$C_{\mathsf{reg}}[\mathbf{x}_i](\mu, \mathsf{sd}, \mathsf{kpABE.mpk}) := \mathsf{kpABE.Enc}\big(\mathsf{kpABE.mpk}, \mathbf{x}_i, \mu; \mathsf{PRF}(\mathsf{sd}, i)\big)\,.$$

$\mathsf{Enc}(\mathsf{mpk}, C, \mu)$**:** Sample $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$ and compute $\mathsf{ct} = (\mathsf{kpABE.sk}_C, \mathsf{sRFE.ct})$ as follows:

$$\begin{aligned}\mathsf{kpABE}.(\mathsf{mpk}, \mathsf{msk}) &\leftarrow \mathsf{kpABE.Setup}(1^\lambda)\\ \mathsf{kpABE.sk}_C &\leftarrow \mathsf{kpABE.KeyGen}(\mathsf{kpABE.msk}, C)\\ \mathsf{sRFE.ct} &\leftarrow \mathsf{sRFE.Enc}\big(\mathsf{sRFE.mpk}, (\mu, \mathsf{sd}, \mathsf{kpABE.mpk})\big)\,.\end{aligned}$$

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, \mathbf{x}_{i^*}, \mathsf{ct} = (\mathsf{kpABE.sk}_C, \mathsf{sRFE.ct}))$**:** Output μ as follows:

$$\begin{aligned}\mathsf{kpABE.ct} &\leftarrow \mathsf{sRFE.Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, C_{\mathsf{reg}}[i^*, \mathbf{x}_{i^*}], \mathsf{sRFE.ct})\\ \mu &\leftarrow \mathsf{kpABE.Dec}(\mathsf{kpABE.sk}_C, C, \mathsf{kpABE.ct}, \mathbf{x}_i)\,.\end{aligned}$$

For correctness, we observe that sRFE decryption yields a kpABE ciphertext of μ generated with respect to the attribute $\mathbf{x}_{i^*}$. Subsequently, this kpABE ciphertext is decrypted using a secret key for C which is included in ct and was generated using the same kpABE master key pair. Thus, if $C(\mathbf{x}_{i^*}) = 0$, kpABE decryption correctly recovers μ. For security, we first rely on the INDr security of kpABE to conclude that kpABE.ct is pseudorandom for every admissible adversary. Therefore, we can invoke the prCT security of sRFE to conclude that sRFE.ct is also pseudorandom which is the only component of ct depending on the challenge.

We can instantiate kpABE straightforwardly with e.g. the bounded depth KP-ABE of BGG^+, in which case we also obtain a CP-sRABE for bounded depth circuits. However, to our knowledge there is currently no INDr secure KP-ABE scheme for unbounded depth circuits in the literature. As we show in Section 5.7, Construction 2.4 can be generalized to KP-ABEs providing a slightly weaker security notion, where not the entire ciphertext must be pseudorandom but only the part depending on the message μ. This more general construction can be instantiated with AKY's KP-ABE for unbounded depth circuits yielding a CP-sRABE for unbounded depth circuits.

Achieving (Nearly) Optimal Parameters. Achieving asymptotically optimal parameters is a central question in the construction of ABE schemes that has received a lot of attention in the classical setting [JLL23, Wee24, AKY24b]. It is therefore natural to study the same problem in the registration-based setting. Ideally, we wish to obtain crs, mpk, $\{\mathsf{hsk}_i\}_i$ and ct such that their sizes do neither depend on the length of the attribute $\mathbf{x}$ nor the size of the circuits $\{C_i\}_i$. In addition, we measure efficiency with respect to the number L of users. To our knowledge, there are no tight lower bounds in the existing literature that would clarify what the minimal (a.k.a. optimal) dependency on L is, from a theoretical perspective. In our constructions, $|\mathsf{crs}|$ is independent of L, and $|\mathsf{mpk}|$, $|\mathsf{hsk}_i|$ and $|\mathsf{ct}|$ contain a (single) factor of $\log L$ each. While these values seem acceptable, we do not know if they are optimal or if it is theoretically possible to reduce the dependency on L even further.

Regardless of these $\log L$ factors, let us focus on eliminating the dependency on the length of the attributes and the size of the circuits. For the sake of generality, we attempt to optimize the parameters of the sRFE in Construction 2.2, while noting that any efficiency improvement for that scheme directly translates into analogous improvements in the applications. We split the input $\mathbf{x} = (\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})$ to the encryption algorithm into a public part $\mathbf{x}_{\mathsf{pub}}$ and a private part $\mathbf{x}_{\mathsf{pri}}$. Since $\mathbf{x}_{\mathsf{pub}}$ must not be hidden, we further assume that it is given to the decryption algorithm as additional input.

As a starting point, we recall that AKY present a compiler in the plain FE setting which achieves $|\mathsf{mpk}|$, $|\mathsf{sk}_C|$ and $|\mathsf{ct}|$ independent of $|\mathbf{x}_{\mathsf{pub}}|$ and $|C|$. The main ingredient of their compiler is a variant of pPRIO called *laconic* pPRIO which enables domains of the form $X = \{X_i\}_{i\in[N]}$ for arbitrary strings $X_1, \ldots, X_N \in \{0,1\}^\ell$ with arbitrary length ℓ. This gives more control on the shape of the domain compared to plain pPRIO, where input domains are always of the form $1, \ldots, N$. Syntax-wise, laconic pPRIO is a triple $\mathsf{LprIO} = (\mathsf{Digest}, \mathsf{Obf}, \mathsf{Eval})$, where Digest is an additional algorithm that, given the domain X as input, produces a short digest dig_X of X whose size does not depend on N. Obf algorithm receives as input the digest dig_X and a circuit E but not the entire description of X, and outputs an obfuscated circuit $\widehat{E}$. Finally, Eval algorithm on input the obfuscated circuit $\widehat{E}$ and the input domain $X = \{X_i\}_{i\in[N]}$ outputs the evaluation results $Y = \{Y_i = E(X_i)\}_{i\in[N]}$.

Using laconic pPRIO, we may try to build a compiler in the registration-based setting that adapts the ideas of AKY. Let us start with the public input $\mathbf{x}_{\mathsf{pub}}$. In Construction 2.2, $\mathbf{x}_{\mathsf{pub}}$ is encrypted as part of $\mathbf{x}$ and the bounded depth sRFE scheme sRFE produces input labels for $\mathbf{x}_{\mathsf{pub}}$ during decryption. Since $\mathbf{x}_{\mathsf{pub}}$ is encrypted inside the ciphertext ct, the size of ct inherently depends on $|\mathbf{x}_{\mathsf{pub}}|$. In other words, to achieve $|\mathsf{ct}|$ independent of $|\mathbf{x}_{\mathsf{pub}}|$, we must avoid encrypting $\mathbf{x}_{\mathsf{pub}}$. For this, we may consider the use of laconic pPRIO. Specifically, we let the encryptor generate a short digest $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}$ of $\mathbf{x}_{\mathsf{pub}}$ (whose size is independent of $|\mathbf{x}_{\mathsf{pub}}|$) and include only $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}$ into the sRFE ciphertext. In this way, the input length of sRFE can be made independent of the length of $\mathbf{x}_{\mathsf{pub}}$. On the negative side, since $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}$ is too short to contain an entire description of $\mathbf{x}_{\mathsf{pub}}$, we cannot use sRFE to produce the input labels $\{\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pub}}[k]}\}_{k\in[|\mathbf{x}_{\mathsf{pub}}|]}$ for $\mathbf{x}_{\mathsf{pub}}$ anymore. However, we still can use sRFE to generate an *obfuscation* of a circuit E_{pub} which, on input $(k, \mathbf{x}_{\mathsf{pub}}[k])$, generates the corresponding label $\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pub}}[k]}$. This is possible since the laconic obfuscation algorithm takes only $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}$ as input, but does not need $\mathbf{x}_{\mathsf{pub}}$ itself. At the time of decryption, sRFE outputs an obfuscation of E_{pub} which can be evaluated to obtain the input labels corresponding to $\mathbf{x}_{\mathsf{pub}}$. After this additional step, we can run bGC evaluation as before in Construction 2.2. We note that E_{pub} can be implemented by a circuit of fixed polynomial size due to the decomposability property of bGC. Therefore, neither the input length nor the circuit size of sRFE depend on $\mathbf{x}_{\mathsf{pub}}$ anymore, as desired.

Next, we may hope that a similar argument helps to remove the dependency on the circuit size as well.

AKY demonstrate that this can work in the case of plain FE. A natural approach is to change the definition of $C_{\mathsf{reg}}[C_i]$ such that it does not generate the actual garbled gates $\{\widetilde{C}_i^{(k)}\}_{k\in[|C_i|]}$ anymore, but only an obfuscation of a circuit E_{cir} that, on input $(k, C_i^{(k)})$, generates the corresponding garbled gate $\widetilde{C}_i^{(k)}$. For this idea to work out, the circuit $C_{\mathsf{reg}}[C_i]$ needs to hardwire a digest specifying the input domain $\{(k, C_i^{(k)})\}_{k\in[|C_i|]}$ of E_{cir}. This is problematic since the digest generation in the laconic pPRIO of AKY is performed by a *randomized* algorithm, thus cannot be carried out by the deterministic aggregator. We note that this problem is particular to the registration-based setting since the key generation algorithm in classical FE can of course be randomized and thus is able to perform the digest computation.

To make the idea work for registered FE, we need to construct a laconic pPRIO scheme with a deterministic Digest algorithm. Without further changes to the model, this seems quite challenging. We therefore modify the laconic pPRIO primitive as introduced in AKY by adding a one-time global setup that produces reusable randomness in the form of a CRS. This leads to our new primitive called *laconic pPRIO with global setup*, which extends the definition of (plain) laconic pPRIO with an additional Setup algorithm that on input 1^λ outputs crs which is later given to all other algorithms as additional input. In the context of registration-based encryption, this change feels extremely natural since sRFE itself crucially relies on such a setup. Relying on this additional algorithm, we are able to adapt the construction of laconic pPRIO by AKY and derandomize the Digest algorithm. More specifically, we first introduce a stronger form of blind batch encryption (BBE) that we coin *unbounded* BBE with *adaptive* security. Using this stronger BBE notion, we can obtain the modified laconic pPRIO primitive needed in the registration-based setting. For the technical details on BBE and laconic pPRIO with global setup, please see Sections A and B (and specifically Sections A.1 and B.1 for their respective technical overviews).

We sketch our construction of (nearly) optimal sRFE using a laconic pPRIO with global setup denoted by $\mathsf{LprIO} = \mathsf{LprIO}.(\mathsf{Setup}, \mathsf{Digest}, \mathsf{Obf}, \mathsf{Eval})$.

Construction 2.5 (Asymptotically Efficient sRFE for Circuits – Informal)**.**

$\mathsf{Setup}(1^\lambda)$**:** Run $\mathsf{sRFE.crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda)$ and $\mathsf{LprIO.crs} \leftarrow \mathsf{LprIO.Setup}(1^\lambda)$. Then output the common reference string $\mathsf{crs} = (\mathsf{sRFE.crs}, \mathsf{LprIO.crs})$.

$\mathsf{Gen}(\mathsf{crs})$**:** Run and output $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{sRFE.Gen}(\mathsf{crs})$.

$\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, C_i)\}_{i\in[L]})$**:** Run $\mathsf{dig}_{C_i} := \mathsf{LprIO.Digest}(\mathsf{LprIO.crs}, \{(k, C_i^{(k)})\}_{k\in[|C_i|]})$ for all $i \in [L]$ and output

$$\big(\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]}\big) \leftarrow \mathsf{sRFE.Agg}\big(\mathsf{crs}, \{(\mathsf{pk}_i, C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{C_i}])\}_{i\in[L]}\big)\,,$$

where $C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{C_i}](\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}})$ does the following.

1. Run $(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) \leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda)$ for $k \in [|\mathbf{x}_{\mathsf{pub}}|]$.
2. Run $\widehat{E}_{\mathsf{pub}} \leftarrow \mathsf{LprIO.Obf}(\mathsf{LprIO.crs}, \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, E_{\mathsf{pub}})$, where E_{pub} takes as input $(k, \mathbf{x}_{\mathsf{pub}}[k])$, computes labels $(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) \leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda)$ and outputs $\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pub}}[k]}$.
3. Run $\widehat{E}_{\mathsf{cir}} \leftarrow \mathsf{LprIO.Obf}(\mathsf{LprIO.crs}, \mathsf{dig}_{C_i}, E_{\mathsf{cir}})$, where E_{cir} takes as input $(k, C_i^{(k)})$, generates a garbled gate $\widetilde{C}_i^{(k)} \leftarrow \mathsf{bGC.Garble}_k(1^\lambda, C_i^{(k)})$ and outputs $\widetilde{C}_i^{(k)}$.
4. Output $(\mathsf{lab}_{\mathbf{x}_{\mathsf{pri}}} = \{\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pri}}[k]}\}_{k\in[|\mathbf{x}_{\mathsf{pri}}|]}, \widehat{E}_{\mathsf{pub}}, \widehat{E}_{\mathsf{cir}})$.

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{x})$**:** Generate a digest $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} \leftarrow \mathsf{LprIO.Digest}(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[|\mathbf{x}_{\mathsf{pri}}|]})$ and output a sRFE ciphertext $\mathsf{ct} \leftarrow \mathsf{sRFE.Enc}(\mathsf{mpk}, (\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}))$.

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, C_{i^*}, \mathsf{ct}, \mathbf{x}_{\mathsf{pub}})$**:** Run

$$\begin{aligned}
(\mathsf{lab}_{\mathbf{x}_{\mathsf{pri}}}, \widehat{E}_{\mathsf{cir}}, \widehat{E}_{\mathsf{pub}}) &\leftarrow \mathsf{sRFE.Dec}\big(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{C_{i^*}}], \mathsf{ct}\big)\\
\mathsf{lab}_{\mathbf{x}_{\mathsf{pub}}} = \{\mathsf{lab}_k\}_k &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_k, \widehat{E}_{\mathsf{pub}}\big)\\
\widetilde{C}_{i^*} = \{\widetilde{C}_{i^*}^{(k)}\}_{k\in[|C_{i^*}|]} &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, C_{i^*}^{(k)})\}_{k\in[|C_{i^*}|]}, \widehat{E}_{\mathsf{cir}}\big)
\end{aligned}$$

and output $z \leftarrow \mathsf{bGC.Eval}(\widetilde{C}_{i^*}, \mathsf{lab}_{\mathbf{x}_{\mathsf{pub}}}, \mathsf{lab}_{\mathbf{x}_{\mathsf{pri}}})$.

Correctness of the scheme follows straightforwardly from the correctness of the ingredients. For efficiency, we can observe that both the input length and size of the registered circuits $C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$ are bounded by $\mathsf{poly}(\lambda, |\mathbf{x}_{\mathsf{pri}}|)$. Noting that the circuit size in particular bounds its depth, we conclude that the sRFE scheme in Construction 2.5 has the following parameters:

$$|\mathsf{crs}| = \mathsf{poly}(|\mathbf{x}_{\mathsf{pri}}|, \lambda) \qquad |\mathsf{mpk}| = \log\log L + \mathsf{poly}(|\mathbf{x}_{\mathsf{pri}}|, \lambda)$$
$$|\mathsf{hsk}_i| = \log L \cdot \mathsf{poly}(|\mathbf{x}_{\mathsf{pri}}|, \lambda) \qquad |\mathsf{ct}| = \log L \cdot \mathsf{poly}(|\mathbf{x}_{\mathsf{pri}}|, \lambda)\,.$$

Using the hybrid encryption framework [AKY24b], we can (1) completely remove the dependency on $|\mathbf{x}_{\mathsf{pri}}|$ from $|\mathsf{crs}|$, $|\mathsf{mpk}|$, $|\mathsf{hsk}_i|$ and (2) reduce the size of ct to an additive dependency on $|\mathbf{x}_{\mathsf{pri}}|$. This is asymptotically optimal since the ciphertext size inherently depends on the length of the private input which it is supposed to hide. The security proof proceeds in several steps.

- First, invoking the prCT security of sRFE, it suffices to show that the output of the registered circuit $C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{C_i}](\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}) = (\widehat{E}_{i,\mathsf{pub}}, \widehat{E}_{i,\mathsf{cir}}, \{\mathsf{lab}_{i,k,\mathbf{x}_{\mathsf{pri}}[k]}\}_{k\in[|\mathbf{x}_{\mathsf{pri}}|]})$ is pseudorandom for $i \in \mathcal{C}$.
- Second, the security of LprIO guarantees that $\widehat{E}_{i,\mathsf{pub}}$ and $\widehat{E}_{i,\mathsf{cir}}$ are pseudorandom if their respective outputs $\mathsf{lab}_{\mathbf{x}_{\mathsf{pub}}}$ and $\widetilde{C}_{i^*}$ are pseudorandom.
- Third, we use the bGC simulator to generate the garbling, i.e.,

$$(\widetilde{C}_{i^*}, \mathsf{lab}_{\mathbf{x}_{\mathsf{pub}}}, \mathsf{lab}_{\mathbf{x}_{\mathsf{pri}}}) \leftarrow \mathsf{bGC.Sim}\big(1^\lambda, C_{i^*}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})\big)\,.$$

- Finally, we use the condition from the pre-game to conclude that $C_{i^*}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})$ is pseudorandom, so we can rely on the blindness of bGC to replace the output of the simulator with random strings.

The above scheme and security proof provides a high-level overview, though there are several details missing. One aspect we have not paid attention to is the fact that the circuits $C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$, E_{pub} and E_{cir} run randomized algorithms. Therefore, we need to include several PRF values in our construction to make sure that all algorithms receive (pseudo-)random coins. Furthermore, Construction 2.5 makes use of an ideal instantiation of LprIO. In reality, we only have a weaker version, where not the entire obfuscated circuit is pseudorandom. This requires a careful modification of the construction and an adjustment of the security notions. We skip the details as related issues also appear in AKY and we handle them in a similar fashion. Putting it all together, we can state the following theorem.

Theorem 2.6 ((Nearly) Optimal sRFE for Circuits – Informal)**.** *Assuming LWE and prFE, there exists a* prCT *secure sRFE scheme supporting unbounded depth circuits with the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log\log L + \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = \log L \cdot \mathsf{poly}(\lambda) + |\mathbf{x}_{\mathsf{pri}}|\,.$$

Combining Theorem 2.6 with the compilers in Constructions 2.3 and 2.4, we obtain the following corollary.

Corollary 2.7 ((Nearly) Optimal sRABE and sRPE)**.** *Assuming LWE and prFE, there exist selectively secure KP-sRABE, CP-sRABE and sRPE schemes supporting unbounded depth circuits with the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log\log L + \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = \log L \cdot \mathsf{poly}(\lambda) + |\mu| + |\mathbf{x}|\,.$$

Furthermore, we can apply the powers-of-two transformation to lift our constructions to the (non-slotted) RFE and RABE models, giving us the statements of Theorems 1.1 and 1.2 from the introduction.

2.3 Pseudorandom RFE and RABE for Turing Machines

The rest of the technical overview is dedicated to the description of our constructions for Turing machines.

Bundling Functionalities. A popular approach to deal with uniform models of computation uses so-called bundling functionalities as an intermediate step [GKW16]. For a functionality $\mathcal{F}_\ell$ where the input length ℓ is specified at setup (e.g., a circuit class), the bundling functionality $\mathcal{F}^{\mathsf{bndl}}$ describes a family of functionalities under one set of public parameters where the input length is only specified by the encryption algorithm. There exist various compilers that turn a FE scheme FE for $\mathcal{F}$ into an FE scheme $\mathsf{FE}^{\mathsf{bndl}}$ for $\mathcal{F}^{\mathsf{bndl}}$ [GKW16, AMY19a, AMVY21, AKY24a]. The key idea behind all these constructions is to delay the execution of $\mathsf{FE.Setup}$ and only perform it during $\mathsf{FE}^{\mathsf{bndl}}.\mathsf{KeyGen}$ instead of $\mathsf{FE}^{\mathsf{bndl}}.\mathsf{Setup}$. If we aimed for a similar compiler in the registration-based model, the natural choice would be to run $\mathsf{FE.Setup}$ during aggregation. But this is impossible since this step is performed by the deterministic and fully transparent key aggregator. Therefore, the idea of approaching uniform models through bundling functionalities seems incompatible with the registration-based setting.

Our Approach. As an alternative pathway, we try to directly generalize the previous construction of sRFE for unbounded depth circuits. We note that it suffices to construct a sRFE where only public inputs have unbounded length while private inputs are still bounded. Indeed, such a scheme can immediately be lifted to unbounded-length private inputs using the hybrid encryption framework.

Let us begin by observing at which point Construction 2.5 fails for unbounded length inputs $\mathbf{x}_{\mathsf{pub}}$. On the positive side, the encryption algorithm still works: thanks to the power of laconic pPRIO to generate a digest $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}$ of *fixed* size $\mathsf{poly}(\lambda)$, the input length of the registered circuit $C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{C_i}](\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}})$ does not change if $\mathbf{x}_{\mathsf{pub}}$ grows. Thus, encryption can be performed as before, regardless of the length of $\mathbf{x}_{\mathsf{pub}}$. The problem occurs during aggregation. In Construction 2.5, the aggregator generates a digest dig_{C_i} and hard-wires it in the circuit $C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$. During decryption, this digest is used to generate an obfuscation of C_i which can later be evaluated on the input $\mathbf{x} = (\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})$. But if the length of $\mathbf{x}_{\mathsf{pub}}$ is unknown at the time of aggregation, then a digest dig_{C_i} for a single circuit C_i is not sufficient anymore. Instead, we would need one circuit $T_\ell[M]$ (and thus one digest dig_{T_ℓ}) for each input length $\ell = |\mathbf{x}|$, where $T_\ell[M]$ simulates the evaluation of M on inputs of length ℓ.

As we cannot hardwire digests for all (unbounded) lengths of $\mathbf{x}_{\mathsf{pub}}$ in the description of C_{reg}, we need a mechanism to generate the digest for a particular input length "on the fly" during decryption. It may seem natural to include the digest generation in the circuit C_{reg} itself, where we recall that C_{reg} is the circuit that is registered in the underlying sRFE scheme sRFE for bounded depth circuits. In this way, we can generate the digest corresponding to the correct length of the current input while evaluating C_{reg}. However, this renders the use of laconic pPRIO rather useless since C_{reg} now needs to hardwire the entire description of the Turing machine M – and the goal of using laconic pPRIO in the first place was to avoid exactly this.

Nevertheless, the idea of performing the generation of the digest by a circuit during decryption turns out to be the right approach. The only issue is that we cannot hardwire M in the circuit C_{reg} which is registered in sRFE. Let us therefore separate the two tasks of

- generating the digest for $T_\ell[M]$ with respect to a particular input length ℓ and the obfuscation of a gate-wise garbling of $T_\ell[M]$, and
- the obfuscation of a bit-wise garbling of an input $\mathbf{x}$ of length ℓ

into two circuits $C_{\mathsf{tm}}[M]$ and C_{reg}, respectively. Since the garbling of $\mathbf{x}$ is performed bit-wise, we can easily use sRFE for the obfuscation by registering C_{reg} during the aggregation phase of sRFE (note that this is exactly the same as in the case of unbounded depth circuits). On the other hand, as mentioned above, the digest generation for $T_\ell[M]$ requires a description of M, so we cannot directly register $C_{\mathsf{tm}}[M]$.

On the positive side, however, we can observe that contrarily to C_{reg}, $C_{\mathsf{tm}}[M]$ does not need the actual input $\mathbf{x}$. Indeed, the digest generation for $T_\ell[M]$ only needs knowledge of the input length ℓ which can be encoded in a binary string of *fixed* length λ. This is crucial as it allows us to use a single circuit C_{tm} with fixed input length λ to generate the digests dig_{T_ℓ} for all possible input lengths ℓ. While this observation still does not help us to directly register the circuit $C_{\mathsf{tm}}[M]$ in sRFE, it does enable the generation of a digest dig_M of $C_{\mathsf{tm}}[M]$

during aggregation such that hopefully $C_{\mathsf{tm}}[M]$ can be evaluated later during decryption. Importantly, we note that being able to encode M in a digest dig_M during *aggregation* saves us from registering a circuit whose size depends on M.

It remains the question of how to connect the circuits $C_{\mathsf{tm}}[M]$ and C_{reg}. Intuitively, we want C_{reg} to perform the evaluation of the circuit $C_{\mathsf{tm}}[M]$ on a particular input without depending on the size of $C_{\mathsf{tm}}[M]$.[8] Fortunately, we already have a solution for this: laconic pPRIO. Indeed, to establish the missing link, the aggregator can generate the succinct digest dig_M of $C_{\mathsf{tm}}[M]$ and hardwire it in the description of C_{reg}. Then, C_{reg} does not evaluate $C_{\mathsf{tm}}[M]$ by itself but only provides an obfuscation of a gate-wise garbling of $C_{\mathsf{tm}}[M]$ which in turn provides a gate-wise obfuscation of a circuit $T_\ell[M]$ that simulates the evaluation of M on an input of length ℓ. We summarize the construction below.

Construction 2.8 (Asymptotically Efficient sRFE for TMs – Informal)**.**

$\mathsf{Setup}(1^\lambda)$**:** Run $\mathsf{sRFE.crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda)$, $\mathsf{LprIO.crs} \leftarrow \mathsf{LprIO.Setup}(1^\lambda)$. Output $\mathsf{crs} = (\mathsf{sRFE.crs}, \mathsf{LprIO.crs})$.

$\mathsf{Gen}(\mathsf{crs})$**:** Run and output $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{sRFE.Gen}(\mathsf{crs})$.

$\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, M_i)\}_{i\in[L]})$**:** Run

$$\mathsf{dig}_{M_i} := \mathsf{LprIO.Digest}\big(\mathsf{LprIO.crs}, \big\{(k, C^{(k)}_{\mathsf{tm},i})\big\}_{k\in[|C_{\mathsf{tm},i}|]}\big)\,,$$

where $C_{\mathsf{tm},i} = C_{\mathsf{tm}}[\mathsf{LprIO.crs}, M_i](\ell)$ is defined as follows:

1. Run $\mathsf{dig}_{T_i} \leftarrow \mathsf{LprIO.Digest}(\mathsf{LprIO.crs}, \{k, T^{(k)}_i\}_{k\in[|T|]})$, where $T_i = T_\ell[M_i]$ denotes a circuit which simulates the execution of M_i on inputs of length ℓ.
2. Run $\widehat{E}_{\mathsf{cir}} \leftarrow \mathsf{LprIO.Obf}(\mathsf{LprIO.crs}, \mathsf{dig}_{T_i}, E_{\mathsf{cir}})$ and output $\widehat{E}_{\mathsf{cir}}$.

Output

$$\big(\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]}\big) \leftarrow \mathsf{sRFE.Agg}\big(\mathsf{crs}, \big\{(\mathsf{pk}_i, C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{M_i}])\big\}_{i\in[L]}\big)\,,$$

where $C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{M_i}](\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}})$ does the following.

1. Run $(\mathsf{lab}'_{k,0}, \mathsf{lab}'_{k,1}) \leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda)$ for $k \in [\lambda]$.
2. Run $(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) \leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda)$ for $k \in [|\mathbf{x}_{\mathsf{pri}}|]$.
3. Run $\widehat{E}'_{\mathsf{cir}} \leftarrow \mathsf{LprIO.Obf}(\mathsf{LprIO.crs}, \mathsf{dig}_{M_i}, E_{\mathsf{cir}})$, where E_{cir} takes as input $(k, C^{(k)}_i)$, computes a garbled gate $\widetilde{C}^{(k)}_i \leftarrow \mathsf{bGC.Garble}_k(1^\lambda, C^{(k)}_i)$ and outputs $\widetilde{C}^{(k)}_i$.
4. Run $\widehat{E}_{\mathsf{pub}} \leftarrow \mathsf{LprIO.Obf}(\mathsf{LprIO.crs}, \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, E_{\mathsf{pub}})$, where E_{pub} takes as input $(k, \mathbf{x}_{\mathsf{pub}}[k])$, computes labels $(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) \leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda)$ and outputs $\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pub}}[k]}$.
5. Output $\big(\mathsf{lab}'_{\mathsf{bits}(\ell)} = \{\mathsf{lab}'_{k,\mathsf{bits}(\ell)[k]}\}_k, \mathsf{lab}_{\mathbf{x}_{\mathsf{pri}}} = \{\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pri}}[k]}\}_k, \widehat{E}'_{\mathsf{cir}}, \widehat{E}_{\mathsf{pub}}\big)$. Here, $\ell = |\mathbf{x}_{\mathsf{pub}}| + |\mathbf{x}_{\mathsf{pri}}|$ denotes the total input length.

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{x})$**:** Generate a digest $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} \leftarrow \mathsf{LprIO.Digest}(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_k)$ and output a sRFE ciphertext $\mathsf{ct} \leftarrow \mathsf{sRFE.Enc}(\mathsf{mpk}, (\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}))$.

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, M_{i^*}, \mathsf{ct}, \mathbf{x}_{\mathsf{pub}})$**:** Run

$$\begin{aligned}
(\mathsf{lab}'_{\mathsf{bits}(\ell)}, \mathsf{lab}_{\mathbf{x}_{\mathsf{pri}}}, \widehat{E}'_{\mathsf{cir}}, \widehat{E}_{\mathsf{pub}}) &\leftarrow \mathsf{sRFE.Dec}\big(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{M_{i^*}}], \mathsf{ct}\big)\\
\widetilde{C}_{\mathsf{tm},i^*} = \{\widetilde{C}^{(k)}_{\mathsf{tm},i^*}\}_k &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, C^{(k)}_{\mathsf{tm},i^*})\}_k, \widehat{E}'_{\mathsf{cir}}\big)\\
\widehat{E}_{\mathsf{cir}} &\leftarrow \mathsf{bGC.Eval}(\widetilde{C}_{\mathsf{tm},i^*}, \mathsf{lab}'_{\mathsf{bits}(\ell)})\\
\widetilde{T}_{i^*} = \{\widetilde{T}^{(k)}_{i^*}\}_k &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, T^{(k)}_{i^*})\}_k, \widehat{E}_{\mathsf{cir}}\big)\\
\mathsf{lab}_{\mathbf{x}_{\mathsf{pub}}} = \{\mathsf{lab}_k\}_k &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_k, \widehat{E}_{\mathsf{pub}}\big)\,,
\end{aligned}$$

[8] In more detail, C_{reg} needs to generate the gate-wise garbling of $T_\ell[M]$ inside $C_{\mathsf{tm}}[M]$ with the same random coins as the bit-wise garbling of the input $\mathbf{x}$ to obtain compatible garbled gates and garbled input bits during decryption. For simplicity, we ignore the fact that both garbling and obfuscation algorithms are randomized in this overview.

where $T_{i^*} = T_\ell[M_{i^*}]$ simulates the execution of M_{i^*} on input length ℓ. Finally, run bGC evaluation $z \leftarrow \mathsf{bGC.Eval}(\widetilde{T}_{i^*}, \mathsf{lab}_{\mathbf{x}_{\mathsf{pub}}}, \mathsf{lab}_{\mathbf{x}_{\mathsf{pri}}})$ and output $\mathbf{z}$.

For correctness, we observe that the output of the evaluation of the garbled circuit $\widetilde{E}'_{\mathsf{cir}}$ is again a garbling of the circuit $C_{\mathsf{tm},i^*} = C_{\mathsf{tm}}[\mathsf{LprIO.crs}, M_{i^*}]$. When being evaluated with respect to the labels $\mathsf{lab}'_{\mathsf{bits}(\ell)}$, $\widetilde{C}_{\mathsf{tm},i^*}$ outputs an obfuscation of the circuit $T_{i^*} = T_\ell[M_{i^*}]$ which simulates the Turing machine M_{i^*} on length-ℓ inputs. Thus, evaluating $\widetilde{T}_{i^*}$ with respect to the labels $\mathsf{lab}_{\mathbf{x}_{\mathsf{pub}}}$ and $\mathsf{lab}_{\mathbf{x}_{\mathsf{pri}}}$ yields the output of M_{i^*} on input $(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})$ as desired. For efficiency, we note that the registered circuits $C_{\mathsf{reg}}[\mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]$, for $i \in [L]$, hardwire only a digest of the Turing machine rather than an entire description of M_i. Similarly, ct encrypts only a digest of $\mathbf{x}_{\mathsf{pub}}$ rather than $\mathbf{x}_{\mathsf{pub}}$ itself. Therefore, all parameter sizes are independent of $|M_i|$ and $|\mathbf{x}_{\mathsf{pub}}|$.

The security proof follows roughly the same idea as the proof sketch of Construction 2.5 outlined above. A notable difference is that we now have to deal with nested obfuscations, where the output of one obfuscation is used to derive another obfuscation. To see why this is relevant, let us informally recall the security model of laconic pPRIO as consisting of a pre- and a post-condition and the requirement that the former implies the latter with respect to all efficient samplers. Here, the pre-condition requires pseudorandomness for the output of the *circuits* and the post-condition implies pseudorandomness of the *obfuscations*. Now, if we consider nested obfuscations, this leads to the situation where the pre-condition of the outer layer contains obfuscations of the inner layer. Therefore, we first need to leverage the security of the inner obfuscation to verify the pre-condition of the outer obfuscation. To make this sequential security proof work, we must carefully design our scheme to ensure that the inner and outer layer of the obfuscations are generated independently. We refer the reader to Section 4.10 for details and summarize our result as follows.

Theorem 2.9 ((Nearly) Optimal sRFE for Turing Machines – Informal)**.** *Assuming LWE and prFE, there exists a* prCT *secure sRFE scheme supporting Turing machines with the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log\log L + \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = \log L \cdot \mathsf{poly}(\lambda) + |\mathbf{x}_{\mathsf{pri}}| \,.$$

Combining this theorem with the powers-of-two compiler, we obtain Theorem 1.3 stated in the introduction.

More Implications. The sRABE and sRPE schemes in Constructions 2.3 and 2.4 naturally extend to the case of Turing machines. In this way, we obtain KP-sRABE, CP-sRABE and sRPE schemes supporting Turing machines. In particular, Construction 2.4 (CP-sRABE) requires a classical KP-ABE scheme for Turing machines. As an instantiation, we could use the recently proposed construction by AKY. However, their scheme does not have optimal parameters which would also ruin the parameters of our CP-sRABE. We therefore first address the task of constructing a KP-ABE for Turing machines with optimal parameters. To this end, we observe that our ideas to build prCT secure sRFE for Turing machines equally work in the case of classical FE, giving us a prCT secure FE for Turing machines with optimal parameters. This scheme can in turn be leveraged to obtain KP-ABE, CP-ABE and PE for Turing machines with optimal parameters, as demonstrated by AKY in the case of circuits.

Theorem 2.10 (Optimal FE for Turing Machines – Informal)**.** *Assuming LWE and prFE, there exists a* prCT *secure FE scheme for Turing machines with the following efficiency parameters:*

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda) \,, \qquad |\mathsf{sk}_M| = \mathsf{poly}(\lambda) \,, \qquad |\mathsf{ct}| = \mathsf{poly}(\lambda) + |\mathbf{x}_{\mathsf{pri}}| \,.$$

As an implication, there exist KP-ABE, CP-ABE and PE schemes for Turing machines with the same efficiency parameters (where we note that the private input in case of PE also includes the attribute).

Please see Section 7 for details about our constructions in the plain FE model. Combining Theorems 2.9 and 2.10 with the compilers in Constructions 2.3 and 2.4, we obtain the following corollary.

Corollary 2.11 ((Nearly) Optimal sRABE and sRPE)**.** *Assuming LWE and prFE, there exist selectively secure KP-sRABE, CP-sRABE and sRPE schemes supporting unbounded depth circuits with the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log\log L + \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = \log L \cdot \mathsf{poly}(\lambda) + |\mu| + |\mathbf{x}| \ .$$

Applying the powers-of-two compiler yields Theorem 1.4 as stated in the introduction.

3 Preliminaries

In this section, we present the preliminaries necessary to understand this work.

3.1 Notational Conventions

Let $\lambda \in \mathbb{N}$ be the security parameter. Except in the definitions, we will suppress λ in subscripts for brevity. A nonnegative function $\varepsilon\colon \mathbb{N} \to \mathbb{R}$ is negligible if $\varepsilon(\lambda) = O(\lambda^{-n})$ for all $n \in \mathbb{N}$. An algorithm is said to be *efficient* if it runs in probabilistic polynomial time (PPT) in the security parameter.

To avoid confusion, we always write vectors $\mathbf{v}$ and matrices $\mathbf{A}$ in boldface and use uppercase letters for the latter. Scalars s are written in italics. We strictly follow the convention of all vectors $\mathbf{v}$ being column vectors. The corresponding row vector is denoted by $\mathbf{v}^\top$. We denote the i-th unit vector in $\mathbb{Z}_q^n$ by $\mathbf{e}_i^{(n)}$ and sometimes omit the superscript if n is clear from the context. Given an object x, we write $\mathsf{bits}(x)$ for its fixed-length bit representation arranged in a row, i.e., a matrix in $\{0,1\}^{1\times N}$ for some N.

Security Experiments and Distributions. Let $\mathbf{Exp}$ be an interactive *experiment* that interacts with an algorithm $\mathcal{A}$ (called the *adversary*), depends on the security parameter λ and has binary outcome. We also refer to such objects as *games* or *hybrids*. We let "$\mathbf{Exp}_{\mathcal{A}}(1^\lambda) \to 1$" denote the event that the outcome of running $\mathbf{Exp}$ with $\mathcal{A}$ on security parameter λ is 1. For two experiments $\mathbf{Exp}^0$ and $\mathbf{Exp}^1$, we define the distinguishing advantage of $\mathcal{A}$ against the tuple $(\mathbf{Exp}^0, \mathbf{Exp}^1)$ as

$$\mathcal{A}_{\mathbf{Exp}^0,\mathbf{Exp}^1,\mathcal{A}}(\lambda) := \left|\Pr\left[\mathbf{Exp}^1_{\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathbf{Exp}^0_{\mathcal{A}}(1^\lambda) \to 1\right]\right| \ .$$

We write $\mathbf{Exp}^0 \approx_c \mathbf{Exp}^1$ if the experiments are *computationally indistinguishable, i.e.* their distinguishing advantage is negligible for all efficient adversaries $\mathcal{A}$. We write $\mathbf{Exp}^0 \approx_s \mathbf{Exp}^1$ if the experiments are *statistically indistinguishable, i.e.* their distinguishing advantage is negligible for all (even unbounded) adversaries. We write $\mathbf{Exp}^0 \equiv \mathbf{Exp}^1$ if the experiments are *identically distributed, i.e.* their distinguishing advantage is 0 for all (even unbounded) adversaries. By default, the term *indistinguishable* refers to computational indistinguishability.

More general, the same notations can be used for sequences of distributions. Let $D^0 = \{D^0_\lambda\}_{\lambda\in\mathbb{N}}$ and $D^1 = \{D^1_\lambda\}_{\lambda\in\mathbb{N}}$ be two sequences of distributions. For $b \in \{0,1\}$, we define $\mathbf{Exp}^b_{\mathcal{A}}(1^\lambda)$ as follows: sample $x \xleftarrow{\$} D^b_\lambda$, run $\mathcal{A}(1^\lambda, x)$ and use the output of $\mathcal{A}$ as the outcome of the experiment. Then we write $D^0 \approx_c D^1$ (resp. $D^0 \approx_s D^1$, $D^0 \equiv D^1$) if $\mathbf{Exp}^0_{\mathcal{A}} \approx_c \mathbf{Exp}^1_{\mathcal{A}}$ (resp. $\mathbf{Exp}^0_{\mathcal{A}} \approx_s \mathbf{Exp}^1_{\mathcal{A}}$, $\mathbf{Exp}^0_{\mathcal{A}} \equiv \mathbf{Exp}^1_{\mathcal{A}}$).

Sets and Indexing. We denote by $\mathbb{Z}$ and $\mathbb{N}$ the sets of integers and natural numbers (positive integers). For integers m and n, we write $[m;n]$ to denote the set $\{z \in \mathbb{Z} : m \le z \le n\}$ and let $[n] := [1;n]$. For a prime number q, $\mathbb{Z}_q$ denotes the finite field of integers modulo q.

To index a vector or the columns of a matrix, we write $\mathbf{v}[i]$ and $\mathbf{A}[j]$. In contrast, objects of some collection that is not regarded as a vector or matrix are indexed using subscripts (or superscripts in some cases). For instance, $\mathbf{v}_i$ represents a vector, not a component of some vector. If i runs through some index set $[n]$, it means that there are n vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$. If the n objects are scalars (or not explicitly vectors), we will write $v_1, \ldots, v_n$ instead.

3.2 Computational Models

Circuits. We let $\mathcal{C}_{\ell_{\mathsf{in}},\ell_{\mathsf{dep}},\ell_{\mathsf{out}}}$ denote the class of functions that can be computed by a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ of depth at most ℓ_{dep}. We may omit the indices ℓ_{dep} and/or ℓ_{out}. In the former case the depth of the circuits is unbounded, in the latter case the default value for ℓ_{out} is 1. Furthermore, in our constructions we may split the input space as $\{0,1\}^{\ell_{\mathsf{pub}}} \times \{0,1\}^{\ell_{\mathsf{pri}}}$. In this case, we replace the index ℓ_{in} with the tuple $(\ell_{\mathsf{pri}}, \ell_{\mathsf{pub}})$.

Given a circuit C, we write $C^{(k)}$ to refer to a description of its k-th gate and let $|C|$ denote the total number of gates in C. We note that the description of $C^{(k)}$ can be encoded in a binary string of length 4λ as it is sufficient to encode its index, two indices of the incoming wires and the type of the gate.

Turing Machines. We recall here the definition of a Turing machine (TM) and some useful lemmata that are taken verbatim from [AKY24a]. The definition considers Turing machines with two tapes, input tape and working tape.

Definition 3.1 (Turing Machine)**.** A (deterministic) Turing machine (TM) M is represented by the tuple $M = (Q, \delta, F)$, where Q is the number of states (we use $[Q]$ as the set of states and 1 is the initial state), $F \subset [Q]$ is the set of accepting states and

$$\delta\colon [Q] \times \{0,1\} \times \{0,1\} \to [Q] \times \{0,1\} \times \{0,\pm 1\} \times \{0,\pm 1\}$$
$$(q, b_1, b_2) \mapsto (q', b_2', \Delta i, \Delta j)$$

is the state transition function, which given the current state q, the symbol b_1 under scan on the input tape, and the symbol b_2 under scan on the work tape, specifies the new state q', the symbol b_2' overwriting b_2, the direction Δi to which the input tape pointer moves, and the direction Δj to which the work tape pointer moves. The machine is required to hand (instead of halting) once it reaches an accepting state, i.e., for all $q \in [Q]$ such that $q \in F$ and $b_1, b_2 \in zo$, it holds that $\delta(q, b_1, b_2) = (q, b_2, 0, 0)$.

For input length $n \geq 1$ and space complexity bound $s > 1$, the set of internal configurations of M is

$$\mathcal{Q}_{M,n,s} = [n] \times [s] \times \{0,1\}^s \times [Q],$$

where $(i, j, W, q) \in \mathcal{Q}_{M,n,s}$ specifies the input tape pointer $i \in [n]$, the work tape pointer $j \in [s]$, the content of the work tape $W \in \{0,1\}^s$ and the machine state $q \in [Q]$. For any bit string $x \in \{0,1\}^n$ such that $n \geq 1$ and time/space complexity bounds $t, s > 1$, the machine M accepts x within time t and space s if there exists a sequence of internal configurations (computation path of t steps) $c_0, \ldots, c_t \in \mathcal{Q}_{M,n,s}$ with $c_k = (i_k, j_k, W_k, q_k)$ such that $(i_0, j_0, W_0, q_0) = (1, 1, 0^s, 1)$ (initial configuration), and

$$\text{for all } 0 \leq k < t\colon \begin{cases} \delta(q_k, x[i_k], W_k[j_k]) = (q_{k+1}, W_{k+1}[j_k], i_{k+1} - i_k, j_{k+1} - j_k) \\ W_{k+1}[j] = W_k[j] \quad \text{for all } j \neq j_k \quad \text{(valid transitions)}; \end{cases}$$

where $x[i]$ is the i-th bit of the string x and $W_k[j]$ is the j-th bit of the string W_k and $q_t \in F$ (accepting). We also say M accepts within time t (without space bound) if M accepts x within time t and space $s = t$.

Next, we define time/space bounded computation with *non-deterministic* Turing machines. The definition is the same as Section 3.2, except for the following changes:

- The transition criterion δ can be any relation between (i.e., any subset of the Cartesian product of) $[Q] \times \{0,1\}^2$ and $[Q] \times \{0,1\} \times \{0,\pm q\}^2$, where $((q, b_1, b_2), (q', b_2', \Delta i, \Delta j)) \in \delta$ means that if the current state is q, the input tape symbol under scan is b_1 and the work tape symbol under scan is b_2, then it is valid to transit into state q', overwrite b_2 with b_2', and move the input and work tape pointer by offset Δi and Δj, respectively.
- The definition of hanging in accepting state is that for all $q \in [Q]$ such that $q \in F$ and for all $b_1, b_2 \in \{0,1\}$,

$$\delta \cap \left(\{(q, b_1, b_2)\} \times ([Q] \times \{0,1\} \times \{0,\pm 1\}^2)\right) = \{(q, b_1, b_2), (q, b_2, 0, 0)\}.$$

- In the definition of acceptance

$$\delta(q_k, x[i_k], W_k[j_k]) = (q_{k+1}, W_{k+1}[j_k], i_{k+1} - i_k, j_{k+1} - j_k)$$

  is changed to

$$((q_k, x[i_k], W_k[j_k]), (q_{k+1}, W_{k+1}[j_k], i_{k+1} - i_k, j_{k+1} - j_k)) \in \delta.$$

The following lemma can be obtained by a simple argument on emulating Turing machine on Boolean circuits.

Lemma 3.2 (Emulating a Turing Machine on Circuit)**.** *Consider a circuit that takes as input a description of a (deterministic) Turing machine $M = (Q, \delta, F)$, input x to M, a configuration $(i, j, W, q) \in \mathcal{Q}_{M,|x|,|W|}$ and outputs the next configuration (i', j', W', q'). We can implement such a circuit with depth $\mathsf{poly}(\log|x|, \log|W|, \log|M|)$ and size $\mathsf{poly}(|x|, |W|, |M|)$.*

We also consider the following lemma, which can be obtained by a simple observation.

Lemma 3.3 (Checking Transition for Non-deterministic Turing Machine)**.** *Consider a circuit that takes as input a description of a non-deterministic Turing machine $M = (Q, \delta, F)$, input x to M, two configurations $(i, j, W, q) \in \mathcal{Q}_{M,|x|,|W|}$ and $(i', j', W', q') \in \mathcal{Q}_{M,|x|,|W|}$, and outputs $((i, j, W, q), (i', j', W', q')) \in \delta$ or not. We can implement such a circuit with depth $\mathsf{poly}(\log|x|, \log|W|, \log|M|)$ and size $\mathsf{poly}(|x|, |W|, |M|)$.*

3.3 Lattice Preliminaries

We recall some facts about lattices that we use in the rest of the sections. Let n, m and q be integers such that $n = \mathsf{poly}(\lambda)$ and $m \geq n\lceil \log q \rceil$. In the following, let $\mathsf{SampZ}(\sigma)$ be a sampling algorithm for the discrete Gaussian distribution over $\mathbb{Z}$ with parameter $\sigma > 0$ whose support is restricted to $z \in \mathbb{Z}$ such that $|z| \leq \sqrt{n}\sigma$.

Let $\mathbf{g} = (1, 2, \ldots, 2^{\lfloor \log q \rfloor})^\top$ be the gadget vector and $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g}^\top$ be the gadget matrix. For a vector $\mathbf{v} \in \mathbb{Z}_q^n$, we write $\mathbf{G}^{-1}(\mathbf{v})$ for the m-bit vector $(\mathsf{bits}(\mathbf{v}[1]), \ldots, \mathsf{bits}(\mathbf{v}[n]))^\top$, where $\mathsf{bits}(\mathbf{v}[i])$ are m/n bits for each $i \in [n]$. The notation extends column-wise to matrices, and it holds that $\mathbf{G}\mathbf{G}^{-1}(\mathbf{V}) = \mathbf{V}$.

Next, we recall some useful lemmata.

Lemma 3.4 (Leftover Hash Lemma [DRS04, AKY24a])**.** *Let n, m, q be lattice parameters where $q > 2$ is prime. If $m \geq 2n \log q$, then for $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, $\mathbf{x} \xleftarrow{\$} \{0,1\}^m$ and $\mathbf{y} \xleftarrow{\$} \mathbb{Z}_q^n$, the statistical difference between $(\mathbf{A}, \mathbf{A}\mathbf{x})$ and $(\mathbf{A}, \mathbf{y})$ is negligible. More concretely, it is bounded by $q^n\sqrt{2^{1-m}}$.*

Lemma 3.5 (Gaussian Tail Bound)**.** *For any $\lambda \in \mathbb{N}$ and $\sigma > 0$, there exists $B \in \Theta(\sqrt{\lambda})$ such that*

$$\Pr\left[|x| > \sigma B : x \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}\right] \leq 2^{-\lambda}.$$

Lemma 3.6 (Smudging Lemma [WWW22])**.** *Let λ be a security parameter. Take any $a \in \mathbb{Z}$ such that $|a| \leq B$. If $\sigma \geq B \cdot \lambda^{\omega(1)}$, then the statistical distance between the distributions $\{z : z \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}\}$ and $\{z + a : z \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}\}$ is negligible in λ.*

Hardness Assumptions. We recall the hardness assumptions that we use in our paper.

Assumption 3.7 (LWE)**.** Let $n = n(\lambda)$, $m = m(\lambda)$ and $q = q(\lambda) > 2$ be integers and $\chi = \chi(\lambda)$ be a distribution over $\mathbb{Z}$. We say that the $\mathsf{LWE}(n, m, q, \chi)$ hardness assumption holds if for any PPT adversary $\mathcal{A}$ we have

$$\left|\Pr\left[\mathcal{A}(\mathbf{A}, \mathbf{s}^\top\mathbf{A} + \mathbf{e}^\top) \to 1\right] - \Pr\left[\mathcal{A}(\mathbf{A}, \mathbf{v}^\top) \to 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the choice of the random coins by the adversary $\mathcal{A}$ and $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\mathbf{e} \xleftarrow{\$} \chi^m$, and $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_q^m$. We also say that $\mathsf{LWE}(n, m, q, \chi)$ problem is (non-uniformly and) subexponentially hard if there exists some constant $0 < \delta < 1$ such that the above distinguishing advantage is bounded by 2^{-n^δ} for all adversaries $\mathcal{A}$ whose running time (resp. size) is 2^{n^δ}.

As shown in [Reg09, BLP+13], if we set $\chi = \mathsf{SampZ}(\sigma)$, the $\mathsf{LWE}(n,m,q,\chi)$ problem is as hard as solving worst-case lattice problems such as GapSVP or SIVP with approximation factor $\mathsf{poly}(n)\cdot(q/\sigma)$ for some $\mathsf{poly}(n)$. Since the best known algorithms for 2^k-approximation of GapSVP and SIVP run in time $2^{\tilde{O}(n/k)}$, it follows that the above $\mathsf{LWE}(n,m,q,\chi)$ with noise-to-modulus ratio 2^{-n^ϵ} is likely to be (subexponentially) hard for some constant ϵ.

3.4 GSW Homomorphic Encryption and Evaluation

We recall the (leveled) GSW fully homomorphic encryption scheme [GSW13] with syntax adapted from [HLL23].

Lemma 3.8 (GSW Scheme)**.** *The leveled GSW scheme works as follows:*

- *The keys are*

$$\textit{(public)}\quad \mathbf{A}_{\mathsf{fhe}} = \begin{bmatrix} \bar{\mathbf{A}}_{\mathsf{fhe}} \\ \bar{\mathbf{s}}^\top \bar{\mathbf{A}}_{\mathsf{fhe}} + \mathbf{e}_{\mathsf{fhe}}^\top \end{bmatrix} \in \mathbb{Z}_q^{n\times m}, \quad \textit{(secret)}\quad \mathbf{s}^\top = (\bar{\mathbf{s}}^\top, -1),$$

  where $\bar{\mathbf{s}} \in \mathbb{Z}^{n-1}$, $\bar{\mathbf{A}}_{\mathsf{fhe}} \in \mathbb{Z}_q^{(n-1)\times m}$ *and* $\mathbf{e}_{\mathsf{fhe}}^\top \in \mathbb{Z}^m$.

- *A ciphertext of* $x \in \{0,1\}$ *is* $\mathbf{X} = \mathbf{A}_{\mathsf{fhe}}\mathbf{R} - x\mathbf{G} \in \mathbb{Z}_q^{n\times m}$*, where* $\mathbf{R} \in \mathbb{Z}^{m\times m}$ *is the encryption randomness. The decryption equation is*

$$\mathbf{s}^\top \mathbf{X} = -\mathbf{e}_{\mathsf{fhe}}^\top \mathbf{R} - x\mathbf{s}^\top \mathbf{G} \in \mathbb{Z}_q^m,$$

  which can be used to extract x *via multiplication by* $\mathbf{G}^{-1}(\lfloor q/2 \rfloor \iota_n)$*, where* ι_n *is the* n*-th unit vector.*

Lemma 3.9 (Homomorphic Evaluation for Vector-Valued Functions [HLL23])**.** *There is an efficient algorithm*

$$\mathsf{MakeVEvalCkt}(1^n, 1^m, q, C) = \mathsf{VEval}_C$$

that takes as input (unary encoded) n, m*, the modulus* q *and a vector-valued circuit* $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \mathbb{Z}_q^{1\times \ell_{\mathsf{out}}}$*, and outputs a circuit*

$$\mathsf{VEval}_C(\mathbf{X}_1, \ldots, \mathbf{X}_{\ell_{\mathsf{in}}}) = \mathbf{C},$$

taking ℓ_{in} *ciphertexts as input and outputting a new ciphertext* $\mathbf{C}$ *of different format.*

- *The depth of* VEval_C *is* $d \cdot O(\log m \log\log q) + O(\log^2 \log q)$ *for* C *of depth* d*.*
- *Suppose* $\mathbf{X}_\ell = \mathbf{A}_{\mathsf{fhe}}\mathbf{R}_\ell - \mathbf{x}[\ell]\mathbf{G}$ *for* $\ell \in [\ell_{\mathsf{in}}]$ *with* $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{in}}}$*, then*

$$\mathbf{C} = \mathbf{A}_{\mathsf{fhe}}\mathbf{R}_C - \begin{bmatrix} \mathbf{0}_{(n-1)\times \ell_{\mathsf{out}}} \\ C(\mathbf{x}) \end{bmatrix} \in \mathbb{Z}_q^{n\times \ell_{\mathsf{out}}},$$

  where $\|\mathbf{R}_C^\top\| \le (m+2)^d \lceil \log q \rceil \max_{\ell\in[\ell_{\mathsf{in}}]} \|\mathbf{R}_\ell^\top\|$*. The new decryption equation is*

$$\mathbf{s}^\top \mathbf{C} = -\mathbf{e}_{\mathsf{fhe}}^\top \mathbf{R}_C + C(\mathbf{x}) \in \mathbb{Z}_q^{1\times \ell_{\mathsf{out}}}.$$

3.5 Homomorphic Evaluation Procedures

We describe the properties of attribute encoding and its homomorphic evaluation, along with the dual-use technique [BTVW17].

Lemma 3.10 (Homomorphic Evaluation for Matrix-Valued Functions [BTVW17])**.** *For ℓ_{in}-bit input $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{in}}}$, the public parameter is $\mathbf{A}_{\mathsf{att}} \in \mathbb{Z}_q^{n \times (\ell_{\mathsf{in}}+1)m}$ and the encoding is*

$$\mathbf{s}^\top (\mathbf{A}_{\mathsf{att}} - (1, \mathbf{x}^\top) \otimes \mathbf{G}) + \mathbf{e}_{\mathsf{att}}^\top,$$

where $\mathbf{s}^\top = (\bar{\mathbf{s}}^\top, -1)$ with $\bar{\mathbf{s}} \in \mathbb{Z}_q^{n-1}$ and $\mathbf{e}_{\mathsf{att}}^\top \in \mathbb{Z}^{(\ell_{\mathsf{in}}+1)m}$. There are efficient deterministic algorithms

$$\mathsf{MEvalC}(\mathbf{A}_{\mathsf{att}}, C) = \mathbf{H}_C \quad \textit{and} \quad \mathsf{MEvalCX}(\mathbf{A}_{\mathsf{att}}, C, \mathbf{x}) = \mathbf{H}_{C,\mathbf{x}}$$

that take as input $\mathbf{A}_{\mathsf{att}}$, a matrix-valued circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \mathbb{Z}_q^{n \times \ell_{\mathsf{out}}}$ (and some $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{in}}}$ for $\mathsf{MEvalCX}$), and output some matrix in $\mathbb{Z}^{(\ell_{\mathsf{in}}+1)m \times \ell_{\mathsf{out}}}$.

- *Suppose C is of depth d, then $\|\mathbf{H}_C^\top\|, \|\mathbf{H}^\top\| \le (m+2)^d \lceil \log q \rceil$.*
- *The matrix encoding homomorphism is $(\mathbf{A}_{\mathsf{att}} - (1, \mathbf{x}^\top) \otimes \mathbf{G})\mathbf{H}_{C,\mathbf{x}} = \mathbf{A}_{\mathsf{att}}\mathbf{H}_C - C(\mathbf{x})$.*

Dual-Use Technique Extension. In [BTVW17], the attribute encoded with secret $\mathbf{s}^\top$ is FHE ciphertext under key $\mathbf{s}^\top$ (hence the name "dual-use"), and the circuit passed to $\mathsf{MEvalCX}$ is some VEval_C. This in turn leads to automatic decryption. Concretely, let C be a vector-valued circuit with codomain $\mathbb{Z}_q^{1 \times \ell_{\mathsf{out}}}$, then VEval_C is $\mathbb{Z}_q^{n \times \ell_{\mathsf{out}}}$-valued and we have that

$$\begin{aligned}
&(\mathbf{s}^\top (\mathbf{A}_{\mathsf{att}} - (1, \mathsf{bits}(\mathbf{X})) \otimes \mathbf{G}) + \mathbf{e}_{\mathsf{att}}^\top) \cdot \mathbf{H}_{\mathsf{VEval}_{C,\mathbf{x}}} \\
&= \mathbf{s}^\top \mathbf{A}_{\mathsf{att}} \mathsf{VEval}_C - \mathbf{s}^\top \mathsf{VEval}_C(\mathbf{X}) + (\mathbf{e}')^\top \quad (\mathsf{MEvalCX}) \\
&= \mathbf{s}^\top \mathbf{A}_{\mathsf{att}} \mathsf{VEval}_C - C(\mathbf{x}) + (\mathbf{e}'')^\top \quad (\mathsf{VEval}\text{ decryption}).
\end{aligned}$$

3.6 Pseudorandom Functions

Definition 3.11 (Pseudorandom Function)**.** A pseudorandom function (PRF) is a family of functions

$$\left\{\mathsf{PRF}(\mathsf{sd}, \cdot)\colon \{0,1\}^{\ell_{\mathsf{in}}(\lambda)} \to \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}\right\}_{\lambda \in \mathbb{N}, \mathsf{sd} \in \{0,1\}^\lambda}$$

with the following properties:

- *efficiency*: one can compute $\mathsf{PRF}(\mathsf{sd}, x)$ in $\mathsf{poly}(\lambda)$-time given x and sd,
- *security*: for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that

$$\left|\Pr\left[\mathcal{A}^{\mathsf{PRF}(\mathsf{sd},\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{R(\cdot)}(1^\lambda) = 1\right]\right| \le \mathsf{negl}(\lambda),$$

  where $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$ and $R \xleftarrow{\$} \mathcal{F}\left(\{0,1\}^{\ell_{\mathsf{in}}(\lambda)} \to \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}\right)$, with $\mathcal{F}\left(\{0,1\}^{\ell_{\mathsf{in}}(\lambda)} \to \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}\right)$ denoting the set of all functions mapping $\ell_{\mathsf{in}}(\lambda)$ bits to $\ell_{\mathsf{out}}(\lambda)$ bits.

3.7 Blind Garbled Circuit

We provide the definition of a blind garbling scheme [BLSV18, AKY24b].

Definition 3.12 (Garbling Scheme)**.** A garbling scheme for circuit class $\mathcal{C} = \{C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}\}$ consists of the following algorithms:

$\mathsf{Garble}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C) \to (\mathsf{lab}, \widetilde{C})$**:** On input a (unary encoded) security parameter λ, an input length ℓ_{in} and an output length ℓ_{out} for circuit C, and a description of the circuit C, it outputs the labels for input wires of garbled circuit $\mathsf{lab} = \{\mathsf{lab}_{j,b}\}_{j \in [\ell_{\mathsf{in}}], b \in \{0,1\}}$, where each $\mathsf{lab}_{j,b} \in \{0,1\}^\lambda$, and the garbled circuit $\widetilde{C}$.

$\mathsf{Eval}(\widetilde{C}, \mathsf{lab}_{\mathbf{x}}) \to \mathbf{y}$**:** On input a garbled circuit $\widetilde{C}$ and the labels $\mathsf{lab}_{\mathbf{x}} = \{\mathsf{lab}_{i,x_i}\}_{i \in [\ell_{\mathsf{in}}]}$ corresponding to an input $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{in}}}$, where x_i denotes the i-th bit of $\mathbf{x}$, it outputs $\mathbf{y} \in \{0,1\}^{\ell_{\mathsf{out}}}$.

$\mathsf{Sim}(1^\lambda, 1^{|C|}, 1^{\ell_{\mathsf{in}}}, \mathbf{y}) \to (\widetilde{\mathsf{lab}}, \widetilde{C})$**:** PPT algorithm that on input a (unary encoded) security parameter λ, a description length of the circuit C, an input length ℓ_{in} and a string $\mathbf{y} \in \{0,1\}^{\ell_{\mathsf{out}}}$, outputs simulated labels lab and a garbled circuit $\widetilde{C}$.

We require a (blind) garbling scheme to satisfy the following properties.

Definition 3.13 (Correctness)**.** A garbling scheme is correct if for any circuit $C \in \mathcal{C}$ and any input $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{in}}}$, the following holds

$$\Pr\left[\mathbf{y} = C(\mathbf{x}) \colon (\mathsf{lab}, \widetilde{C}) \leftarrow \mathsf{Garble}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C), \mathbf{y} := \mathsf{Eval}(\widetilde{C}, \mathsf{lab}_{\mathbf{x}})\right] = 1.$$

Definition 3.14 (Simulation Security)**.** A garbling scheme satisfies simulation security if for any circuit $C \in \mathcal{C}$ and any input $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{in}}}$, the following holds

$$\left\{(\widetilde{C}, \mathsf{lab}_{\mathbf{x}}) : (\mathsf{lab}, \widetilde{C}) \leftarrow \mathsf{Garble}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C)\right\} \approx_c \left\{(\widetilde{C}, \mathsf{lab}_{\mathbf{x}}) : (\widetilde{C}, \mathsf{lab}_{\mathbf{x}}) \leftarrow \mathsf{Sim}(1^\lambda, 1^{|C|}, 1^{\ell_{\mathsf{in}}}, C(\mathbf{x}))\right\},$$

where $\mathsf{lab} = \{\mathsf{lab}_{j,b}\}_{j \in [\ell_{\mathsf{in}}], b \in \{0,1\}}$ and $\mathsf{lab}_{\mathbf{x}} = \{\mathsf{lab}_{i,x_i}\}_{i \in [\ell_{\mathsf{in}}]}$.

Definition 3.15 (Blindness)**.** A garbling scheme is called blind if the distribution $\mathsf{Sim}(1^\lambda, 1^{|C|}, 1^{\ell_{\mathsf{in}}}, \mathbf{y})$ for $\mathbf{y} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{out}}}$, representing the output of the simulator on a uniformly random output, is indistinguishable from a completely uniform bit string. (Note that the distinguisher must not know the random output value that was used for the simulation.)

Definition 3.16 (Decomposability)**.** The algorithm $\mathsf{Garble}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C)$ can be decomposed, using shared randomness st, as follows: (i) $\mathsf{Garble}_k(1^\lambda, C_k; \mathsf{st})$ for $k \in [|C|]$, where $\mathsf{Garble}_k(1^\lambda, C_k)$ outputs the garbling of k-th gate of the circuit C (denoted by C_k), and (ii) $\mathsf{Garble}_{\mathsf{inp},k}(1^\lambda; \mathsf{st}) = (\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1})$, for $k \in [\ell_{\mathsf{in}}]$, which outputs the labels corresponding to the k-th input bit.

Fact 3.17 ([BLSV18])**.** *Assume that one-way function exists. Then, there exists a blind garbled circuit scheme.*

3.8 (Unbounded) Blind Batch Encryption

We provide here a definition of a blind batch encryption adapted from [BLSV18, AKY24b].

Definition 3.18 ((Unbounded) Blind Batch Encryption)**.** A blind batch encryption scheme with message space $\mathcal{M} = \{\mathcal{M}_\lambda = \{0,1\}^\lambda\}_\lambda$ consists of the following algorithms:

$\mathsf{Setup}(1^\lambda, 1^N) \to \mathsf{crs}$**:** On input the (unary encoded) security parameter λ and key length N, it outputs a common reference string crs.

$\mathsf{Gen}(\mathsf{crs}, x) \to h$**:** On input the common reference string crs and a secret key $x \in \{0,1\}^{\lambda N}$, it outputs a public key h.

$\mathsf{SingleEnc}(\mathsf{crs}, h, i, (m_0, m_1)) \to \mathsf{ct}$**:** On input the common reference string crs, a public key h, an index $i \in [N]$ and a message pair $(m_0, m_1) \in \mathcal{M}_\lambda^2$, it outputs a (single) ciphertext ct.

$\mathsf{SingleDec}(\mathsf{crs}, x, i, \mathsf{ct}) \to m$**:** On input the common reference string crs, a secret key $x \in \{0,1\}^{\lambda N}$, an index $i \in [N]$ and a (single) ciphertext ct, it outputs a message $m \in \mathcal{M}_\lambda$ or $\perp$.

We say a blind batch encryption scheme is *unbounded*, if the length of the secret keys are not bounded by N, i.e., $x \in \{0,1\}^*$, and the Setup algorithm does not take the additional input 1^N.

Additionally, we consider batch encryption and decryption algorithms that not only operate on a single index $i \in [N]$, but instead on *all* $i \in [N]$ at the same time.

$\mathsf{Enc}(\mathsf{crs}, h, \mathbf{M})$**:** On input the common reference string crs, a public key h, and a message $\mathbf{M} \in \mathcal{M}_\lambda^{N \times 2}$, it puts a ciphertext ct.

$\mathsf{Dec}(\mathsf{crs}, x, \mathsf{ct})$**:** On input the common reference string crs, a secret key $x \in \{0,1\}^{\lambda N}$ and a ciphertext ct, it outputs a message vector $\mathbf{m} \in \mathcal{M}_\lambda^N$ or $\perp$.

We simply require that $\mathsf{Enc}(\mathsf{crs}, h, \mathbf{M}) = (\mathsf{ct}_i)_{i\in[N]}$ for $\mathsf{ct}_i \leftarrow \mathsf{SingleEnc}(\mathsf{crs}, h, i, \mathbf{m}_i)$, where $\mathbf{m}_i$ denotes the i-th row of $\mathbf{M}$. Similarly, we require that for $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[n]}$, $\mathsf{Dec}(\mathsf{crs}, x, \mathsf{ct})$ computes $m_i \leftarrow \mathsf{SingleDec}(\mathsf{crs}, x, i, \mathsf{ct}_i)$ for all $i \in [n]$, and outputs their concatenation.

Next, we recall the correctness and succinctness definitions.

Definition 3.19 (Correctness)**.** A blind batch encryption scheme is said to be correct if for any $\lambda, N \in \mathbb{N}$, $x \in \{0,1\}^{\lambda N}$, $i \in [N]$, $(m_0, m_1) \in \mathcal{M}_\lambda^2$, $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ and $h := \mathsf{Gen}(\mathsf{crs}, x)$, it holds that

$$\Pr\left[m = m_{x[(i-1)\lambda+1:i\lambda]} \mid m = \mathsf{SingleDec}(\mathsf{crs}, x, i, \mathsf{SingleEnc}(\mathsf{crs}, h, i, (m_0, m_1)))\right] = 1.$$

Definition 3.20 (Succinctness)**.** A blind batch encryption scheme is α-succinct if letting $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, $h := \mathsf{Gen}(\mathsf{crs}, x)$, for some $x \in \{0,1\}^{\lambda N}$, it holds that $|h| \le \alpha N$. It is fully succinct if $|h| \le p(\lambda)$ for some fixed polynomial $p(\lambda)$.

We require a blind batch encryption to satisfy the following adaptive security, which is stronger than the one defined in [BLSV18].

Definition 3.21 (Adaptive Security)**.** A blind batch encryption scheme is said to be adaptively secure if for any PPT adversary $\mathcal{A}$ and any $N \in \mathbb{N}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that the following holds

$$\Pr\left[b' = b \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N) \\ \left(x \in \{0,1\}^{\lambda N}, i \in [N], \mathbf{m}^{(0)} = (m_0^{(0)}, m_1^{(0)}), \mathbf{m}^{(1)} = (m_0^{(1)}, m_1^{(1)})\right) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b \xleftarrow{\$} \{0,1\}, h := \mathsf{Gen}(\mathsf{crs}, x) \\ \mathsf{ct}_b \leftarrow \mathsf{SingleEnc}(\mathsf{crs}, h, i, \mathbf{m}^{(b)}) \\ b' \leftarrow \mathcal{A}(\mathsf{ct}_b) \end{array}\right] \le \frac{1}{2} + \mathsf{negl}(\lambda),$$

where we require that $\mathbf{m}^{(0)}, \mathbf{m}^{(1)} \in \mathcal{M}_\lambda^2$ and $m^{(0)}_{x[i]} = m^{(1)}_{x[i]}$.

We consider in this work the following adaptive blindness definition, which is stronger than the blindness definition given in [AKY24b].

Definition 3.22 (Adaptive Blindness)**.** A blind batch encryption scheme is said to be adaptively blind if for any PPT adversary $\mathcal{A}$ and any $N \in \mathbb{N}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that the following holds

$$\Pr\left[b' = b \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N) \\ (x \in \{0,1\}^{\lambda N}, i \in [N]) \leftarrow \mathcal{A}(\mathsf{crs}) \\ h := \mathsf{Gen}(\mathsf{crs}, x) \\ \mathbf{m} \xleftarrow{\$} \mathcal{M}^2, b \xleftarrow{\$} \{0,1\} \\ \mathsf{ct}_0 \leftarrow \mathsf{SingleEnc}(\mathsf{crs}, h, i, \mathbf{m}), \mathsf{ct}_1 \xleftarrow{\$} \mathcal{CT} \\ b' \leftarrow \mathcal{A}(\mathsf{ct}_b) \end{array}\right] \le \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathcal{CT}$ is the ciphertext space of the scheme.

Remark 3.23 (Unbounded BBE Security and Blindness). We note that if the underlying BBE scheme is unbounded, then the Setup algorithm does not take 1^N as input and $\mathcal{A}$ can output a secret key x of arbitrary length in Definitions 3.21 and 3.22.

3.9 Symmetric Key Encryption

We recall the definition of a symmetric key encryption scheme.

Definition 3.24 (Syntax of SKE)**.** A Symmetric Key Encryption (SKE) scheme for message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of three efficient algorithms:

$\mathsf{Setup}(1^\lambda) \to \mathsf{sk}$**.** On input the security parameter 1^λ, this algorithm outputs a secret key sk.

$\mathsf{Enc}(\mathsf{sk}, \mu) \to \mathsf{ct}$**.** On input the secret key sk and a message $\mu \in \mathcal{M}_\lambda$, this algorithm outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mu$**.** On input the secret key sk and a ciphertext ct, this algorithm outputs a message $\mu \in \mathcal{M}_\lambda$.

For notational convenience, we may pass a second argument $1^{\ell(\lambda)}$ to Setup indicating that the message space of the scheme is $\mathcal{M} = \{\mathcal{M}_\lambda = \{0,1\}^{\ell(\lambda)}\}_{\lambda\in\mathbb{N}}$.

Correctness. A SKE scheme is correct if for all $\lambda, \ell \in \mathbb{N}$ and $\boldsymbol{\mu} \in \mathcal{M}_\lambda$, we have that

$$\Pr\left[\mu = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \;\middle|\; \begin{array}{l} \mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu) \end{array}\right] = 1 \ ,$$

where the probability is taken over the random coins of Setup and Enc.

Security. We define security with pseudorandom ciphertexts.

Definition 3.25 (INDr Security for SKE)**.** An SKE scheme SKE with ciphertext space $\mathcal{CT} = \{\mathcal{CT}\}_{\lambda\in\mathbb{N}}$ satisfies INDr security if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all PPT adversaries $\mathcal{A}$, we have that

$$\Pr\left[b' = b \;\middle|\; \begin{array}{l} b \xleftarrow{\$} \{0,1\};\ \mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda) \\ b' \leftarrow \mathcal{A}^{\mathsf{QEnc}(\mathsf{sk},\cdot),\mathsf{QEnc}_b(\mathsf{sk},\cdot)}(1^\lambda) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where the oracles are defined as follows:

- $\mathsf{QEnc}(\mathsf{sk}, \cdot)$. On input $\mu \in \mathcal{M}_\lambda$, return $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu)$.
- $\mathsf{QEnc}_b(\mathsf{sk}, \cdot)$. On input $\mu \in \mathcal{M}_\lambda$, return ct_b, where $\mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu)$ and $\mathsf{ct}_1 \xleftarrow{\$} \mathcal{CT}_\lambda$.

3.10 Functional Encryption

We recall the definition of functional encryption with pseudorandom ciphertext security. Let $\mathcal{X} = \{\mathcal{X}_\lambda = \mathcal{X}_{\lambda,\mathsf{pub}} \times \mathcal{X}_{\lambda,\mathsf{pri}}\}_{\lambda\in\mathbb{N}}$, $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be sequences of input, output and function spaces, respectively, where $F_\lambda \colon \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$ for all $\lambda \in \mathbb{N}$. Here, we assume that each input space $\mathcal{X}_\lambda$ consists of a public component $\mathcal{X}_{\lambda,\mathsf{pub}}$ and a private component $\mathcal{X}_{\lambda,\mathsf{pri}}$.

Definition 3.26 (Syntax of FE)**.** A FE scheme FE for $\mathcal{F}$ consists of four efficient algorithms:

$\mathsf{Setup}(1^\lambda, \mathsf{param}) \to (\mathsf{mpk}, \mathsf{msk})$**.** On input the security parameter 1^λ and parameters param specifying $\mathcal{F}$, this algorithm outputs a master public key mpk and a master secret key msk.

$\mathsf{KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$**.** On input the master secret key msk and a function $f \in \mathcal{F}_\lambda$, this algorithm outputs a secret key sk_f.

$\mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}) \to \mathsf{ct}$**.** The encryption algorithm proceeds in two steps.

$\mathsf{EncOff}(\mathsf{mpk}) \to (\mathsf{ct}_{\mathsf{off}}, \mathsf{st})$**.** On input the master public key mpk, this algorithm outputs an offline ciphertext $\mathsf{ct}_{\mathsf{off}}$ and a state st.

$\mathsf{EncOn}(\mathsf{st}, x_{\mathsf{pub}}, x_{\mathsf{pri}}) \to \mathsf{ct}_{\mathsf{on}}$**.** On input the state st and an input $(x_{\mathsf{pub}}, x_{\mathsf{pri}}) \in \mathcal{X}_{\lambda,\mathsf{pub}} \times \mathcal{X}_{\lambda,\mathsf{pri}}$, this algorithm outputs an online ciphertext $\mathsf{ct}_{\mathsf{on}}$.

The final output of $\mathsf{Enc}(\mathsf{mpk}, x)$ is $\mathsf{ct} := (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}})$.

$\mathsf{Dec}(\mathsf{sk}_f, f, \mathsf{ct}, x_{\mathsf{pub}}) \to y \vee \bot$**.** On input a secret key sk_f, the corresponding function $f \in \mathcal{F}_\lambda$, a ciphertext ct and the corresponding public input $x_{\mathsf{pub}} \in \mathcal{X}_{\lambda,\mathsf{pub}}$, this algorithm outputs an element $y \in \mathcal{Y}_\lambda$ or $\bot$ indicating failure of decryption.

We say that FE has a *trivial offline encryption* if $\mathsf{EncOff}(\mathsf{mpk})$ outputs $(\mathsf{ct}_{\mathsf{off}} := \bot, \mathsf{st} := \mathsf{mpk})$. In this case, we may simplify the syntax by completely ignoring EncOff and letting $\mathsf{Enc} := \mathsf{EncOn}$ and $\mathsf{ct} := \mathsf{ct}_{\mathsf{on}}$.

Correctness. A FE scheme is correct if for all $\lambda \in \mathbb{N}$, $(x_{\mathsf{pub}}, x_{\mathsf{pri}}) \in \mathcal{X}_{\lambda,\mathsf{pub}} \times \mathcal{X}_{\lambda,\mathsf{pri}}$ and $f \in \mathcal{F}_\lambda$, we have

$$\Pr\left[\begin{array}{l|l} y' = f(x_{\mathsf{pub}}, x_{\mathsf{pri}}) & \begin{array}{l}(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param}) \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \\ (\mathsf{ct}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{EncOff}(\mathsf{mpk}) \\ \mathsf{ct}_{\mathsf{on}} \leftarrow \mathsf{EncOn}(\mathsf{st}, x_{\mathsf{pub}}, x_{\mathsf{pri}}) \\ y' := \mathsf{Dec}(\mathsf{sk}_f, f, \mathsf{ct} = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}}), x_{\mathsf{pub}})\end{array}\end{array}\right] \geq 1 - \mathsf{negl}(\lambda) \ ,$$

where the probability is taken over the random coins of Setup, KeyGen and Enc.

Security. We define reusable pseudorandom ciphertext security as in [AKY24b].

Definition 3.27 (Reusable VerSel-prCT Security for FE)**.** For a FE scheme FE supporting a function class $\mathcal{F} = \{\mathcal{F}_\lambda : \mathcal{X}_\lambda \to \mathcal{Y}_\lambda\}$, we let Samp be a PPT algorithm that on input 1^λ outputs

$$\left(\mathsf{aux} \in \{0,1\}^*, \{f^i\}_{i\in[Q_{\mathsf{key}}]} \subseteq \mathcal{F}_\lambda, \{x^j = (x^j_{\mathsf{pub}}, x^j_{\mathsf{pri}})\}_{j\in[Q_{\mathsf{msg}}]} \subseteq \mathcal{X}_\lambda\right) .$$

We define the following advantage functions:

$$\begin{aligned}\mathbf{Adv}^{\mathsf{pre}}_{\mathsf{Samp},\mathcal{A}_{\mathsf{pre}}}(\lambda) = \Big|&\Pr\Big[\mathcal{A}_{\mathsf{pre}}\big(\mathsf{aux}, \{f^i\}_{i\in[Q_{\mathsf{key}}]}, \{x^j_{\mathsf{pub}}\}_{j\in[Q_{\mathsf{msg}}]}, \{\delta^*_{i,j} := f^i(x^j)\}_{(i,j)\in[Q_{\mathsf{key}}]\times[Q_{\mathsf{msg}}]}\big) \to 1\Big] \\ &- \Pr\Big[\mathcal{A}_{\mathsf{pre}}\big(\mathsf{aux}, \{f^i\}_{i\in[Q_{\mathsf{key}}]}, \{x^j_{\mathsf{pub}}\}_{j\in[Q_{\mathsf{msg}}]}, \{\delta^{\$}_{i,j} \xleftarrow{\$} \mathcal{Y}_\lambda\}_{(i,j)\in[Q_{\mathsf{key}}]\times[Q_{\mathsf{msg}}]}\big) \to 1\Big]\Big| \ , \\ \mathbf{Adv}^{\mathsf{post}}_{\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}(\lambda) = \Big|&\Pr\Big[\mathcal{A}_{\mathsf{post}}\big(\mathsf{aux}, \mathsf{mpk}, \mathsf{ct}_{\mathsf{off}}, \{f^i, \mathsf{sk}_{f^i}\}_{i\in[Q_{\mathsf{key}}]}, \{x^j_{\mathsf{pub}}, \Delta^*_j := \mathsf{ct}^j_{\mathsf{on}}\}_{j\in[Q_{\mathsf{msg}}]}\big) \to 1\Big] \\ &- \Pr\Big[\mathcal{A}_{\mathsf{pre}}\big(\mathsf{aux}, \mathsf{mpk}, \mathsf{ct}_{\mathsf{off}}, \{f^i, \mathsf{sk}_{f^i}\}_{i\in[Q_{\mathsf{key}}]}, \{x^j_{\mathsf{pub}}, \Delta^{\$}_j \xleftarrow{\$} \mathcal{CT}_{\mathsf{on}}\}_{j\in[Q_{\mathsf{msg}}]}\big) \to 1\Big]\Big| \ ,\end{aligned}$$

where $\mathcal{CT}_{\mathsf{on}}$ denotes the online part of the ciphertext space and

$$\begin{aligned}&\left(\mathsf{aux} \in \{0,1\}^*, \{f^i\}_{i\in[Q_{\mathsf{key}}]} \subseteq \mathcal{F}_\lambda, \{x^j = (x^j_{\mathsf{pub}}, x^j_{\mathsf{pri}})\}_{j\in[Q_{\mathsf{msg}}]} \subseteq \mathcal{X}_\lambda\right) \leftarrow \mathsf{Samp}(1^\lambda) \ , \\ &(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param}) \ , \quad \{\mathsf{sk}_{f^i} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f^i)\}_{i\in[Q_{\mathsf{key}}]} \ , \\ &(\mathsf{ct}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{EncOff}(\mathsf{mpk}) \ , \quad \{\mathsf{ct}^j_{\mathsf{on}} \leftarrow \mathsf{EncOn}(\mathsf{st}, x^j)\}_{j\in[Q_{\mathsf{msg}}]} \ .\end{aligned}$$

We say that FE satisfies reusable VerSel-prCT security if for all PPT samplers Samp and PPT adversaries $\mathcal{A}_{\mathsf{post}}$, there exists another PPT adversary $\mathcal{A}_{\mathsf{pre}}$ such that

$$\mathbf{Adv}^{\mathsf{pre}}_{\mathsf{Samp},\mathcal{A}_{\mathsf{pre}}}(\lambda) \geq \mathbf{Adv}^{\mathsf{post}}_{\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}(\lambda)/\mathsf{poly}(\lambda) - \mathsf{negl}(\lambda)$$

and $\mathsf{Time}(\mathcal{A}_{\mathsf{pre}}) \leq \mathcal{A}_{\mathsf{post}} \cdot \mathsf{poly}(\lambda)$.

We note that Definition 3.27 is a hybrid between single-challenge and multi-challenge security in the sense that only a single offline part but multiple online parts of the challenge ciphertext are provided to the adversary. Nevertheless, this hybrid notion is essentially equivalent to full-fledged multi-challenge security if the FE scheme has trivial offline encryption.

Fact 3.28 (Adapted from [AKY24b, Theorem 3.5 and Theorem 5.12])**.** *Assuming LWE and (private-coin) evasive LWE, there exist a FE scheme for bounded depth circuits satisfying trivial offline encryption and reusable* VerSel-prCT *security with the following parameters:*

$$|\mathsf{mpk}| = \ell_{\mathsf{pri}} \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda) \,, \qquad |\mathsf{sk}|_C = \ell_{\mathsf{out}} \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda) \,, \qquad |\mathsf{ct}| = \ell_{\mathsf{pri}} \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda) \,,$$

where ℓ_{pri}, ℓ_{out} and ℓ_{dep} denote the private input length, the output length and the maximum depth, respectively. (We refer to this primitive as "prFE" in the rest of the paper).

Furthermore, assuming LWE and prFE, there exists a FE scheme for unbounded depth circuits satisfying reusable VerSel-prCT *security with the following parameters:*

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda) \,, \qquad |\mathsf{sk}|_C = \mathsf{poly}(\lambda) \,, \qquad |\mathsf{mpk}| = \mathsf{poly}(\lambda) + \ell_{\mathsf{pri}} \,.$$

3.11 Attribute-Based and Predicate Encryption

The definitions for ABE and Predicate Encryption (PE) are nearly identical. We therefore give the formal definitions only for the case of ABE and mention differences compared to PE along the way. Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$, $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be sequences of message, ciphertext attribute and key attribute spaces, respectively. Furthermore, let $R = \{R_\lambda\}_{\lambda\in\mathbb{N}}$ be a sequence of relations, where $R_\lambda : \mathcal{X}_\lambda \times \mathcal{Y}_\lambda \to \{0,1\}$ for all $\lambda \in \mathbb{N}$.

Definition 3.29 (Syntax of ABE)**.** An Attribute-Based Encryption (ABE) scheme for $\mathcal{M}$ and R consists of four efficient algorithms:

$\mathsf{Setup}(1^\lambda, \mathsf{param}) \to (\mathsf{mpk}, \mathsf{msk})$**.** On input the security parameter 1^λ and parameters param specifying R, this algorithm outputs a master public key mpk and a master secret key msk. We assume that msk implicitly contains mpk.

$\mathsf{KeyGen}(\mathsf{msk}, y) \to \mathsf{sk}_y$**.** On input the master secret key msk and a key attribute $y \in \mathcal{Y}_\lambda$, this algorithm outputs a secret key sk_y.

$\mathsf{Enc}(\mathsf{mpk}, x, \mu) \to \mathsf{ct}_x$**.** The encryption algorithm proceeds in two steps.

- $\mathsf{EncOff}(\mathsf{mpk}) \to (\mathsf{st}, \mathsf{ct}_{\mathsf{off}})$**.** On input the master public key mpk, this algorithm outputs a state st and an offline ciphertext $\mathsf{ct}_{\mathsf{off}}$.
- $\mathsf{EncOn}(\mathsf{st}, x, \mu) \to \mathsf{ct}_{\mathsf{on}}$**.** On input the state st, a ciphertext attribute $x \in \mathcal{X}_\lambda$ and a message $\mu \in \mathcal{M}_\lambda$, this algorithm outputs an online ciphertext $\mathsf{ct}_{\mathsf{on}}$.

The final output of $\mathsf{Enc}(\mathsf{mpk}, x, \mu)$ is $\mathsf{ct} := (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}})$.

$\mathsf{Dec}(\mathsf{sk}_y, y, \mathsf{ct}, x) \to \mu'$ **or** $\perp$**.** On input a secret key sk_y, key attribute $y \in \mathcal{Y}_\lambda$, a ciphertext ct and ciphertext attribute $x \in \mathcal{X}_\lambda$, this algorithm outputs a message $\mu' \in \mathcal{M}_\lambda$ or $\perp$ indicating failure of decryption. In the case of PE, the algorithm does not take x as input.

Correctness. An ABE scheme is correct if for all $\lambda \in \mathbb{N}$, $\mu \in \mathcal{M}_\lambda$, $x \in \mathcal{X}_\lambda$ and $y \in \mathcal{Y}_\lambda$ such that $R_\lambda(x, y) = 0$, we have that

$$\Pr\left[\mu = \mathsf{Dec}(\mathsf{sk}_y, y, \mathsf{ct}_x, x) \;\middle|\; \begin{array}{l} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param}) \\ \mathsf{sk}_y \leftarrow \mathsf{KeyGen}(\mathsf{msk}, y) \\ \mathsf{ct}_x \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, \mu) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda) \,,$$

where the probability is taken over the random coins of Setup, KeyGen and Enc.

Security. We start by defining traditional VerSel-IND security.

Definition 3.30 (VerSel-IND Security for ABE)**.** An ABE scheme ABE for a relation $R = \{R_\lambda : \mathcal{X}_\lambda \times \mathcal{Y}_\lambda \to \{0,1\}\}$ satisfies VerSel-IND security if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all PPT adversaries, we have

$$\Pr\left[b' = b \;\middle|\; \begin{array}{l} b \xleftarrow{\$} \{0,1\};\ (\{y^i\}_{i\in[Q_{\mathsf{key}}]}, x^*, \mu_0^*, \mu_1^*) \leftarrow \mathcal{A}(\lambda) \\ (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \{\mathsf{sk}_{y^i} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, y^i)\}_{i\in[Q_{\mathsf{key}}]} \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x^*, \mu_b^*) \\ b' \leftarrow \mathcal{A}(\mathsf{coins}_{\mathcal{A}}, \mathsf{mpk}, \{\mathsf{sk}_{y^i}\}_{i\in[Q_{\mathsf{key}}]}, \mathsf{ct}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)\,,$$

where we require that $R(x^*, y^i) = 1$ for all $i \in [Q_{\mathsf{key}}]$.

In the case of PE, the adversary $\mathcal{A}$ outputs two challenge attributes (x_0^*, x_1^*) and the challenger computes $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_b^*, \mu_b^*)$. Furthermore, we consider the notion of *reusable* VerSel-INDr security [AKY24b].

Definition 3.31 (Reusable VerSel-INDr Security for ABE)**.** An ABE scheme ABE with two-stage encryption for a relation $R = \{R_\lambda : \mathcal{X}_\lambda \times \mathcal{Y}_\lambda \to \{0,1\}\}$ satisfies reusable VerSel-INDr security if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all PPT adversaries, we have

$$\Pr\left[b' = b \;\middle|\; \begin{array}{l} b \xleftarrow{\$} \{0,1\};\ (\mathsf{coins}_{\mathcal{A}}, \{y^i\}_{i\in[Q_{\mathsf{key}}]}, \{x^j\}_{j\in[Q_{\mathsf{att}}]}, \mu) \leftarrow \mathcal{A}(\lambda) \\ (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \{\mathsf{sk}_{y^i} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, y^i)\}_{i\in[Q_{\mathsf{key}}]} \\ (\mathsf{ct}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{EncOff}(\mathsf{mpk}) \\ \{\mathsf{ct}^j_{\mathsf{on},0} \leftarrow \mathsf{EncOn}(\mathsf{st}, x^j, \mu);\ \mathsf{ct}^j_{\mathsf{on},1} \xleftarrow{\$} \mathcal{CT}_{\mathsf{on}}\}_{j\in[Q_{\mathsf{att}}]} \\ b' \leftarrow \mathcal{A}(\mathsf{coins}_{\mathcal{A}}, \mathsf{mpk}, \{\mathsf{sk}_{y^i}\}_{i\in[Q_{\mathsf{key}}]}, \mathsf{ct}_{\mathsf{off}}, \{\mathsf{ct}^j_{\mathsf{on},b}\}_{j\in[Q_{\mathsf{att}}]}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)\,,$$

where $\mathcal{CT}_{\lambda,\mathsf{on}}$ denotes the ciphertext space of EncOn. We require $R(x^j, y^i) = 1$ for $i \in [Q_{\mathsf{key}}]$ and $j \in [Q_{\mathsf{att}}]$.

We note that reusable VerSel-INDr security implies VerSel-IND security.

Policy Classes. In this work, we consider ABE with respect to the following relations. Recall that $\mathcal{C}_\ell$ (resp. $\mathcal{T}$) denotes the class of all unbounded depth circuits $C\colon \{0,1\}^\ell \to \{0,1\}$ (resp. Turing machines).

- *Key-Policy ABE for Circuits.* KP-ABE for circuits is described by $\mathcal{X}_\lambda = \{0,1\}^{\ell(\lambda)}$, $\mathcal{Y}_\lambda = \mathcal{C}_{\ell(\lambda)}$ and the relation $R_\lambda : \mathcal{X}_\lambda \times \mathcal{Y}_\lambda \to \{0,1\}$, where $R_\lambda(\mathbf{x}, C) = C(\mathbf{x})$.
- *Ciphertext-Policy ABE for Circuits.* CP-ABE for circuits is described by $\mathcal{X}_\lambda = \mathcal{C}_{\ell(\lambda)}$, $\mathcal{Y}_\lambda = \{0,1\}^{\ell(\lambda)}$ and the relation $R_\lambda : \mathcal{X}_\lambda \times \mathcal{Y}_\lambda \to \{0,1\}$, where $R_\lambda(C, \mathbf{x}) = C(\mathbf{x})$.
- *Key-Policy ABE for Turing Machines.* KP-ABE for Turing machines is described by $\mathcal{X}_\lambda = \{0,1\}^*$, $\mathcal{Y}_\lambda = \mathcal{T}$ and the relation $R_\lambda : \mathcal{X}_\lambda \times \mathcal{Y}_\lambda \to \{0,1\}$, where

$$R_\lambda((1^t, \mathbf{x}), M) = \begin{cases} 0 & \text{if } M \text{ accepts } \mathbf{x} \text{ in } t \text{ steps}\,, \\ 1 & \text{otherwise}\,. \end{cases}$$

- *Ciphertext-Policy ABE for Circuits.* CP-ABE for Turing machines is described by $\mathcal{X}_\lambda = \mathcal{T}$, $\mathcal{Y}_\lambda = \{0,1\}^*$ and the relation $R_\lambda : \mathcal{X}_\lambda \times \mathcal{Y}_\lambda \to \{0,1\}$, where

$$R_\lambda(M, (1^t, \mathbf{x})) = \begin{cases} 0 & \text{if } M \text{ accepts } \mathbf{x} \text{ in } t \text{ steps}\,, \\ 1 & \text{otherwise}\,. \end{cases}$$

Fact 3.32 ([AKY24b])**.** *Assuming LWE, pPRIO and FE with reusable* VerSel-prCT *security for bounded depth circuits, there exist KP-ABE and CP-ABE schemes supporting unbounded depth circuits with master public keys, secret keys and ciphertexts all of size* $\mathsf{poly}(\lambda)$.

3.12 Registered Attribute-Based and Predicate Encryption

The definitions for RABE and RPE are nearly identical. We therefore give the formal definitions only for the case of RABE and mention differences compared to RPE along the way.

Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$, $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be sequences of message, ciphertext attribute and key attribute spaces, respectively. Furthermore, let $R = \{R_\lambda\}_{\lambda\in\mathbb{N}}$ be a sequence of relations, where $R_\lambda : \mathcal{X}_\lambda \times \mathcal{Y}_\lambda \to \{0,1\}$ for all $\lambda \in \mathbb{N}$.

Definition 3.33 (Syntax of RABE)**.** A RABE scheme for message space $\mathcal{M}$ and relation R consists consists of six efficient algorithms:

$\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$**:** On input the security parameter 1^λ and the maximum number of users 1^L, this algorithm outputs a common reference string crs.

$\mathsf{Gen}(\mathsf{crs}, \mathsf{aux}) \to (\mathsf{pk}_i, \mathsf{sk}_i)$**:** On input the crs and a state aux, this algorithm outputs a pair of a public and a secret key $(\mathsf{pk}_i, \mathsf{sk}_i)$.

$\mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, y) \to (\mathsf{mpk}, \mathsf{aux}')$**:** On input the crs, a state aux, a public key pk and an attribute $y \in \mathcal{Y}_\lambda$, this algorithm outputs a master public key mpk and an updated state aux′. We require Reg to be deterministic.

$\mathsf{Enc}(\mathsf{mpk}, x, \mu) \to \mathsf{ct}$**:** On input the master public key mpk, a public input $x \in \mathcal{X}_\lambda$ and a message $\mu \in \mathcal{M}_\lambda$, this algorithm outputs a ciphertext ct.

$\mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}) \to \mathsf{hsk}$**:** On input the crs, a state aux and a public key pk, this algorithm outputs a helper secret key hsk. We require Update to be deterministic.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{hsk}, y, \mathsf{ct}, x) \to \mu' \vee \perp \vee \mathsf{GetUpdate}$**:** On input a secret key sk, a helper secret key hsk, a registered attribute $y \in \mathcal{Y}_\lambda$ and a ciphertext ct with corresponding attribute x, this algorithm either outputs a message $\mu \in \mathcal{M}_\lambda$, or a special symbol $\perp$ indicating decryption failure, or a special message GetUpdate indicating an updated helper secret key is needed to decrypt the ciphertext. We require Dec to be deterministic. In the case of RPE, the algorithm does not take x as input.

We recall the definitions of correctness, compactness and update efficiency.

Definition 3.34 (Correctness, Compactness and Update Efficiency of RABE)**.** Given a RABE scheme RABE and a PPT adversary $\mathcal{A}$, we define the experiment $\mathbf{Exp}_{\mathsf{RABE},\mathcal{A}}$ as follows:

- *Setup.* Run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ and send crs to $\mathcal{A}$. Initialize the auxiliary input $\mathsf{aux} := \perp$, two empty dictionaries $\mathcal{E}, \mathcal{R}$ and counters $i_{\mathsf{reg}}, i^*_{\mathsf{reg}}, i_{\mathsf{enc}} := 1$ to keep track of QRegNT, QRegT and QEnc queries. Let $b = 0$ and $y^*, \mathsf{pk}^*, \mathsf{sk}^*, \mathsf{hsk}^* := \perp$.
- *Query.* Repeat the following for arbitrarily many rounds determined by $\mathcal{A}$. The oracle QRegT can be queried exactly once. In each round, $\mathcal{A}$ has four options.
  - $\mathsf{QRegNT}(\mathsf{pk}, y)$: upon $\mathcal{A}$ submitting a public key pk and an attribute $y \in \mathcal{Y}_\lambda$, run $(\mathsf{mpk}, \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, y)$ and return $(i_{\mathsf{reg}}, \mathsf{mpk}, \mathsf{aux}')$ to $\mathcal{A}$. Update $\mathcal{R}[i_{\mathsf{reg}}] := (\mathsf{mpk}, \mathsf{aux}')$, $\mathsf{aux} := \mathsf{aux}'$ and $i_{\mathsf{reg}} := i_{\mathsf{reg}} + 1$.
  - $\mathsf{QRegT}(y)$: upon $\mathcal{A}$ submitting a function $y \in \mathcal{Y}_\lambda$, run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{aux})$, $(\mathsf{mpk}, \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, y)$, $\mathsf{hsk} := \mathsf{Update}(\mathsf{crs}, \mathsf{aux}', \mathsf{pk})$ and return $(i_{\mathsf{reg}}, \mathsf{mpk}, \mathsf{aux}', \mathsf{pk}, \mathsf{sk}, \mathsf{hsk})$ to $\mathcal{A}$. Update $\mathcal{R}[i_{\mathsf{reg}}] := (\mathsf{mpk}, \mathsf{aux}')$, $\mathsf{aux} := \mathsf{aux}'$, $i_{\mathsf{reg}} := i_{\mathsf{reg}} + 1$ and $i^*_{\mathsf{reg}} := i_{\mathsf{reg}}$. Furthermore, set $y^* := y$, $\mathsf{pk} := \mathsf{pk}^*$, $\mathsf{sk} := \mathsf{sk}^*$ and $\mathsf{hsk} := \mathsf{hsk}^*$,
  - $\mathsf{QEnc}(j, x, \mu)$: upon $\mathcal{A}$ submitting an index $j \in [i^*_{\mathsf{reg}}; i_{\mathsf{reg}}]$, an attribute $x \in \mathcal{X}_\lambda$ and a message $\mu \in \mathcal{M}_\lambda$, retrieve $(\mathsf{mpk}, \star) := \mathcal{R}[j]$, run $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, \mu)$ and return $(i_{\mathsf{enc}}, \mathsf{ct})$ to $\mathcal{A}$. Set $\mathcal{E}[i_{\mathsf{enc}}] := (x, \mu, \mathsf{ct})$ and $i_{\mathsf{enc}} := i_{\mathsf{enc}} + 1$.

– $\mathsf{QDec}(j)$: upon $\mathcal{A}$ submitting an index $j \in [i_{\mathsf{enc}}]$, check if $\mathsf{sk}^* = \bot$ and return $\bot$ in this case. Otherwise, retrieve the tuple $(x, \mu, \mathsf{ct}) := \mathcal{E}[j]$ and run $\mu' \leftarrow \mathsf{Dec}(\mathsf{sk}^*, \mathsf{hsk}^*, y^*, \mathsf{ct}, x)$. If $\mu' = \mathsf{GetUpdate}$, run $\mathsf{hsk}^* \leftarrow \mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*)$ and recompute $\mu' \leftarrow \mathsf{Dec}(\mathsf{sk}^*, \mathsf{hsk}^*, y^*, \mathsf{ct}, x)$. Set $b = 1$ if $\mu \neq \mu'$.

We say that RABE is

- correct if $\Pr[\mathbf{Exp}_{\mathsf{RABE},\mathcal{A}}(1^\lambda) \to 0] = 1$ for all PPT adversaries $\mathcal{A}$,
- compact if $|\mathsf{mpk}|, |\mathsf{hsk}| = \mathsf{poly}(\lambda, \log L)$ and $|\mathsf{ct}| = \mathsf{poly}(\lambda, \log L) + |\mu|$ for RABE, $|\mathsf{ct}| = \mathsf{poly}(\lambda, \log L) + |\mu| + |x|$ for RPE at any stage during the execution of the experiment $\mathbf{Exp}_{\mathsf{RABE},\mathcal{A}}(1^\lambda)$, and
- update efficient if the oracle QDec invokes Update at most $O(\log|\mathcal{R}|)$ times and each invocation runs in time $\mathsf{poly}(\log|\mathcal{R}|)$.

Security. We define selective IND-CPA security.

Definition 3.35 (Selective IND-Security of RABE)**.** A RABE scheme RABE is said to be selectively IND-secure if $\mathbf{Exp}^{\mathsf{rabe\text{-}0}}_{\mathsf{RABE},\mathcal{A}}(1^\lambda) \approx_c \mathbf{Exp}^{\mathsf{rabe\text{-}1}}_{\mathsf{RABE},\mathcal{A}}(1^\lambda)$ for all PPT adversaries $\mathcal{A}$, where $\mathbf{Exp}^{\mathsf{rabe\text{-}b}}_{\mathsf{RABE},\mathcal{A}}$ for $b \in \{0,1\}$ proceeds as follows.

- *Setup.* Run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$ and send crs to $\mathcal{A}$. Initialize the auxiliary input $\mathsf{aux} := \bot$, the master public key $\mathsf{mpk} := \bot$, a counter $i_{\mathsf{reg}} := 0$ to keep track of QRegHK queries, an empty set $\mathcal{C} := \varnothing$ and empty dictionaries $\mathcal{D}, \mathcal{R}$.
- *Query.* Repeat the following for arbitrarily many rounds determined by $\mathcal{A}$. In each round, $\mathcal{A}$ has three options.
  - $\mathsf{QRegCK}(\mathsf{pk}, y)$: upon $\mathcal{A}$ submitting a public key pk and an attribute $y \in \mathcal{Y}_\lambda$, run $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, y)$ and return $(\mathsf{mpk}', \mathsf{aux}')$ to $\mathcal{A}$. Set $\mathsf{mpk} := \mathsf{mpk}'$, $\mathsf{aux} := \mathsf{aux}'$, $\mathcal{D}[\mathsf{pk}] := \mathcal{D}[\mathsf{pk}] \cup \{y\}$ and add pk to $\mathcal{C}$.
  - $\mathsf{QRegHK}(y)$: upon $\mathcal{A}$ submitting an attribute $y \in \mathcal{Y}_\lambda$, run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{aux})$, $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, y)$ and return $(i_{\mathsf{reg}}, \mathsf{mpk}', \mathsf{aux}', \mathsf{pk})$ to $\mathcal{A}$. Set $\mathsf{mpk} := \mathsf{mpk}'$, $\mathsf{aux} := \mathsf{aux}'$, $\mathcal{D}[\mathsf{pk}] := \mathcal{D}[\mathsf{pk}] \cup \{y\}$, $\mathcal{R}[i_{\mathsf{reg}}] := (\mathsf{pk}, \mathsf{sk})$ and $i_{\mathsf{reg}} := i_{\mathsf{reg}} + 1$.
  - $\mathsf{QCorHK}(j)$: upon $\mathcal{A}$ submitting an index $j \in [i_{\mathsf{reg}}]$, retrieve $(\mathsf{pk}, \mathsf{sk}) := \mathcal{R}[j]$ and return sk to $\mathcal{A}$. Add pk to $\mathcal{C}$.
- *Challenge.* The adversary submits a pair of attribute-messages $(x_0, \mu_0), (x_1, \mu_1) \in \mathcal{X}_\lambda \times \mathcal{M}_\lambda$. Run $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_b, \mu_b)$ and return ct^* to $\mathcal{A}$. In the case of RABE, we have $x_0 = x_1 = x^*$.
- *Guess.* The adversary outputs a bit $b' \in \{0,1\}$. The outcome of the experiment is b' if $R(x_0, y) = R(x_1, y) = 1$ for all $y \in \bigcup_{\mathsf{pk} \in \mathcal{C}} \mathcal{D}[\mathsf{pk}]$. Otherwise, the outcome is set to $\bot$.

Hohenberger et al. [HLWW23] showed that the simpler *slotted* RABE primitive (Definition 5.1) can be generically upgraded to full-fledged RABE.

Fact 3.36. *If there exists a* Sel-IND*-secure sRABE (sRPE) scheme for a message space $\mathcal{M}_\lambda$ and a relation R, then there exists a* Sel-IND*-secure RABE (RPE) scheme for the same message space and the same relation.*

3.13 Poly-Domain Obfuscation for Pseudorandom Functionalities

We recall the definition of poly-domain obfuscation for pseudorandom functionalities (pPRIO) as defined in [AKY24c].

Definition 3.37. A pPRIO scheme supporting any poly-size circuit consists of the following algorithms:

$\mathsf{Obf}(1^\lambda, C) \to \widehat{C}$**:** On input a (unary encoded) security parameter λ and a circuit $C\colon [N] \to [M]$ with size $|C| \le S$ for some arbitrary polynomial $S = S(\lambda)$, it outputs an obfuscated circuit $\widehat{C}$. We consider that the Obf algorithm can be decomposed into offline and online phases.

$\mathsf{ObfOff}(1^\lambda, 1^S) \to (\widehat{C}_{\mathsf{off}}, \mathsf{st})$**:** On input a (unary encoded) security parameter λ and circuit size bound S, it outputs $\widehat{C}_{\mathsf{off}}$ and a state st.

$\mathsf{ObfOn}(\mathsf{st}, C) \to \widehat{C}_{\mathsf{on}}$**:** On input a state st and a circuit C, it outputs $\widehat{C}_{\mathsf{on}}$.

With the above decomposition, the obfuscation algorithm outputs $\widehat{C} := (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on}})$.

$\mathsf{Eval}(\widehat{C}, x) \to y$**:** On input an obfuscated circuit $\widehat{C}$ and an input $x \in [N]$, it outputs $y \in [M]$.

We require a pPRIO scheme to satisfy the following correctness and security properites.

Definition 3.38 (Correctness)**.** A pPRIO scheme is correct, if for all $\lambda, N, M \in \mathbb{N}$, for any $C\colon [N] \to [M]$, $S = S(\lambda)$ such that $|C| \le S$ and every input $x \in [N]$, we have

$$\Pr\left[\mathsf{Eval}(\widehat{C}, x) = C(x) \mid \widehat{C} := (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on}}), (\widehat{C}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{ObfOff}(1^\lambda, 1^S), \widehat{C}_{\mathsf{on}} \leftarrow \mathsf{ObfOn}(\mathsf{st}, C)\right] = 1,$$

where the probability is taken over the random coins of the obfuscator Obf.

Definition 3.39 (Security)**.** Let Samp be a PPT algorithm that on input 1^λ, it outputs

$$\left(\mathsf{aux}, 1^{N_1+\cdots+N_Q}, 1^S, C_1, \ldots, C_Q\right), \quad \text{where} \quad C_i\colon [N_i] \to [M_i],\ |C_i| \le S, \text{ for all } i \in [Q],$$

where we enforce Samp to output $1^{N_1+\cdots+N_Q}$ to make sure that all N_i are bounded by $\mathsf{poly}(\lambda)$. We then require that

$$\begin{aligned}
&\textbf{if} \quad \left(\mathsf{aux}, \{C_1(i)\}_{i\in[N_1]}, \ldots, \{C_Q(i)\}_{i\in[N_Q]}\right) \approx_c \left(\mathsf{aux}, \{\Delta_{1,i}\}_{i\in[N_1]}, \ldots, \{\Delta_{Q,i}\}_{i\in[N_Q]}\right)\\
&\textbf{then} \quad \left(\mathsf{aux}, \widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on},1}, \ldots, \widehat{C}_{\mathsf{on},Q}\right) \approx_c \left(\mathsf{aux}, \widehat{C}_{\mathsf{off}}, \delta_1 \xleftarrow{\$} \mathcal{CT}_1, \ldots, \delta_Q \xleftarrow{\$} \mathcal{CT}_Q\right),
\end{aligned}$$

where

$$\begin{aligned}
&\Delta_{k,i} \xleftarrow{\$} [M_k], \text{ for } k \in [Q], i \in [N_k],\\
&(\widehat{C}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{ObfOff}(1^\lambda, 1^S),\\
&\widehat{C}_{\mathsf{on},k} \leftarrow \mathsf{ObfOn}(\mathsf{dig}_k, C_k) \text{ for } k \in [Q],
\end{aligned}$$

and $\mathcal{CT}_k$ denotes the set of binary strings of the same length as the output of $\mathsf{ObfOn}(\mathsf{st}, C_k)$ algorithm.

Fact 3.40 ([AKY24c])**.** *Assuming LWE and prFE, there exists a secure pPRIO scheme satisfying*

$$|\widehat{C}_{\mathsf{off}}| = \mathsf{poly}(S, \lambda),\ |\widehat{C}_{\mathsf{on}}| = \mathsf{poly}(S, \lambda),$$

where $\widehat{C} := (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on}}) \leftarrow \mathsf{Obf}(1^\lambda, C)$ *for a circuit* $C\colon [N] \to [M]$*, whose size is bounded by* $S = S(\lambda)$*.*

3.14 Laconic Poly-Domain Obfuscation for Pseudorandom Functionalities

Below, we give our definition of laconic poly-domain pseudorandom obfuscation (laconic pPRIO) with *global setup*. The definition of (plain) laconic pPRIO (as per [AKY24b]) can be obtained by dropping the setup algorithm and replacing all occurrences of the CRS with 1^λ.

Definition 3.41 (Laconic pPRIO (with Global Setup))**.** A laconic pPRIO scheme with global setup supporting any poly-size circuit consists of the following algorithms:

$\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$**:** On input a security parameter 1^λ, it outputs a common reference string crs.

$\mathsf{Digest}(\mathsf{crs}, X = \{X_i\}_{i\in[N]}) \to \mathsf{dig}$**:** On input a common reference string crs and an input space X of the form $X = \{X_i \in \{0,1\}^{\ell_{\mathsf{in}}}\}_{i\in[N]}$ for some $\ell_{\mathsf{in}} = \ell_{\mathsf{in}}(\lambda)$ and $N \in \mathbb{N}$ (we assume that X encodes the information on ℓ_{in} and N and one can retrieve them efficiently), it outputs a digest string dig.

$\mathsf{Obf}(\mathsf{crs}, \mathsf{dig}, C) \to \widehat{C}$**:** On input a common reference string crs, a digest string dig and a circuit $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ of size $|C| = S$, it outputs an obfuscated circuit $\widehat{C}$. We decompose this algorithm into an offline and an online phase.

- $\mathsf{ObfOff}(\mathsf{crs}, 1^S) \to (\widehat{C}_{\mathsf{off}}, \mathsf{st})$**:** On input a common reference string crs and circuit size S, it outputs $\widehat{C}_{\mathsf{off}}$ and a state st.
- $\mathsf{ObfOn}(\mathsf{crs}, \mathsf{st}, \mathsf{dig}, C) \to \widehat{C}_{\mathsf{on}}$**:** On input a common reference string crs, a state st, a digest string dig and a circuit C, it outputs $\widehat{C}_{\mathsf{on}}$.

With the above decomposition, the obfuscation algorithm outputs $\widehat{C} := (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on}})$.

$\mathsf{Eval}(\mathsf{crs}, X, \widehat{C}) \to Y$**:** On input a common reference string crs, input $X = \{X_i \in \{0,1\}^{\ell_{\mathsf{in}}}\}_{i\in[N]}$ and obfuscated circuit $\widehat{C}$, it outputs $Y = \{Y_i \in \{0,1\}^{\ell_{\mathsf{out}}}\}_{i\in[N]}$.

We require a laconic pPRIO with global to satisfy the following correctness, compactness and security properties.

Definition 3.42 (Correctness)**.** A laconic pPRIO scheme with global setup is correct, if for all $\ell_{\mathsf{in}}, N \in \mathbb{N}$, for any $X = \{X_i \in \{0,1\}^{\ell_{\mathsf{in}}}\}_{i\in[N]}$ and $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$, such that $|C| \le S$, for an arbitrary polynomial $S = S(\lambda)$, we have

$$\Pr\left[\mathsf{Eval}(\mathsf{crs}, X, (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on}})) = \{C(X_i)\}_{i\in[N]} \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{dig} \leftarrow \mathsf{Digest}(\mathsf{crs}, X = \{X_i\}_{i\in[N]}) \\ (\widehat{C}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{ObfOff}(\mathsf{crs}, 1^S) \\ \widehat{C}_{\mathsf{on}} \leftarrow \mathsf{ObfOn}(\mathsf{crs}, \mathsf{st}, \mathsf{dig}, C) \end{array}\right] = 1$$

Definition 3.43 (Compactness)**.** A laconic pPRIO with global setup scheme is compact, if for all $\ell_{\mathsf{in}}, N \in \mathbb{N}$, for any $X = \{X_i \in \{0,1\}^{\ell_{\mathsf{in}}}\}_{i\in[N]}$, $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ and $\mathsf{dig} \leftarrow \mathsf{Digest}(\mathsf{crs}, X)$, it holds that $|\mathsf{crs}| = \mathsf{poly}(\lambda)$ and $|\mathsf{dig}| = \mathsf{poly}(\lambda)$, i.e., the size of crs and dig are independent of N.

Definition 3.44 (Security)**.** Let Samp be a PPT algorithm that on input $(1^\lambda, \mathsf{crs})$, it outputs

$$\left(\mathsf{aux}, 1^S, X_1 = \{X_{1,i}\}_{i\in[N_1]}, \ldots, X_Q = \{X_{Q,i}\}_{i\in[N_Q]}, C_1, \ldots, C_Q\right),$$

where for all $k \in [Q]$, $i \in [N_k]$, $X_{k,i} \in \{0,1\}^{\ell_{\mathsf{in},k}}$, $C_k\colon \{0,1\}^{\ell_{\mathsf{in},k}} \to \{0,1\}^{\ell_{\mathsf{out},k}}$ and $|C_k| \le S$. We require that

$$\begin{aligned} &\textbf{if} \quad \left(\mathsf{aux}, 1^S, \{X_k\}_{k\in[Q]}, \{C_k(X_{k,i})\}_{k\in[Q], i\in[N_k]}\right) \approx_c \left(\mathsf{aux}, 1^S, \{X_k\}_{k\in[Q]}, \{\Delta_{k,i}\}_{k\in[Q], i\in[N_k]}\right) \\ &\textbf{then} \quad \left(\mathsf{aux}, \mathsf{crs}, \{X_k\}_{k\in[Q]}, \widehat{C}_{\mathsf{off}}, \{\widehat{C}_{\mathsf{on},k}\}_{k\in[Q]}\right) \approx_c \left(\mathsf{aux}, \mathsf{crs}, \{X_k\}_{k\in[Q]}, \widehat{C}_{\mathsf{off}}, \{\delta_k\}_{k\in[Q]}\right), \end{aligned}$$

where

- $\Delta_{k,i} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{out},k}}$, for $k \in [Q]$ and $i \in [N_k]$,
- $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\widehat{C}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{ObfOff}(\mathsf{crs}, 1^S)$,
- $\mathsf{dig}_k \leftarrow \mathsf{Digest}(\mathsf{crs}, X_k)$ and $\widehat{C}_{\mathsf{on},k} \leftarrow \mathsf{ObfOn}(\mathsf{crs}, \mathsf{dig}_k, C_k)$ for $k \in [Q]$, and
- $\delta_k \xleftarrow{\$} \mathcal{O}_{\mathsf{on}}$ for $k \in [Q]$, where $\mathcal{O}_{\mathsf{on}}$ is the co-domain of ObfOn algorithm.

We recall the following about plain laconic pPRIO.

Fact 3.45 ([AKY24b])**.** *Assuming LWE and prFE, there exists a secure laconic pPRIO scheme satisfying*

$$|\mathsf{dig}| = O(\lambda),\ |\widehat{C}_{\mathsf{off}}| = \mathsf{poly}(S, \lambda),\ |\widehat{C}_{\mathsf{on}}| = \mathsf{poly}(S, \lambda),$$

where $\mathsf{dig} \leftarrow \mathsf{Digest}(1^\lambda, X = \{X_i \in \{0,1\}^{\ell_{\mathsf{in}}}\}_{i\in[N]})$, $\widehat{C} := (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on}}) \leftarrow \mathsf{Obf}(1^\lambda, \mathsf{dig}, C)$ *for a circuit* $C\colon \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$*, whose size is bounded by* $S = S(\lambda)$.

In Section B, we give a construction of a laconic pPRIO scheme with global setup providing similar properties.

We will make use of the following lemma in our security proofs. Intuitively, the lemma states that if a part of the auxiliary information is pseudorandom in the pre-condition, then it is also pseudorandom in the corresponding post-condition. We give the statement in two variants, for [plain] laconic pPRIO and laconic pPRIO [with global setup].

Lemma 3.46. *Let* $\mathsf{LprIO} = (\boxed{\mathsf{Setup}}, \mathsf{Digest}, \mathsf{Obf}, \mathsf{Eval})$ *be a* [plain] *laconic pPRIO scheme* [with global setup] *and* Samp *be a PPT algorithm that takes as input* $(1^\lambda, \boxed{\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)})$ *and outputs*

$$\left(\mathsf{aux} = (\mathsf{aux}_1, \mathsf{aux}_2) \in \{0,1\}^* \times \mathcal{S}, 1^S, X_1 = \{X_{1,i}\}_{i\in[N_1]}, \ldots, X_Q = \{X_{Q,i}\}_{i\in[N_Q]}, C_1, \ldots, C_Q\right)$$

for some set $\mathcal{S}$*, where for all* $k \in [Q]$*,* $i \in [N_k]$*,* $X_{k,i} \in \{0,1\}^{\ell_{\mathsf{in},k}}$*,* $C_k \colon \{0,1\}^{\ell_{\mathsf{in},k}} \to \{0,1\}^{\ell_{\mathsf{out},k}}$ *and* $|C_k| \le S$*. Let us assume that*

$$\begin{aligned}&\left((\mathsf{aux}_1, \mathsf{aux}_2), 1^S, \{X_k\}_{k\in[Q]}, \{C_k(X_{k,i})\}_{k\in[Q], i\in[N_k]}\right)\\ &\approx_c \left((\mathsf{aux}_1, s), 1^S, \{X_k\}_{k\in[Q]}, \{\Delta_{k,i}\}_{k\in[Q], i\in[N_k]}\right),\end{aligned}$$

where $s \xleftarrow{\$} \mathcal{S}$*,* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $\Delta_{k,i} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{out},k}}$ *for* $k \in [Q]$ *and* $i \in [N_k]$*. Then the security of* LprIO *with respect to* Samp *implies that*

$$\left((\mathsf{aux}_1, \mathsf{aux}_2), \boxed{\mathsf{crs}}, \{X_k\}_{k\in[Q]}, \widehat{C}_{\mathsf{off}}, \{\widehat{C}_{\mathsf{on},k}\}_{k\in[Q]}\right) \approx_c \left((\mathsf{aux}_1, s), \boxed{\mathsf{crs}}, \{X_k\}_{k\in[Q]}, \widehat{C}_{\mathsf{off}}, \{\delta_k\}_{k\in[Q]}\right),$$

where $(\widehat{C}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{ObfOff}(\boxed{\mathsf{crs}}, 1^S)$*,* $\mathsf{dig}_k \leftarrow \mathsf{Digest}(\boxed{\mathsf{crs}}, X_k)$*,* $\widehat{C}_{\mathsf{on},k} \leftarrow \mathsf{ObfOn}(\boxed{\mathsf{crs}}, \mathsf{dig}_k, C_k)$ *and* $\delta_k \xleftarrow{\$} \mathcal{O}_{\mathsf{on}}$ *for* $k \in [Q]$*.*

The statement for [plain] laconic pPRIO was proven in [AKY24b]. The proof for laconic pPRIO with [global setup] can be obtained analogously and is therefore omitted.

4 prCT-Secure sRFE for Unbounded Depth Circuits and Turing Machines

4.1 Definition

Let $\{\mathcal{X}_{\lambda,\mathsf{pub}}\}_{\lambda\in\mathbb{N}}$, $\{\mathcal{X}_{\lambda,\mathsf{pri}}\}_{\lambda\in\mathbb{N}}$ and $\{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be sequences of public input spaces, private input spaces and output spaces, respectively. We consider a functionality $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ where each $\mathcal{F}_\lambda$ contains functions $f_\lambda \colon \mathcal{X}_{\lambda,\mathsf{pub}} \times \mathcal{X}_{\lambda,\mathsf{pri}} \to \mathcal{Y}_\lambda$.

Definition 4.1 (Syntax of Slotted RFE (sRFE))**.** A sRFE scheme for the functionality $\mathcal{F}$ consists of five efficient algorithms:

$\mathsf{Setup}(1^\lambda, \mathsf{param}) \to \mathsf{crs}$**.** On input the security parameter 1^λ and some parameter param specifying $\mathcal{F}$, this algorithm outputs a common reference string crs.

$\mathsf{Gen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$**.** On input the crs, this algorithm outputs a pair of a public and a secret key $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Agg}(\mathsf{crs}, (\mathsf{pk}_i, f_i)_{i\in[L]}) \to (\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]})$**.** On input the crs and L pairs of the form (pk_i, f_i) with $f_i \in \mathcal{F}_\lambda$, this algorithm outputs a master public key mpk and L helper secret keys $\{\mathsf{hsk}_i\}_{i\in[L]}$. We require Agg to be deterministic.

$\mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}) \to \mathsf{ct}$**.** On input the master public key mpk, a public input $x_{\mathsf{pub}} \in \mathcal{X}_{\lambda,\mathsf{pub}}$ and a private input $x_{\mathsf{pri}} \in \mathcal{X}_{\lambda,\mathsf{pri}}$, this algorithm outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}_i, \mathsf{hsk}_i, f_i, \mathsf{ct}, x_{\mathsf{pub}}) \to y \vee \bot$**.** On input a secret key sk_i with corresponding helper secret key hsk_i and registered function f_i as well as a ciphertext ct with corresponding public input x_{pub}, this algorithm outputs a value $y \in \mathcal{Y}_\lambda$ or a special symbol $\bot$ indicating failure. We require Dec to be deterministic.

Correctness. A sRFE scheme is *correct* if for all $\lambda, L \in \mathbb{N}$, $i^* \in [L]$, $(x_{\mathsf{pri}}, x_{\mathsf{pub}}) \in \mathcal{X}_{\lambda,\mathsf{pri}} \times \mathcal{X}_{\lambda,\mathsf{pub}}$ and all $\{f_i\}_{i\in[L]} \subseteq \mathcal{F}_\lambda$, it holds that

$$\Pr\left[y_{i^*} = f_{i^*}(x_{\mathsf{pub}}, x_{\mathsf{pri}}) \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param}) \\ \{(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{crs})\}_{i\in[L]} \\ (\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]}) := \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, f_i)\}_{i\in[L]}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}) \\ y_{i^*} := \mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, f_{i^*}, \mathsf{ct}, x_{\mathsf{pub}}) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of the algorithms Setup, Gen and Enc.

Compactness. A sRFE scheme is *compact* if for all $\lambda, L \in \mathbb{N}$ and $i \in [L]$, it holds that

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

Security. We define our security notion. At a high level, we require ciphertexts to be pseudorandom so long as the output of the functionality itself is pseudorandom.

Definition 4.2 (Sel-prCT Security for sRFE)**.** Given a sRFE scheme sRFE with ciphertext space $\mathcal{CT} = \{\mathcal{CT}_\lambda\}_{\lambda\in\mathbb{N}}$, an interactive PPT algorithm Samp and a PPT adversary $\mathcal{A}$, we define $\mathbf{Exp}^{\mathsf{xxx\text{-}yyy}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}}$, for $\mathsf{xxx} \in \{\boxed{\mathsf{pre}}, \boxed{\mathsf{post}}\}$ and $\mathsf{yyy} \in \{\mathsf{real}, \mathsf{rand}\}$, as follows.

- *Setup.* Launch $\mathsf{Samp}(1^\lambda)$ and receive from it a challenge message $(x^*_{\mathsf{pub}}, x^*_{\mathsf{pri}}) \in \mathcal{X}_{\lambda,\mathsf{pub}} \times \mathcal{X}_{\lambda,\mathsf{pri}}$. Run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param})$ and send crs to Samp. Initialize an empty set $\mathcal{C} := \varnothing$, an empty dictionary $\mathcal{D}$ and a transcript $\boxed{\mathsf{rec}_{\mathsf{pre}} := x^*_{\mathsf{pub}}}$ $\boxed{\mathsf{rec}_{\mathsf{post}} := (x^*_{\mathsf{pub}}, \mathsf{crs})}$.
- *Query.* Repeat the following for arbitrarily many rounds determined by Samp. In each round, Samp has two options.
  - $\mathsf{QGen}()$: run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{crs})$ and return pk to Samp. Set $\mathcal{D}[\mathsf{pk}] := \mathsf{sk}$ $\boxed{\text{and } \mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| \mathsf{pk}}$.
  - $\mathsf{QCor}(\mathsf{pk})$: upon Samp submitting a public key pk, return $\mathcal{D}[\mathsf{pk}]$ to Samp. Furthermore, add pk to $\mathcal{C}$ $\boxed{\text{and set } \mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| (\mathsf{pk}, \mathsf{sk})}$.
- *Challenge.* Samp submits $\mathsf{aux} \in \{0,1\}^*$ and L pairs $\{(\mathsf{pk}^*_i, f_i)\}_{i\in[L]}$ with $\{f_i\}_{i\in[L]} \subseteq \mathcal{F}_\lambda$ and $L \in \mathbb{N}$ being determined by Samp. Let $\mathcal{I}_{\mathsf{cor}} = \{i \in [L] : \mathsf{pk}^*_i \in \mathcal{C}\}$ and $\mathcal{I}_{\mathsf{mal}} = \{i \in [L] : \mathcal{D}[\mathsf{pk}^*_i] = \bot\}$. Compute

$$\begin{array}{rl} \{\delta^*_i := f_i(x^*_{\mathsf{pub}}, x^*_{\mathsf{pri}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, & \{\delta^{\$}_i \xleftarrow{\$} \mathcal{Y}_\lambda\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \\ (\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]}) \leftarrow \mathsf{Agg}(\mathsf{crs}, (\mathsf{pk}^*_i, f_i)_{i\in[L]}), & \\ \mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x^*_{\mathsf{pub}}, x^*_{\mathsf{pri}}), & \mathsf{ct}^{\$} \xleftarrow{\$} \mathcal{CT}_\lambda, \end{array}$$

$$\begin{array}{ll} \text{in } \mathbf{Exp}^{\mathsf{pre\text{-}real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}}: & \mathsf{rec}_{\mathsf{pre}} := \mathsf{rec}_{\mathsf{pre}} \| \left(\mathsf{aux}, \{f_i\}_{i\in[L]}, \{\delta^*_i\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right), \\ \text{in } \mathbf{Exp}^{\mathsf{pre\text{-}rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}}: & \mathsf{rec}_{\mathsf{pre}} := \mathsf{rec}_{\mathsf{pre}} \| \left(\mathsf{aux}, \{f_i\}_{i\in[L]}, \{\delta^{\$}_i\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right), \\ \text{in } \mathbf{Exp}^{\mathsf{post\text{-}real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}}: & \mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| \left(\mathsf{aux}, \{(\mathsf{pk}^*_i, f_i)\}_{i\in[L]}, \mathsf{ct}^*\right), \\ \text{in } \mathbf{Exp}^{\mathsf{post\text{-}rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}}: & \mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| \left(\mathsf{aux}, \{(\mathsf{pk}^*_i, f_i)\}_{i\in[L]}, \mathsf{ct}^{\$}\right). \end{array}$$

- *Guess.* Run the adversary $\mathcal{A}$ on input $(1^\lambda, \boxed{\mathsf{rec}_{\mathsf{pre}}}\ \boxed{\mathsf{rec}_{\mathsf{post}}})$ whose output $b \in \{0,1\}$ is also the outcome of the experiment.

For $\mathsf{xxx} \in \{\mathsf{pre}, \mathsf{post}\}$, we define the advantage function

$$\mathbf{Adv}^{\mathsf{xxx}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathbf{Exp}^{\mathsf{xxx\text{-}real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{xxx\text{-}rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}}(1^\lambda) \to 1\right]\right|.$$

The sRFE scheme sRFE is said to be Sel-prCT secure if for every PPT adversary $\mathcal{A}_{\mathsf{post}}$, there exists another PPT adversary $\mathcal{A}_{\mathsf{pre}}$ such that

$$\mathbf{Adv}^{\mathsf{pre}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{pre}}}(\lambda) \geq \mathbf{Adv}^{\mathsf{post}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}(\lambda)/\mathsf{poly}(\lambda) - \mathsf{negl}(\lambda)$$

and $\mathsf{Time}(\mathcal{A}_{\mathsf{pre}}) \leq \mathsf{Time}(\mathcal{A}_{\mathsf{post}}) \cdot \mathsf{poly}(\lambda)$.

Two-Stage Encryption. For some of our constructions, we are not able to prove full Sel-prCT security as per Definition 4.2. We therefore introduce a slightly weaker security notion that applies to sRFE schemes with a special syntax and is still strong enough for our applications. A sRFE scheme is said to have *two-stage encryption* if the algorithm $\mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}) \to \mathsf{ct}$ can be decomposed into two efficient algorithms with the following syntax

$$\mathsf{EncOff}(\mathsf{mpk}) \to (\mathsf{ct}_{\mathsf{off}}, \mathsf{st})$$
$$\mathsf{EncOn}(\mathsf{st}, x_{\mathsf{pub}}, x_{\mathsf{pri}}) \to \mathsf{ct}_{\mathsf{on}}$$

such that $\mathsf{ct} = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}})$. We define a relaxed security for sRFE schemes with two-stage encryption algorithms, where we only require the online part of ciphertexts to be pseudorandom. Intuitively, this is enough because the offline part is generated without knowledge of $(x_{\mathsf{pub}}, x_{\mathsf{pri}})$, thus it cannot leak any secret even if it is not pseudorandom.

Definition 4.3 (Two-Stage Sel-prCT Security for sRFE)**.** Given a sRFE scheme sRFE, we let $\{\mathcal{CT}_{\lambda,\mathsf{off}}\}_{\lambda\in\mathbb{N}}$ and $\{\mathcal{CT}_{\lambda,\mathsf{on}}\}_{\lambda\in\mathbb{N}}$ denote the offline and online parts of sRFE's ciphertext space $\{\mathcal{CT}_\lambda\}_{\lambda\in\mathbb{N}}$, i.e., $\mathcal{CT}_\lambda = \mathcal{CT}_{\lambda,\mathsf{off}} \times \mathcal{CT}_{\lambda,\mathsf{on}}$ for all $\lambda \in \mathbb{N}$. We say that sRFE is two-stage Sel-prCT secure if it satisfies Sel-prCT security as per Definition 4.2 except for the following modification in the challenge phase $\mathbf{Exp}^{\mathsf{post\text{-}rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}}$: instead of directly sampling $\mathsf{ct}^\$ \xleftarrow{\$} \mathcal{CT}_\lambda$, the challenger in the two-stage variant of the game runs $(\mathsf{ct}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{EncOff}(\mathsf{mpk})$, samples $\mathsf{ct}^\$_{\mathsf{on}} \xleftarrow{\$} \mathcal{CT}_{\lambda,\mathsf{on}}$ and sets $\mathsf{ct}^\$:= (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}^\$_{\mathsf{on}})$.

We note that (full-fledged) Sel-prCT security implies two-stage Sel-prCT security with a trivial EncOff algorithm that outputs $\mathsf{ct}_{\mathsf{off}} = \bot$.

4.2 Construction for Bounded Depth Circuits

We construct a Sel-prCT secure sRFE scheme for the class $\mathcal{C}_{\ell_{\mathsf{pri}},\ell_{\mathsf{dep}},\ell_{\mathsf{out}}} = \{C : \mathcal{X}_{\mathsf{pub}} \times \mathcal{X}_{\mathsf{pri}} \to \mathcal{Y}\}$ containing all circuits with public input space $\mathcal{X}_{\mathsf{pub}} = \{0,1\}^0 = \{\varepsilon\}$ (i.e., $\ell_{\mathsf{pub}} = 0$), private input space $\mathcal{X}_{\mathsf{pri}} = \{0,1\}^{\ell_{\mathsf{pri}}}$, output space $\mathcal{Y} = \{0,1\}^{\ell_{\mathsf{out}}}$ and depth at most ℓ_{dep}. For notational convenience, we will view a circuit $C \in \mathcal{C}_{\ell_{\mathsf{pri}},\ell_{\mathsf{dep}},\ell_{\mathsf{out}}}$ as a function $C : \mathcal{X}_{\mathsf{pri}} \to \mathcal{Y}$ and omit the public input x_{pub} to the algorithms Enc and Dec. We denote the parameters representing the supported class of circuits by $\mathsf{param} = (1^{\ell_{\mathsf{pri}}}, 1^{\ell_{\mathsf{out}}}, 1^{\ell_{\mathsf{dep}}})$.

Looking ahead, this basic construction will serve as the main building block for our Sel-prCT secure sRFE schemes supporting unbounded depth circuits and Turing machines (Constructions 4.8 and 4.11).

Construction 4.4 (Sel-prCT secure sRFE for Bounded Depth Circuits)**.** The construction uses the following building blocks:

- A pPRIO scheme $\mathsf{pPRIO} = \mathsf{pPRIO}.(\mathsf{Obf}, \mathsf{Eval})$ for polynomial-size circuits. We can build pPRIO assuming prFE and LWE (Fact 3.40).
- Two pseudorandom functions

$$\mathsf{PRF}_{\mathsf{coins}} \colon \{0,1\}^\lambda \times [2^\lambda] \to \{0,1\}^\lambda$$
$$\mathsf{PRF}_{\mathsf{noise}} \colon \{0,1\}^\lambda \times [2^\lambda] \to [-q/4 + B; q/4 - B]^{\ell_{\mathsf{out}}}$$

that can both be evaluated by a circuit of depth at most $\ell_{\mathsf{dep}}(\lambda) = \mathsf{poly}(\lambda)$. PRFs can be constructed from one-way functions.

The details of the sRFE scheme for $\mathcal{C}_{\ell_{\mathsf{pri}},\ell_{\mathsf{dep}},\ell_{\mathsf{out}}}$ are as follows.

$\mathsf{Setup}(1^\lambda, \mathsf{param})$**:** Let $\ell_{\mathbf{X}} = (\ell_{\mathsf{pri}} + \lambda)mn\lceil \log q \rceil$. Sample $\mathbf{A}_{\mathsf{att}} \xleftarrow{\$} \mathbb{Z}_q^{n \times (\ell_{\mathbf{X}}+1)m}$, $\mathbf{A}_{\mathsf{user}}, \mathbf{B}_0, \mathbf{B}_1, \mathbf{D} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and output $\mathsf{crs} := (\mathbf{A}_{\mathsf{att}}, \mathbf{A}_{\mathsf{user}}, \mathbf{B}_0, \mathbf{B}_1, \mathbf{D})$.

$\mathsf{Gen}(\mathsf{crs})$**:** Sample $\mathbf{K} \xleftarrow{\$} \{0,1\}^{m\times \ell_{\mathsf{out}}}$, compute $\mathbf{U} = \mathbf{A}_{\mathsf{user}}\mathbf{K}$ and output the key pair $(\mathsf{pk} := \mathbf{U}, \mathsf{sk} := \mathbf{K})$.

$\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, C_i)\}_{i\in[L]})$**:** We assume that $L = 2^\ell$ for some integer $\ell \in \mathbb{N}$.[9] For $i \in [L]$, define the encoding circuit $E[i, C_i]\colon \{0,1\}^{\ell_{\mathsf{pri}}} \times \{0,1\}^\lambda \to \mathbb{Z}_q^{\ell_{\mathsf{out}}}$ as

$$E[i, C_i](\mathbf{x}, \mathsf{sd}_{\mathsf{noise}}) = C_i(\mathbf{x}) \cdot \lfloor q/2 \rceil + \mathsf{PRF}_{\mathsf{noise}}(\mathsf{sd}_{\mathsf{noise}}, i) \,.$$

Then run

$$\mathsf{VEval}_{E[i,C_i]} \leftarrow \mathsf{MakeVEvalCkt}(n, m, q, E[i, C_i]) \,, \qquad \mathbf{H}_{\mathbf{A}_{\mathsf{att}}}^{E[i,C_i]} \leftarrow \mathsf{MEvalC}(\mathbf{A}_{\mathsf{att}}, \mathsf{VEval}_{E[i,C_i]}) \,,$$

and compute $\mathbf{Y}_{\mathsf{bits}(i)} = \mathbf{A}_{E[i,C_i]} + \mathbf{D}\mathbf{G}^{-1}(\mathbf{U}_i)$, where $\mathsf{bits}(i) \in \{0,1\}^\ell$ denotes the bit decomposition of i and $\mathbf{A}_{E[i,C_i]} = \mathbf{A}_{\mathsf{att}} \cdot \mathbf{H}_{\mathbf{A}_{\mathsf{att}}}^{E[i,C_i]}$. Given a string $\mathsf{str} \in \{0,1\}^\ell$, we denote the prefix of length $j \in [0;\ell]$ by $\mathsf{str}_{1:j}$. In particular, $\mathsf{str}_{1:0}$ denotes the empty string ε. For $j = \ell - 1, \ldots, 0$ and $\mathsf{str} \in \{0,1\}^j$, compute

$$\mathbf{Y}_{\mathsf{str}} = \mathbf{B}_0\mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{str}\|0}) + \mathbf{B}_1\mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{str}\|1}) \,.$$

Output $\mathsf{mpk} := (\mathsf{crs}, \mathbf{Y}_\varepsilon, \ell)$ and $\mathsf{hsk}_i := \{\mathbf{Y}_{\mathsf{bits}(i)_{1:j-1}\|b}\}_{j\in[\ell], b\in\{0,1\}}$ for all $i \in [L]$.

$\mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pri}} = \mathbf{x} \in \{0,1\}^{\ell_{\mathsf{pri}}})$**:** Sample $\mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}} \xleftarrow{\$} \{0,1\}^\lambda$, then compute and output $\mathsf{ct} := \widehat{C}_{\mathsf{enc}}$ as

$$\widehat{C}_{\mathsf{enc}} \leftarrow \mathsf{pPRIO.Obf}(1^\lambda, C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}]) \,,$$

where the circuit $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}]$ is defined in Figure 3.

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, C_{i^*}, \mathsf{ct})$**:** Parse $\mathsf{sk}_{i^*} = \mathbf{K}_{i^*}$, $\mathsf{hsk}_{i^*} = \{\mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|b}\}_{j\in[\ell], b\in\{0,1\}}$ and $\mathsf{ct} = \widehat{C}_{\mathsf{enc}}$. Run

$$\left(\{\mathbf{c}_j\}_{j\in[0;\ell]}, \mathbf{c}_{\mathsf{att}}, \mathbf{c}_{\mathsf{user}}, \mathbf{X}\right) \leftarrow \mathsf{pPRIO.Eval}(\widehat{C}_{\mathsf{enc}}, i^*) \,, \qquad \mathbf{H}_{\mathbf{A}_{\mathsf{att}},\mathbf{X}}^{E[i^*,C_{i^*}]} \leftarrow \mathsf{MEvalCX}(\mathbf{A}_{\mathsf{att}}, \mathsf{VEval}_{E[i^*,C_{i^*}]}, \mathbf{X}) \,,$$

where $E[i^*, C_{i^*}]$ is defined as in Agg. Compute

$$\mathbf{z} = \mathbf{c}_0 + \sum_{j\in[\ell]} \mathbf{c}_j^\top \begin{bmatrix} \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|0}) \\ \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|1}) \end{bmatrix} + \mathbf{c}_{\mathsf{att}}^\top \begin{bmatrix} \mathbf{H}_{\mathbf{A}_{\mathsf{att}},\mathbf{X}}^{E[i^*,C_{i^*}]} \\ \mathbf{G}^{-1}(\mathbf{U}_{i^*}) \end{bmatrix} + \mathbf{c}_{\mathsf{user}}^\top \mathbf{K}_{i^*} \,.$$

For $j \in [\ell_{\mathsf{out}}]$, set $y_j = 0$ if $\mathbf{z}[j] \in [-q/4; q/4 - 1]$, and $y_j = 1$ otherwise. Output $\mathbf{y} = (y_1, \ldots, y_{\ell_{\mathsf{out}}})$.

Parameters. We set the parameters of the construction as follows.

$$\beta = \ell_{\mathsf{pri}} \cdot \ell_{\mathsf{out}} \cdot 2^{O(\ell_{\mathsf{dep}} \log q)} \,, \qquad q = 2^{8\lambda}\beta \,, \qquad n = \mathsf{poly}(\lambda, \ell_{\mathsf{dep}}) \,, \qquad m = O(n \log q) \,,$$
$$B = 2^{6\lambda}\beta \,, \qquad \sigma_0 = 2^{4\lambda}\beta \,, \qquad \sigma_1 = 2^{2\lambda} \,.$$

Proposition 4.5 (Correctness and Compactness)**.** *The sRFE scheme for the circuit class $\mathcal{C}_{\ell_{\mathsf{pri}},\ell_{\mathsf{dep}},\ell_{\mathsf{out}}}$ in Construction 4.4 is correct and compact. Specifically, it has the following parameters:*

$$|\mathsf{crs}| = \ell_{\mathsf{pri}} \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda) \,, \qquad |\mathsf{mpk}| = \log\log L + (\ell_{\mathsf{pri}} + \ell_{\mathsf{out}}) \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda) \,,$$
$$|\mathsf{hsk}_i| = \log L \cdot \ell_{\mathsf{out}} \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda) \,, \qquad |\mathsf{ct}| = (\log L + \ell_{\mathsf{pri}} + \ell_{\mathsf{out}}) \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda) \,.$$

Proposition 4.6 (Security)**.** *Let* pPRIO *(Definition 3.39),* $\mathsf{PRF}_{\mathsf{coins}}$ *and* $\mathsf{PRF}_{\mathsf{noise}}$ *be secure. Then the sRFE scheme in Construction 4.4 is* Sel-prCT *secure assuming LWE.*

The proofs of Propositions 4.5 and 4.6 are provided in Sections 4.3 and 4.4.

[9] This assumption is without loss of generality as we can always "pad" the remaining slots to the next power of 2 with dummy users. Specifically, for all $i \in [L+1; 2^{\lceil \log L \rceil}]$ we can set $\mathsf{pk}_i = \mathbf{U}_i := \mathbf{0}_{n\times \ell_{\mathsf{out}}}$ and let $C_i(\mathbf{x})$ be a pseudorandom function $\mathsf{PRF}_{\mathsf{dummy}}(\mathsf{sd} \in \{0,1\}^\lambda, i \in [2^\lambda]) \to \{0,1\}^{\ell_{\mathsf{out}}}$. During the security proof, we can treat these dummy slots as part of the malicious slots, i.e., $[L+1; 2^{\lceil \log L \rceil}] \subseteq \mathcal{I}_{\mathsf{mal}}$.

Input: a slot index $i \in [L]$

Output: a ciphertext $\mathsf{ct} = (\{\mathbf{c}_j\}_{j\in[0;\ell]}, \mathbf{c}_{\mathsf{att}}, \mathbf{c}_{\mathsf{user}}, \mathbf{X})$

Hardwired Values:
- a master public key $\mathsf{mpk} = (\mathsf{crs} = (\mathbf{A}_{\mathsf{att}}, \mathbf{A}_{\mathsf{user}}, \mathbf{B}_0, \mathbf{B}_1, \mathbf{D}), \mathbf{Y}_\varepsilon, \ell)$
- an input vector $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{pri}}}$
- PRF seeds $\mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}} \in \{0,1\}^\lambda$

- Compute $R = \mathsf{PRF}_{\mathsf{coins}}(\mathsf{sd}_{\mathsf{coins}}, i) \in \{0,1\}^\lambda$ and run the subroutine $\mathsf{Sample}(R)$ defined below. Note that random coins R of fixed length λ are sufficient since we can use a PRF to derive longer (pseudo-)random coins if needed. $\mathsf{Sample}(R)$ does the following.
  - Sample $\bar{\mathbf{A}}_{\mathsf{fhe}} \xleftarrow{\$} \mathbb{Z}_q^{(n-1)\times m}$ and $\mathbf{R} \xleftarrow{\$} \{0,1\}^{m\times(\ell_{\mathsf{pri}}+\lambda)m}$.
  - Sample $\{\mathbf{s}_j \xleftarrow{\$} \mathbb{Z}_q^n\}_{j\in[0;\ell+1]\setminus\{\ell\}}$ and $\bar{\mathbf{s}} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$. Set $\mathbf{s}_\ell^\top = (\bar{\mathbf{s}}^\top, -1) \in \mathbb{Z}_q^n$.
  - Sample $\mathbf{e}_0 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma_0}^{\ell_{\mathsf{out}}}$, $\{\mathbf{e}_j \leftarrow \mathcal{D}_{\mathbb{Z},\sigma_1}^{2m}\}_{j\in[\ell]}$, $\mathbf{e}_{\mathsf{att}} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma_1}^{m(\ell_{\mathbf{X}}+2)}$, $\mathbf{e}_{\mathsf{user}} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma_1}^{m}$ and $\mathbf{e}_{\mathsf{fhe}} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma_1}^{m}$.
  - Output $(\bar{\mathbf{A}}_{\mathsf{fhe}}, \mathbf{R}, \{\mathbf{s}_j\}_{j\in[0;\ell+1]}, \{\mathbf{e}_j\}_{j\in[0;\ell]}, \mathbf{e}_{\mathsf{att}}, \mathbf{e}_{\mathsf{user}}, \mathbf{e}_{\mathsf{fhe}})$.
- Compute a GSW encryption $\mathbf{X}$ of $\mathbf{x}$ as

$$\mathbf{A}_{\mathsf{fhe}} = \begin{bmatrix} \bar{\mathbf{A}}_{\mathsf{fhe}} \\ \bar{\mathbf{s}}^\top \bar{\mathbf{A}}_{\mathsf{fhe}} + \mathbf{e}_{\mathsf{fhe}}^\top \end{bmatrix}, \qquad \mathbf{X} = \mathbf{A}_{\mathsf{fhe}}\mathbf{R} - (\mathbf{x}^\top, \mathsf{sd}_{\mathsf{noise}}^\top) \otimes \mathbf{G}\,.$$

  Note. The value $\ell_{\mathbf{X}}$ defined in Setup *corresponds to the bit length of* $\mathbf{X}$.
- Let $(\beta_1, \ldots, \beta_\ell) = \mathsf{bits}(i) \in \{0,1\}^\ell$ be the bit representation of i and denote $\bar{\beta}_j = 1 - \beta_j$ for $j \in [\ell]$. Compute

$$\begin{aligned}
\mathbf{c}_0^\top &= \mathbf{s}_0^\top \mathbf{Y}_\varepsilon + \mathbf{e}_0^\top \\
\{\mathbf{c}_j^\top &= -\mathbf{s}_{j-1}^\top (\mathbf{B}_0 \,\|\, \mathbf{B}_1) + \mathbf{s}_j^\top \left(\bar{\beta}_j \cdot \mathbf{G} \,\|\, \beta_j \cdot \mathbf{G}\right) + \mathbf{e}_j^\top\}_{j\in[\ell]} \\
\mathbf{c}_{\mathsf{att}}^\top &= -\mathbf{s}_\ell^\top (\mathbf{A}_{\mathsf{att}} - \mathsf{bits}(1, \mathbf{X}) \otimes \mathbf{G} \,\|\, \mathbf{D}) + \mathbf{s}_{\ell+1}^\top (\mathbf{0}_{n\times(\ell_{\mathbf{X}}+1)m} \,\|\, \mathbf{G}) + \mathbf{e}_{\mathsf{att}}^\top \\
\mathbf{c}_{\mathsf{user}}^\top &= -\mathbf{s}_{\ell+1}^\top \mathbf{A}_{\mathsf{user}} + \mathbf{e}_{\mathsf{user}}^\top\,.
\end{aligned}$$

- Output $(\mathbf{X}, \{\mathbf{c}_j\}_{j\in[0;\ell]}, \mathbf{c}_{\mathsf{att}}, \mathbf{c}_{\mathsf{user}})$.

Figure 3: Definition of the circuit $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}]$ in Construction 4.4

4.3 Proof of Correctness and Compactness

Proof of Proposition 4.5. We argue that Construction 4.4 is correct and compact.

Correctness. Let $\lambda, L \in \mathbb{N}$, $i^* \in [L]$, $\{C_i\}_{i\in[L]} \subseteq \mathcal{C}_{\ell_{\mathsf{pri}},\ell_{\mathsf{dep}},\ell_{\mathsf{out}}}$ and $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{pri}}}$. Then we have

$$\begin{aligned}
\mathsf{crs} := (\mathbf{A}_{\mathsf{att}}, \mathbf{A}_{\mathsf{user}}, \mathbf{B}_0, \mathbf{B}_1, \mathbf{D}) &\leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param}) \\
\{(\mathsf{pk}_i := \mathbf{U}_i, \mathsf{sk}_i := \mathbf{K}_i) &\leftarrow \mathsf{Gen}(\mathsf{crs})\}_{i\in[L]} \\
\begin{pmatrix} \mathsf{mpk} := (\mathsf{crs}, \mathbf{Y}_\varepsilon, \ell), \\ \{\mathsf{hsk}_i := \{\mathbf{Y}_{\mathsf{bits}(i)_{1:j-1}\|b}\}_{j\in[\ell],b\in\{0,1\}}\}_{i\in[L]} \end{pmatrix} &\leftarrow \mathsf{Agg}(\mathsf{crs}, (\mathsf{pk}_i, C_i)_{i\in[L]}) \\
\mathsf{ct} = \widehat{C}_{\mathsf{enc}} &\leftarrow \mathsf{pPRIO.Obf}(1^\lambda, C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}])\,.
\end{aligned}$$

Here, $\mathbf{Y}_\varepsilon$ is derived as follows. First, define $E[i, C_i](\mathbf{x}, \mathsf{sd}_{\mathsf{noise}}) = C_i(\mathbf{x}) \cdot \lfloor q/2 \rceil + \mathsf{PRF}_{\mathsf{noise}}(\mathsf{sd}_{\mathsf{noise}}, i)$, run

$$\mathsf{VEval}_{E[i,C_i]} \leftarrow \mathsf{MakeVEvalCkt}(n, m, q, E[i, C_i])\,, \qquad \mathbf{H}_{\mathbf{A}_{\mathsf{att}}}^{E[i,C_i]} \leftarrow \mathsf{MEvalC}(\mathbf{A}_{\mathsf{att}}, \mathsf{VEval}_{E[i,C_i]})\,,$$

and set $\mathbf{A}_{E[i,C_i]} = \mathbf{A}_{\mathsf{att}} \cdot \mathbf{H}^{E[i,C_i]}_{\mathbf{A}_{\mathsf{att}}}$ and $\mathbf{Y}_{\mathsf{bits}(i)} = \mathbf{A}_{E[i,C_i]} + \mathbf{D}\mathbf{G}^{-1}(\mathbf{U}_i)$. Then, for $j = \ell-1,\ldots,0$ and $\mathsf{str} \in \{0,1\}^j$, recursively compute $\mathbf{Y}_{\mathsf{str}} = \mathbf{B}_0\mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{str}\|0}) + \mathbf{B}_1\mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{str}\|1})$. Decryption starts by computing $\mathsf{pPRIO.Eval}(\widehat{C}_{\mathsf{enc}}, i^*)$ which, according to the definition of the circuit $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}]$ and by the correctness of pPRIO, yields $(\{\mathbf{c}_j\}_{j\in[0;\ell]}, \mathbf{c}_{\mathsf{att}}, \mathbf{c}_{\mathsf{user}}, \mathbf{X})$ as follows:

$$
\begin{aligned}
\mathbf{X} &= \begin{bmatrix} \mathbf{A}_{\mathsf{fhe}} \\ \mathbf{s}_\ell^\top \mathbf{A}_{\mathsf{fhe}} + \mathbf{e}_{\mathsf{fhe}}^\top \end{bmatrix} \mathbf{R} - (\mathbf{x}^\top, \mathsf{sd}_{\mathsf{noise}}^\top) \otimes \mathbf{G} \\
\mathbf{c}_0^\top &= \mathbf{s}_0^\top \mathbf{Y}_\varepsilon + \mathbf{e}_0^\top \\
\{\mathbf{c}_j^\top &= -\mathbf{s}_{j-1}^\top (\mathbf{B}_0 \parallel \mathbf{B}_1) + \mathbf{s}_j^\top \left(\bar{\beta}_j \cdot \mathbf{G} \parallel \beta_j \cdot \mathbf{G}\right) + \mathbf{e}_j^\top\}_{j\in[\ell]} \\
\mathbf{c}_{\mathsf{att}}^\top &= -\mathbf{s}_\ell^\top (\mathbf{A}_{\mathsf{att}} - \mathsf{bits}(1,\mathbf{X}) \otimes \mathbf{G} \parallel \mathbf{D}) + \mathbf{s}_{\ell+1}^\top (\mathbf{0}_{n\times(\ell_{\mathbf{X}}+1)m} \parallel \mathbf{G}) + \mathbf{e}_{\mathsf{att}}^\top \\
\mathbf{c}_{\mathsf{user}}^\top &= -\mathbf{s}_{\ell+1}^\top \mathbf{A}_{\mathsf{user}} + \mathbf{e}_{\mathsf{user}}^\top ,
\end{aligned}
$$

where

- $\bar{\mathbf{A}}_{\mathsf{fhe}} \xleftarrow{\$} \mathbb{Z}_q^{(n-1)\times m}$ and $\mathbf{R} \xleftarrow{\$} \{0,1\}^{m\times(\ell_{\mathsf{pri}}+\lambda)m}$,
- $\{\mathbf{s}_j \xleftarrow{\$} \mathbb{Z}_q^n\}_{j\in[0;\ell+1]\setminus\{\ell\}}$, $\bar{\mathbf{s}} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$ and $\mathbf{s}_\ell^\top = (\bar{\mathbf{s}}^\top, -1) \in \mathbb{Z}_q^n$,
- $\mathbf{e}_0 \leftarrow \mathcal{D}^{\ell_{\mathsf{out}}}_{\mathbb{Z},\sigma_0}$, $\{\mathbf{e}_j \leftarrow \mathcal{D}^{2m}_{\mathbb{Z},\sigma_1}\}_{j\in[\ell]}$, $\mathbf{e}_{\mathsf{att}} \leftarrow \mathcal{D}^{m(\ell_{\mathbf{X}}+2)}_{\mathbb{Z},\sigma_1}$, $\mathbf{e}_{\mathsf{user}} \leftarrow \mathcal{D}^{m}_{\mathbb{Z},\sigma_1}$ and $\mathbf{e}_{\mathsf{fhe}} \leftarrow \mathcal{D}^{m}_{\mathbb{Z},\sigma_1}$.

Let $(\beta_1,\ldots,\beta_\ell) = \mathsf{bits}(i^*) \in \{0,1\}^{1\times\ell}$. Decryption computes

$$
\mathbf{z} = \mathbf{c}_0 + \sum_{j\in[\ell]} \underbrace{\mathbf{c}_j^\top \begin{bmatrix} \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|0}) \\ \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|1}) \end{bmatrix}}_{\mathbf{z}_j} + \underbrace{\mathbf{c}_{\mathsf{att}}^\top \begin{bmatrix} \mathbf{H}^{E[i^*,C_{i^*}]}_{\mathbf{A}_{\mathsf{att}},\mathbf{X}} \\ \mathbf{G}^{-1}(\mathbf{U}_{i^*}) \end{bmatrix}}_{\mathbf{z}_{\mathsf{att}}} + \underbrace{\mathbf{c}_{\mathsf{user}}^\top \mathbf{K}_{i^*}}_{\mathbf{z}_{\mathsf{user}}} .
$$

Analyzing the terms, we can observe for $j \in [\ell]$ that

$$
\begin{aligned}
\mathbf{z}_j &= \left(-\mathbf{s}_{j-1}^\top (\mathbf{B}_0 \parallel \mathbf{B}_1) + \mathbf{s}_j^\top \left(\bar{\beta}_j \cdot \mathbf{G} \parallel \beta_j \cdot \mathbf{G}\right) + \mathbf{e}_j^\top\right) \begin{bmatrix} \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|0}) \\ \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|1}) \end{bmatrix} \\
&= -\mathbf{s}_{j-1}^\top \mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}} + \mathbf{s}_j^\top \mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|\beta_j} + \underbrace{\mathbf{e}_j^\top \begin{bmatrix} \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|0}) \\ \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|1}) \end{bmatrix}}_{\widehat{\mathbf{e}}_j^\top} ,
\end{aligned}
$$

where we use the fact that $\bar{\beta}_j \mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|0} + \beta_j \mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|1} = \mathbf{Y}_{\mathsf{bits}(i^*)_{1:j-1}\|\beta_j}$. Furthermore, we have that

$$
\begin{aligned}
\mathbf{z}_{\mathsf{att}} &= \left(-\mathbf{s}_\ell^\top (\mathbf{A}_{\mathsf{att}} - \mathsf{bits}(1,\mathbf{X}) \otimes \mathbf{G} \parallel \mathbf{D}) + \mathbf{s}_{\ell+1}^\top (\mathbf{0}_{n\times(\ell_{\mathbf{X}}+1)m} \parallel \mathbf{G}) + \mathbf{e}_{\mathsf{att}}^\top\right) \begin{bmatrix} \mathbf{H}^{E[i^*,C_{i^*}]}_{\mathbf{A}_{\mathsf{att}},\mathbf{X}} \\ \mathbf{G}^{-1}(\mathbf{U}_{i^*}) \end{bmatrix} \\
&= -\mathbf{s}_\ell^\top \left((\mathbf{A}_{\mathsf{att}} - \mathsf{bits}(1,\mathbf{X}) \otimes \mathbf{G}) \mathbf{H}^{E[i^*,C_{i^*}]}_{\mathbf{A}_{\mathsf{att}},\mathbf{X}} + \mathbf{D}\mathbf{G}^{-1}(\mathbf{U}_{i^*})\right) + \mathbf{s}_{\ell+1}^\top \mathbf{U}_{i^*} + \underbrace{\mathbf{e}_{\mathsf{att}}^\top \begin{bmatrix} \mathbf{H}^{E[i^*,C_{i^*}]}_{\mathbf{A}_{\mathsf{att}},\mathbf{X}} \\ \mathbf{G}^{-1}(\mathbf{U}_{i^*}) \end{bmatrix}}_{\widehat{\mathbf{e}}_{\mathsf{att}}^\top} \\
&= -\mathbf{s}_\ell^\top \big(\underbrace{\mathbf{A}_{\mathsf{att}} \mathbf{H}^{E[i^*,C_{i^*}]}_{\mathbf{A}_{\mathsf{att}}}}_{\mathbf{A}_{E[i^*,C_{i^*}]}} - \underbrace{\mathsf{VEval}_{E[i^*,C_{i^*}]}(\mathsf{bits}(\mathbf{X}))}_{=\mathbf{A}_{\mathsf{fhe}}\mathbf{R}_{E[i^*,C_{i^*}]} - \begin{bmatrix} \mathbf{0}_{(n-1)\times\ell_{\mathsf{out}}} \\ E[i^*,C_{i^*}](\mathbf{x},\mathsf{sd}_{\mathsf{noise}}) \end{bmatrix}} + \mathbf{D}\mathbf{G}^{-1}(\mathbf{U}_{i^*})\big) + \mathbf{s}_{\ell+1}^\top \mathbf{U}_{i^*} + \widehat{\mathbf{e}}_{\mathsf{att}}^\top \\
&= -\mathbf{s}_\ell^\top \mathbf{A}_{E[i^*,C_{i^*}]} + E[i^*,C_{i^*}](\mathbf{x},\mathsf{sd}_{\mathsf{noise}}) + \mathbf{e}_{\mathsf{fhe}}^\top \mathbf{R}_{E[i^*,C_{i^*}]} - \mathbf{s}_\ell^\top \mathbf{D}\mathbf{G}^{-1}(\mathbf{U}_{i^*}) + \mathbf{s}_{\ell+1}^\top \mathbf{U}_{i^*} + \widehat{\mathbf{e}}_{\mathsf{att}}^\top
\end{aligned}
$$

$$= -\mathbf{s}_\ell^\top \mathbf{Y}_{\mathsf{bits}(i^*)} + C_{i^*}(\mathbf{x})\lfloor q/2 \rfloor + \mathsf{PRF}_{\mathsf{noise}}(\mathsf{sd}_{\mathsf{noise}}, i^*) + \mathbf{s}_{\ell+1}^\top \mathbf{U}_{i^*} + \mathbf{e}_{\mathsf{fhe}}^\top \mathbf{R}_{E[i^*,C_{i^*}]} + \widehat{\mathbf{e}}_{\mathsf{att}}^\top .$$

Finally, it is straightforward to see that

$$\mathbf{z}_{\mathsf{user}} = (-\mathbf{s}_{\ell+1}^\top \mathbf{A}_{\mathsf{user}} + \mathbf{e}_{\mathsf{user}}^\top)\mathbf{K}_{i^*} = -\mathbf{s}_{\ell+1}^\top \mathbf{U}_{i^*} + \underbrace{\mathbf{e}_{\mathsf{user}}^\top \mathbf{K}_{i^*}}_{\widehat{\mathbf{e}}_{\mathsf{user}}^\top} .$$

Putting all together, we obtain that

$$\mathbf{z} = C_{i^*}(\mathbf{x})\lfloor q/2 \rfloor + \mathsf{PRF}_{\mathsf{noise}}(\mathsf{sd}_{\mathsf{noise}}, i^*) + \underbrace{\mathbf{e}_0^\top + \sum_{j\in[\ell]} \widehat{\mathbf{e}}_j^\top + \mathbf{e}_{\mathsf{fhe}}^\top \mathbf{R}_{E[i^*,C_{i^*}]} + \widehat{\mathbf{e}}_{\mathsf{att}}^\top + \widehat{\mathbf{e}}_{\mathsf{user}}^\top}_{\mathbf{e}^\top} .$$

We note that if $\|\mathbf{e}\| \le B$, then $\|\mathsf{PRF}_{\mathsf{noise}}(\mathsf{sd}_{\mathsf{noise}}, i^*)\| + \|\mathbf{e}\| < (q/4 - B) + B = q/4$ and Dec would output $C_{i^*}(\mathbf{x})$ correctly. Hence, it suffices to show that $\|\mathbf{e}\| \le B$ with probability at least $1 - \mathsf{negl}(\lambda)$. To do so, we first bound the norms of $\mathbf{R}_{E[i^*,C_{i^*}]}$ and $\mathbf{H}^{E[i^*,C_{i^*}]}_{\mathbf{A}_{\mathsf{att}},\mathbf{X}}$. By Lemma 3.9, we have that

$$\left\|\mathbf{R}_{E[i^*,C_{i^*}]}\right\| \le (m+2)^{\ell_{\mathsf{dep}}} \lceil \log q \rceil \cdot m \le (m+2)^{\ell_{\mathsf{dep}}} O(\log q) \le \beta . \tag{11}$$

To bound the norm of $\mathbf{H}^{E[i^*,C_{i^*}]}_{\mathbf{A}_{\mathsf{att}},\mathbf{X}}$, we recall again from Lemma 3.9 that the depth of $\mathsf{VEval}_{E[i^*,C_{i^*}]}$ is bounded by $d \le \ell_{\mathsf{dep}} \cdot O(\log m \log\log q) + O(\log^2 \log q)$. Plugging this into the norm bound of Lemma 3.10 gives

$$\left\|\mathbf{H}^{E[i^*,C_{i^*}]}_{\mathbf{A}_{\mathsf{att}},\mathbf{X}}\right\| \le (m+2)^d \lceil \log q \rceil \le 2^{\ell_{\mathsf{dep}} \cdot O(\log \lambda)} \le \beta . \tag{12}$$

Then a standard calculation using the Gaussian tail bound (Lemma 3.5) shows that with probability at least $1 - \mathsf{negl}(\lambda)$, we have

$$\|\widehat{\mathbf{e}}_1\|, \ldots, \|\widehat{\mathbf{e}}_\ell\|, \|\widehat{\mathbf{e}}_{\mathsf{user}}\|, \|\mathbf{e}_{\mathsf{att}}\|, \|\mathbf{e}_{\mathsf{fhe}}\| \le \sigma_1 \beta = 2^{2\lambda}\beta \qquad \text{and} \qquad \|\widehat{\mathbf{e}}_0\| \le \sigma_0 \beta = 2^{4\lambda}\beta . \tag{13}$$

This implies that $\|\mathbf{e}\| < 2^{6\lambda}\beta = B$ as desired.

Compactness. From our parameter setting, we have $n, m = \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda)$ and $\ell_{\mathbf{X}} = \ell_{\mathsf{pri}} \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda)$. Analyzing the dimensions of the components of crs, we can immediately observe that $|\mathsf{crs}| = \ell_{\mathsf{pri}} \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda)$. Since $\mathsf{mpk} = (\mathsf{crs}, \mathbf{Y}_\varepsilon, \ell)$ additionally contains $\mathbf{Y}_\varepsilon \in \mathbb{Z}_q^{n \times \ell_{\mathsf{out}}}$ and $\ell = \log L$, we have that $|\mathsf{mpk}| = (\ell_{\mathsf{pri}} + \ell_{\mathsf{out}}) \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda) + \log \ell$. Similarly, the size of $\mathsf{hsk}_i = \{\mathbf{Y}_{\mathsf{bits}(i)_{1:j-1}\|b} \in \mathbb{Z}_q^{n \times \ell_{\mathsf{out}}}\}_{j\in\ell, b\in\{0,1\}}$ is $\ell \cdot \ell_{\mathsf{out}} \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda)$. Finally, our instantiation of pPRIO (see Fact 3.40) guarantees that $|\mathsf{ct}| = \mathsf{poly}(S, \lambda)$, where S is a bound on the size of the circuit $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}]$ which is obfuscated during encryption. Analyzing the computations performed by this circuit, we can observe that $S = (\ell + \ell_{\mathsf{pri}} + \ell_{\mathsf{out}}) \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda)$, where

- the factor $\ell \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda)$ is induced by the ℓ ciphertexts $\{\mathbf{c}_j\}_{j\in[\ell]}$ of size $\mathsf{poly}(\ell_{\mathsf{dep}}, \lambda)$ each,
- the factor $\ell_{\mathsf{pri}} \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda)$ stems from the computation of $\mathbf{c}_{\mathsf{att}} \in \mathbb{Z}_q^{m(\ell_{\mathbf{X}}+2)}$ (to see this, we recall that $\ell_{\mathbf{X}} = \ell_{\mathsf{pri}} \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda)$), and
- the factor $\ell_{\mathsf{out}} \cdot \mathsf{poly}(\ell_{\mathsf{dep}}, \lambda)$ appears during the computation of $\mathbf{c}_0 \in \mathbb{Z}_q^{\ell_{\mathsf{out}}}$.

□

4.4 Proof of Security

Proof of Proposition 4.6. Let sRFE denote the sRFE scheme in Construction 4.4. Consider a PPT sampler Samp in the security game $\mathbf{Exp}^{\mathsf{post\text{-}real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$. We recall this game to fix notations.

- *Setup.* Launch $\mathsf{Samp}(1^\lambda)$ and receive from it a challenge message $(x^*_{\mathsf{pub}} = \bot, x^*_{\mathsf{pri}} = \mathbf{x})$. Sample $\mathbf{A}_{\mathsf{att}} \xleftarrow{\$} \mathbb{Z}_q^{n \times (\ell_{\mathbf{X}}+1)m}$, $\mathbf{A}_{\mathsf{user}}, \mathbf{B}_0, \mathbf{B}_1, \mathbf{D} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ and send $\mathsf{crs} = (\mathbf{A}_{\mathsf{att}}, \mathbf{A}_{\mathsf{user}}, \mathbf{B}_0, \mathbf{B}_1, \mathbf{D})$ to Samp. Initialize an empty set $\mathcal{C} := \varnothing$, an empty dictionary $\mathcal{D}$ and a transcript $\mathsf{rec}_{\mathsf{post}} := \mathsf{crs}$.

- *Query.* Repeat the following for arbitrarily many rounds determined by Samp. In each round, Samp has two options.
  - $\mathsf{QGen}()$: sample $\mathbf{K} \xleftarrow{\$} \{0,1\}^{m \times \ell_{\mathsf{out}}}$, compute $\mathbf{U} = \mathbf{A}_{\mathsf{user}}\mathbf{K}$ and return $\mathsf{pk} := \mathbf{U}$ to Samp. Set $\mathcal{D}[\mathsf{pk}] = \mathsf{sk} := \mathbf{K}$ and $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| \mathsf{pk}$.
  - $\mathsf{QCor}(\mathsf{pk})$: upon Samp submitting a public key pk, return $\mathcal{D}[\mathsf{pk}]$ to Samp. Furthermore, add pk to $\mathcal{C}$ and set $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| (\mathsf{pk}, \mathsf{sk})$.
- *Challenge.* Samp submits $\mathsf{aux} \in \{0,1\}^*$ and $\{(\mathsf{pk}_i^*, C_i)\}_{i \in [L]}$. Parse $\mathsf{pk}_i^* = \mathbf{U}_i$. For $i \in [L]$, define the encoding circuit $E[i, C_i] \colon \{0,1\}^{\ell_{\mathsf{pri}}} \times \{0,1\}^{\lambda} \to \mathbb{Z}_q^{\ell_{\mathsf{out}}}$ as

$$E[i, C_i](\mathbf{x}, \mathsf{sd}_{\mathsf{noise}}) = C_i(\mathbf{x}) \cdot \lfloor q/2 \rceil + \mathsf{PRF}_{\mathsf{noise}}(\mathsf{sd}_{\mathsf{noise}}, i) \,.$$

  Then run

$$\mathsf{VEval}_{E[i,C_i]} \leftarrow \mathsf{MakeVEvalCkt}(n, m, q, E[i, C_i])$$
$$\mathbf{H}_{\mathbf{A}_{\mathsf{att}}}^{E[i,C_i]} \leftarrow \mathsf{MEvalC}(\mathbf{A}_{\mathsf{att}}, \mathsf{VEval}_{E[i,C_i]})$$

  and compute $\mathbf{Y}_{\mathsf{bits}(i)} = \mathbf{A}_{\mathsf{att}} \cdot \mathbf{H}_{\mathbf{A}_{\mathsf{att}}}^{E[i,C_i]} + \mathbf{D}\mathbf{G}^{-1}(\mathbf{U}_i)$. For $j = \ell - 1, \ldots, 0$ and $\mathsf{str} \in \{0,1\}^j$, let

$$\mathbf{Y}_{\mathsf{str}} = \mathbf{B}_0\mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{str}\|0}) + \mathbf{B}_1\mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{str}\|1}) \,.$$

  Define $\mathsf{mpk} := (\mathsf{crs}, \mathbf{Y}_\varepsilon, \ell)$, sample $\mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}} \xleftarrow{\$} \{0,1\}^{\lambda}$ and compute $\mathsf{ct}^* := \widehat{C}_{\mathsf{enc}}$ as follows

$$\widehat{C}_{\mathsf{enc}} \leftarrow \mathsf{pPRIO.Obf}(1^{\lambda}, C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}]) \,,$$

  where $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}]$ is defined in Figure 3. Set $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| (\mathsf{aux}, \{(\mathsf{pk}_i^*, C_i)\}_{i \in [L]}, \mathsf{ct}^*)$.
- *Guess.* Run the adversary $\mathcal{A}_{\mathsf{post}}$ on input $(1^{\lambda}, \mathsf{rec}_{\mathsf{post}})$ whose output $b \in \{0,1\}$ is also the outcome of the experiment.

Let $\mathcal{I}_{\mathsf{cor}} = \{i \in [L] : \mathsf{pk}_i^* \in \mathcal{C}\}$ and $\mathcal{I}_{\mathsf{mal}} = \{i \in [L] : \mathcal{D}[\mathsf{pk}_i^*] = \bot\}$. We need to show that if

$$\left(\mathsf{aux}, \{C_i\}_{i \in [L]}, \{\delta_i^* := C_i(\mathbf{x})\}_{i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}}\right) \approx_c \left(\mathsf{aux}, \{C_i\}_{i \in [L]}, \{\delta_i^{\$} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{out}}}\}_{i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}}\right) , \tag{14}$$

then $\mathbf{Exp}^{\text{post-real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}} \approx_c \mathbf{Exp}^{\text{post-rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$, where $\mathbf{Exp}^{\text{post-rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$ proceeds in the same fashion as the experiment $\mathbf{Exp}^{\text{post-real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$ recalled above except that it replaces ct^* with a random element $\mathsf{ct}^{\$} \xleftarrow{\$} \mathcal{O}$. Here, $\mathcal{O}$ denotes the set of all binary strings having the same length as the output of the algorithm $\mathsf{pPRIO.Obf}(1^{\lambda}, C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}])$.

We invoke the security of pPRIO with respect to a sampler $\mathsf{Samp}_{\mathsf{pPRIO}}(1^{\lambda})$ which works as follows. First, it simulates the setup, query and challenge phase of $\mathbf{Exp}^{\text{post-real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$. Upon Samp submitting aux and tuples $\{(\mathsf{pk}_i^*, C_i)\}_{i \in [L]}$, $\mathsf{Samp}_{\mathsf{pPRIO}}$ outputs

$$\left(1^L, 1^S, \mathsf{aux}_{\mathsf{pPRIO}} := (\mathsf{aux}, \mathsf{rec}'_{\mathsf{post}}), C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}]\right)$$

where 1^S is an upper bound on the size of $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}]$ and $\mathsf{rec}'_{\mathsf{post}}$ is the same as $\mathsf{rec}_{\mathsf{post}}$ except that it does not contain the last component ct^*. Under the security of pPRIO, it suffices to show that

$$\left(\mathsf{aux}_{\mathsf{pPRIO}}, \{\Delta_i^* := C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}](i)\}_{i \in [L]}\right) \approx_c \left(\mathsf{aux}_{\mathsf{pPRIO}}, \{\Delta_i^{\$} \xleftarrow{\$} \mathcal{CT}\}_{i \in [L]}\right) , \tag{15}$$

where $\mathcal{CT} = \mathbb{Z}_q^{n \times (\ell_{\mathsf{pri}} + \lambda)m} \times \mathbb{Z}_q^{1 \times \ell_{\mathsf{out}}} \times (\mathbb{Z}_q^{1 \times 2m})^{\ell} \times \mathbb{Z}_q^{1 \times m(\ell_{\mathbf{X}} + 2)} \times \mathbb{Z}_q^{1 \times m}$ denotes the output space of the encryption circuit $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}]$. More specifically, we parse the image $C_{\mathsf{enc}}[\mathsf{mpk}, \mathbf{x}, \mathsf{sd}_{\mathsf{coins}}, \mathsf{sd}_{\mathsf{noise}}](i)$ as

$(\mathbf{X}_i, \mathbf{c}_{i,0}, \{\mathbf{c}_{i,j}\}_{j\in[\ell]}, \mathbf{c}_{i,\mathsf{att}}, \mathbf{c}_{i,\mathsf{user}})$, where

$$\begin{aligned}
\mathbf{X}_i &= \mathbf{A}_{i,\mathsf{fhe}}\mathbf{R}_i - (\mathbf{x}^\top, \mathsf{sd}_{\mathsf{noise}}^\top) \otimes \mathbf{G} \in \mathbb{Z}_q^{n\times(\ell_{\mathsf{pri}}+\lambda)m} \\
\mathbf{c}_{i,0}^\top &= \mathbf{s}_{i,0}^\top \mathbf{Y}_\varepsilon + \mathbf{e}_{i,0}^\top \in \mathbb{Z}_q^{1\times\ell_{\mathsf{out}}} \\
\left\{\mathbf{c}_{i,j}^\top = -\mathbf{s}_{i,j-1}^\top(\mathbf{B}_0 \parallel \mathbf{B}_1) + \mathbf{s}_{i,j}^\top\left(\bar{\beta}_{i,j}\cdot\mathbf{G} \parallel \beta_{i,j}\cdot\mathbf{G}\right) + \mathbf{e}_{i,j}^\top \in \mathbb{Z}_q^{1\times 2m}\right\}_{j\in[\ell]} \\
\mathbf{c}_{i,\mathsf{att}}^\top &= -\mathbf{s}_{i,\ell}^\top(\mathbf{A}_{\mathsf{att}} - \mathsf{bits}(1,\mathbf{X}_i)\otimes\mathbf{G} \parallel \mathbf{D}) + \mathbf{s}_{i,\ell+1}^\top(\mathbf{0}_{n\times(\ell_{\mathbf{X}}+1)m} \parallel \mathbf{G}) + \mathbf{e}_{i,\mathsf{att}}^\top \in \mathbb{Z}_q^{1\times m(\ell_{\mathbf{X}}+2)} \\
\mathbf{c}_{i,\mathsf{user}}^\top &= -\mathbf{s}_{i,\ell+1}^\top \mathbf{A}_{\mathsf{user}} + \mathbf{e}_{i,\mathsf{user}}^\top \in \mathbb{Z}_q^{1\times m}\,.
\end{aligned}$$

Here, we parse the output of $\mathsf{Sample}(R_i)$ as $(\bar{\mathbf{A}}_{i,\mathsf{fhe}}, \mathbf{R}_i, \{\mathbf{s}_{i,j}\}_{j\in[0;\ell+1]}, \{\mathbf{e}_{i,j}\}_{j\in[0;\ell]}, \mathbf{e}_{i,\mathsf{att}}, \mathbf{e}_{i,\mathsf{user}}, \mathbf{e}_{i,\mathsf{fhe}}) \leftarrow \mathsf{Sample}(R_i)$ when run on input $R_i = \mathsf{PRF}(\mathsf{sd}_{\mathsf{coins}}, i)$, and $\mathbf{A}_{i,\mathsf{fhe}}$ is computed as

$$\mathbf{A}_{i,\mathsf{fhe}} = \begin{bmatrix} \bar{\mathbf{A}}_{i,\mathsf{fhe}} \\ \bar{\mathbf{s}}_i^\top \bar{\mathbf{A}}_{i,\mathsf{fhe}} + \mathbf{e}_{i,\mathsf{fhe}}^\top \end{bmatrix},$$

where $\mathbf{s}_{i,\ell}^\top = (\bar{\mathbf{s}}_i^\top, -1)$. We show Equality (15) via the following sequence of hybrids.

Game G_0: This is the L.H.S. distribution of Equation (15). Note that this distribution is the result of an interactive game between $\mathsf{Samp}_{\mathsf{pPRIO}}$ and Samp.

Game G_1: This is the same as G_0 except that we sample $R_1, \ldots, R_L \xleftarrow{\$} \{0,1\}^\lambda$. Since $\mathsf{sd}_{\mathsf{coins}}$ is not used anywhere else, it follows from the security of $\mathsf{PRF}_{\mathsf{coins}}$ that $\mathsf{G}_0 \approx_c \mathsf{G}_1$.

Game G_2: This is the same as G_1 except that, for all $i \in [L]$, we compute $\mathbf{c}_{i,0}^\top$ as follows

$$\mathbf{c}_{i,0}^\top = -\sum_{j\in[\ell]} \mathbf{c}_{i,j}^\top \mathbf{V}_{i,j} - \mathbf{c}_{i,\mathsf{att}}^\top \mathbf{V}_{i,\mathsf{att}} + \mathbf{s}_{i,\ell+1}^\top \mathbf{U}_i + C_i(\mathbf{x})\lfloor q/2\rceil + \mathsf{PRF}_{\mathsf{noise}}(\mathsf{sd}_{\mathsf{noise}}, i) + \mathbf{e}_{i,0}^\top\,,$$

where

$$\mathbf{V}_{i,j} = \begin{bmatrix} \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i)_{1:j-1}\|0}) \\ \mathbf{G}^{-1}(\mathbf{Y}_{\mathsf{bits}(i)_{1:j-1}\|1}) \end{bmatrix}, \qquad \mathbf{V}_{i,\mathsf{att}} = \begin{bmatrix} \mathbf{H}_{\mathbf{A}_{\mathsf{att}},\mathbf{X}}^{E[i,C_i]} \\ \mathbf{G}^{-1}(\mathbf{U}_i) \end{bmatrix}.$$

We have $\mathsf{G}_1 \approx_s \mathsf{G}_2$. First, let us recall from the proof of correctness (Proposition 4.5) that the overall noise term in $-\sum_{j\in[\ell]} \mathbf{c}_{i,j}^\top \mathbf{V}_{i,j} - \mathbf{c}_{i,\mathsf{att}}^\top \mathbf{V}_{i,\mathsf{att}}$ is

$$(\mathbf{e}')^\top := -\sum_{j\in[\ell]} \mathbf{e}_{i,j}^\top \mathbf{V}_{i,j} - \mathbf{e}_{i,\mathsf{att}}^\top \mathbf{V}_{i,\mathsf{att}} - \mathbf{e}_{i,\mathsf{fhe}}^\top \mathbf{R}_{E[i,C_i]}\,.$$

Using the same analysis as in Equations (11), (12) and (13) from the correctness proof, we conclude that with probability at least $1 - \mathsf{negl}(\lambda)$, we have that $\|\mathbf{e}'\| \le \sigma_1\beta = 2^{2\lambda}\beta$. Then $\mathsf{G}_1 \approx_s \mathsf{G}_2$ follows from the smudging lemma (Lemma 3.6) and the fact that $\mathbf{e}_0$ is sampled from a distribution with Gaussian parameter $\sigma_0 = 2^{4\lambda}\beta$.

Game G_3: This is the same as G_2 except that we sample $\mathbf{c}_{i,j}^\top \xleftarrow{\$} \mathbb{Z}_q^{1\times 2m}$ for all $i \in [L]$ and $j \in [\ell]$. To argue $\mathsf{G}_2 \approx_c \mathsf{G}_3$, we define a sequence of intermediate hybrids $\mathsf{G}_{2,j}$ for $j \in [0;\ell]$, where $\mathsf{G}_{2,j}$ is the same as G_2 except that we sample $\mathbf{c}_{i,j'}^\top \xleftarrow{\$} \mathbb{Z}_q^{1\times 2m}$ for $(i,j') \in [L]\times[j]$. It is not hard to see that $\mathsf{G}_{2,0} \equiv \mathsf{G}_1$ and $\mathsf{G}_{2,\ell} \equiv \mathsf{G}_2$. Furthermore, it holds that $\mathsf{G}_{2,j-1} \approx_c \mathsf{G}_{2,j}$ for $j \in [\ell]$. This immediately follows from the LWE assumption which implies that

$$\mathbf{c}_{i,j}^\top = -\mathbf{s}_{i,j-1}^\top(\mathbf{B}_0 \parallel \mathbf{B}_1) + \mathbf{e}_{i,j}^\top$$

is pseudorandom.

Game G_4: This is the same as G_3 except that we sample $\mathbf{A}_{i,\mathsf{fhe}} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and $\mathbf{c}_{i,\mathsf{att}} \xleftarrow{\$} \mathbb{Z}_q^{m(\ell_{\mathbf{x}}+2)}$ for all $i \in [L]$. To prove $\mathsf{G}_3 \approx_c \mathsf{G}_4$, we define a sequence of intermediate hybrids $\mathsf{G}_{3,i}$ for $i \in [0;L]$, where $\mathsf{G}_{3,i}$ is the same as G_3 except that for $i' \in [i]$, we sample $\mathbf{A}_{i',\mathsf{fhe}} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and $\mathbf{c}_{i',\mathsf{att}} \xleftarrow{\$} \mathbb{Z}_q^{m(\ell_{\mathbf{x}}+2)}$. It is not hard to see that $\mathsf{G}_{3,0} \equiv \mathsf{G}_3$ and $\mathsf{G}_{3,L} \equiv \mathsf{G}_4$. Below, we prove the following claim for all $i \in [L]$.

Claim 4.7. *Assuming the hardness of LWE, we have that* $\mathsf{G}_{3,i-1} \approx_c \mathsf{G}_{3,i}$.

Game G_5: This is the same as G_4 except that we sample $\mathbf{X}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times(\ell_{\mathsf{pri}}+\lambda)m}$ for all $i \in [L]$. We have $\mathsf{G}_4 \approx_s \mathsf{G}_5$ which can be seen as follows. From the leftover hash lemma (Lemma 3.4), it follows for all $i \in [L]$ that the statistical distance between $\mathbf{A}_{i,\mathsf{fhe}}\mathbf{R}_i$ and a uniform matrix $\mathbf{M}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times(\ell_{\mathsf{pri}}+\lambda)m}$ is negligible. This implies that the statistical distance between $\mathbf{A}_{i,\mathsf{fhe}}\mathbf{R}_i - (\mathbf{x}^\top, \mathsf{sd}_{\mathsf{noise}}^\top)\otimes\mathbf{G}$ and $\mathbf{M}_i$ is negligible as adding independent terms does not make the task of distinguishing the two distributions any easier.

Game G_6: Recall that for all $i \in \mathcal{I}_{\mathsf{hon}} := [L] \setminus (\mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}})$, we have $\mathbf{K}_i = \mathcal{D}[\mathbf{U}_i] \neq \bot$ satisfying $\mathbf{A}_{\mathsf{user}}\mathbf{K}_i = \mathbf{U}_i$. The game G_6 is the same as G_5 except that we compute

$$\mathbf{c}_{i,0}^\top = -\sum_{j\in[\ell]} \mathbf{c}_{i,j}^\top \mathbf{V}_{i,j} - \mathbf{c}_{i,\mathsf{att}}^\top \mathbf{V}_{i,\mathsf{att}} - \boxed{\mathbf{c}_{i,\mathsf{user}}^\top \mathbf{K}_i} + C_i(\mathbf{x})\lfloor q/2\rceil + \mathsf{PRF}_{\mathsf{noise}}(\mathsf{sd}_{\mathsf{noise}}, i) + \mathbf{e}_{i,0}^\top$$

for all $i \in \mathcal{I}_{\mathsf{hon}}$. Using the Gaussian tail bound (Lemma 3.5), we note that the noise term in $\mathbf{c}_{i,\mathsf{user}}^\top \mathbf{K}_i$ is bounded by $\|\mathbf{e}_{i,\mathsf{user}}^\top \mathbf{K}_i\| \le \sigma_0\beta = 2^{2\lambda}\beta$ with probability at least $1 - \mathsf{negl}(\lambda)$. Then it follows from an application of the smudging lemma (Lemma 3.6) that $\mathsf{G}_5 \approx_s \mathsf{G}_6$, where we recall that $\mathbf{e}_0$ is sampled from a distribution with Gaussian parameter $\sigma_0 = 2^{4\lambda}\beta$.

Game G_7: This is the same as G_6 except that we sample $\mathbf{c}_{i,\mathsf{user}} \xleftarrow{\$} \mathbb{Z}_q^m$ for all $i \in \mathcal{I}_{\mathsf{hon}}$. We have $\mathsf{G}_6 \approx_c \mathsf{G}_7$ assuming the hardness of LWE.

Game G_8: This is the same as G_7 except that we sample $\mathbf{c}_{i,0} \xleftarrow{\$} \mathbb{Z}_q^{\ell_{\mathsf{out}}}$ for all $i \in \mathcal{I}_{\mathsf{hon}}$. An application of the leftover hash lemma (Lemma 3.4) yields

$$\begin{bmatrix} \mathbf{A}_{\mathsf{user}} \\ \mathbf{c}_{i,\mathsf{user}}^\top \end{bmatrix} \cdot \mathbf{K}_i \approx_s \begin{bmatrix} \mathbf{U}_i \\ \mathbf{u}_i^\top \end{bmatrix},$$

where $\mathbf{U}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times\ell_{\mathsf{out}}}$ and $\mathbf{u}_i \xleftarrow{\$} \mathbb{Z}_q^{\ell_{\mathsf{out}}}$. Thus, we have that $\mathsf{G}_7 \approx_s \mathsf{G}_8$.

Game G_9: This is the same as G_8 except that for $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$, we compute

$$\mathbf{c}_{i,0}^\top = -\sum_{j\in[\ell]} \mathbf{c}_{i,j}^\top \mathbf{V}_{i,j} - \mathbf{c}_{i,\mathsf{att}}^\top \mathbf{V}_{i,\mathsf{att}} + \mathbf{s}_{i,\ell+1}^\top \mathbf{U}_i + C_i(\mathbf{x})\lfloor q/2\rceil + \boxed{R_i'} + \mathbf{e}_{i,0}^\top,$$

where $R_i' \xleftarrow{\$} [-q/4+B; q/4-B]^{1\times\ell_{\mathsf{out}}}$ instead of $R_i' = \mathsf{PRF}_{\mathsf{noise}}(\mathsf{sd}_{\mathsf{noise}}, i)$. Since $\mathsf{sd}_{\mathsf{noise}}$ is not used anywhere else, it follows from the security of $\mathsf{PRF}_{\mathsf{noise}}$ that $\mathsf{G}_8 \approx_c \mathsf{G}_9$.

Game G_{10}: This is the same as G_9 except that we sample $R_i' \xleftarrow{\$} [-q/4; q/4]^{1\times\ell_{\mathsf{out}}}$ instead of $R_i' \xleftarrow{\$} [-q/4+B; q/4-B]^{1\times\ell_{\mathsf{out}}}$ for $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$. We claim that $\mathsf{G}_9 \approx_s \mathsf{G}_{10}$. To see this, we note that the statistical distance between the uniform distributions on $[-q/4+B; q/4-B]$ and $[-q/4; q/4]$ is

$$\frac{1}{2}\left((q/2-2B)\cdot\left|\frac{1}{q/2} - \frac{1}{q/2-2B}\right| + 2B\cdot\left|\frac{1}{q/2}\right|\right) = \frac{4B}{q} \le \frac{\mathsf{poly}(\lambda)}{2^\lambda}$$

which is negligible in λ since B is chosen to be exponentially smaller than $q/4$ by our parameter setting. Then the indistinguishability $\mathsf{G}_9 \approx_s \mathsf{G}_{10}$ follows by a hybrid argument where we change the distribution of the vector $(R_{i_1}' \| \cdots \| R_{i_K}')$ in a coordinate-wise manner. Here, we parse $\mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}} = \{i_j\}_{j\in[K]}$.

Game G_{11}: This is the same as G_{10} except that we sample $\mathbf{c}_{i,0} \xleftarrow{\$} \mathbb{Z}_q^{\ell_{\mathsf{out}}}$ for all $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$. We claim that this change is only conceptual. First, recall that the pre-condition (see Equation (14)) guarantees that

$$\left(\mathsf{aux}, \{C_i\}_{i\in[L]}, \{\delta_i^* := C_i(\mathbf{x})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right) \approx_c \left(\mathsf{aux}, \{C_i\}_{i\in[L]}, \{\delta_i^\$ \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{out}}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right),$$

which implies that

$$\left(\mathsf{aux}, \{C_i\}_{i\in[L]}, \{\Delta_i^* := C_i(\mathbf{x})\lfloor q/2\rceil + R'_i\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right) \approx_c \left(\mathsf{aux}, \{C_i\}_{i\in[L]}, \{\Delta_i^{\$} \xleftarrow{\$} \mathbb{Z}_q^{\ell_{\mathsf{out}}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right),$$

where $R'_i \xleftarrow{\$} [-q/4; q/4]^{1\times\ell_{\mathsf{out}}}$. Since adding independent random terms does not make the task of distinguishing the two distributions any easier, we obtain that $\mathsf{G}_{10} \equiv \mathsf{G}_{11}$.

Game G_{12}: This is the same as G_{11} except that we sample $\mathbf{c}_{i,\mathsf{user}} \xleftarrow{\$} \mathbb{Z}_q^m$ for all $i \in \mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}$. We have $\mathsf{G}_{11} \approx_c \mathsf{G}_{12}$ assuming the hardness of LWE. Finally, we observe that G_{12} corresponds to the R.H.S. distribution of Equation (15) which concludes the proof.

□

We now turn to the proof of the claim.

Proof of Claim 4.7. We show that if there exists an adversary $\mathcal{A}$ who can distinguish between $\mathsf{G}_{3.i-1}$ and $\mathsf{G}_{3,i}$, then there exists a reduction $\mathcal{B}$ that breaks the security of LWE with non-negligible advantage. The reduction works as follows:

- *Setup.* Initially, the LWE challenger sends an LWE challenge $(\mathbf{A}_{\mathsf{LWE}}, \mathbf{b}_{\mathsf{LWE}})$ to $\mathcal{B}$, where

$$\mathbf{A}_{\mathsf{LWE}} = \left(\bar{\mathbf{A}}_{\mathsf{fhe}} \in \mathbb{Z}_q^{(n-1)\times m}, \bar{\mathbf{A}}_{\mathsf{att}} \in \mathbb{Z}_q^{(n-1)\times(\ell_{\mathbf{X}}+1)m}, \bar{\mathbf{D}} \in \mathbb{Z}_q^{(n-1)\times m}\right)$$
$$\mathbf{b}_{\mathsf{LWE}} = \left(\mathbf{b}_{\mathsf{fhe}}^\top \in \mathbb{Z}_q^{1\times m}, \mathbf{b}_{\mathsf{att}}^\top \in \mathbb{Z}_q^{1\times(\ell_{\mathbf{X}}+1)m}, \mathbf{b}_{\mathbf{D}}^\top \in \mathbb{Z}_q^{1\times m}\right).$$

  $\mathcal{B}$ launches $\mathsf{Samp}(1^\lambda)$ and receives $\mathbf{x} \in \mathbb{Z}_q^{\ell_{\mathsf{pri}}}$ in return. It samples $\mathbf{A}_{\mathsf{user}}, \mathbf{B}_0, \mathbf{B}_1 \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ along with $\mathbf{R} \xleftarrow{\$} \{0,1\}^{m\times(\ell_{\mathsf{pri}}+\lambda)m}$, $\underline{\mathbf{a}}_{\mathsf{att}} \xleftarrow{\$} \mathbb{Z}_q^{(\ell_{\mathbf{X}}+1)m}$, $\underline{\mathbf{d}} \xleftarrow{\$} \mathbb{Z}_q^m$ and $\mathsf{sd}_{\mathsf{noise}} \xleftarrow{\$} \{0,1\}^\lambda$. Then $\mathcal{B}$ computes

$$\mathbf{A}_{i,\mathsf{fhe}} = \begin{bmatrix}\bar{\mathbf{A}}_{\mathsf{fhe}}\\ \mathbf{b}_{\mathsf{fhe}}^\top\end{bmatrix}, \qquad \mathbf{X}_i = \mathbf{A}_{i,\mathsf{fhe}}\mathbf{R} - (\mathbf{x}^\top, \mathsf{sd}_{\mathsf{noise}}^\top)\otimes\mathbf{G},$$
$$\mathbf{A}_{\mathsf{att}} = \begin{bmatrix}\bar{\mathbf{A}}_{\mathsf{att}}\\ \underline{\mathbf{a}}_{\mathsf{att}}^\top\end{bmatrix} + \mathsf{bits}(1,\mathbf{X}_i)\otimes\mathbf{G}, \qquad \mathbf{D} = \begin{bmatrix}\bar{\mathbf{D}}\\ \underline{\mathbf{d}}^\top\end{bmatrix},$$

  and returns $\mathsf{crs} = (\mathbf{A}_{\mathsf{att}}, \mathbf{A}_{\mathsf{user}}, \mathbf{B}_0, \mathbf{B}_1, \mathbf{D})$ to Samp.

  Note. The embedding of $\mathbf{x}$ into $\mathbf{A}_{\mathsf{att}}$ is the reason why we require Samp to output the challenge upfront, thus we achieve only selective security.

- *Query.* $\mathcal{B}$ simulates the query phase as in $\mathsf{G}_{3,i-1}$.

- *Challenge.* Upon Samp submitting some auxiliary information $\mathsf{aux} \in \{0,1\}^*$ and $\{(\mathsf{pk}_i^*, C_i)\}_{i\in[L]}$, $\mathcal{B}$ computes $\mathsf{aux}_{\mathsf{pPRIO}}$ and $\{(\mathbf{X}_{i'}, \{\mathbf{c}_{i',j}\}_{j\in[0;\ell]}, \mathbf{c}_{i',\mathsf{att}}, \mathbf{c}_{i',\mathsf{user}})\}_{i'\in[L]}$ as in $\mathsf{G}_{3,i-1}$ except for the components $(\mathbf{X}_i, \mathbf{c}_{i,\mathsf{att}})$. While $\mathbf{X}_i$ was already generated during the setup phase, $\mathcal{B}$ now also computes

$$\mathbf{c}_{i,\mathsf{att}}^\top = -(\mathbf{b}_{\mathsf{att}}^\top - \underline{\mathbf{a}}_{\mathsf{att}}^\top \,\|\, \mathbf{b}_{\mathbf{D}}^\top - \underline{\mathbf{d}}^\top) + \mathbf{s}_{\ell+1}^\top(\mathbf{0}_{n\times(\ell_{\mathbf{X}}+1)m} \,\|\, \mathbf{G}).$$

- *Guess.* $\mathcal{B}$ sends $(\mathsf{aux}_{\mathsf{pPRIO}}, \{(\mathbf{X}_{i'}, \{\mathbf{c}_{i',j}\}_{j\in[0;\ell]}, \mathbf{c}_{i',\mathsf{att}}, \mathbf{c}_{i',\mathsf{user}})\}_{i'\in[L]})$ to the adversary $\mathcal{A}$ and forwards the response $b \in \{0,1\}$ to the LWE challenger.

We note that if the LWE challenger sends $\mathbf{b}_{\mathsf{LWE}}^\top = \bar{\mathbf{s}}\mathbf{A}_{\mathsf{LWE}} + \mathbf{e}_{\mathsf{LWE}}^\top$ (resp. $\mathbf{b}_{\mathsf{LWE}} \xleftarrow{\$} \mathbb{Z}_q^{m+(\ell_{\mathbf{X}}+1)m+m}$), then $\mathcal{B}$ simulates $\mathsf{G}_{3,i-1}$ (resp. $\mathsf{G}_{3,i}$). To see the former, we note that if $\mathbf{b}_{\mathsf{LWE}}^\top = \bar{\mathbf{s}}\mathbf{A}_{\mathsf{LWE}} + \mathbf{e}_{\mathsf{LWE}}^\top$, then $\mathbf{b}_{\mathsf{fhe}}^\top = \bar{\mathbf{s}}^\top\bar{\mathbf{A}}_{\mathsf{fhe}} + \mathbf{e}_{\mathsf{fhe}}^\top$

and $(\mathbf{b}_{\mathsf{att}}^\top \| \mathbf{b}_{\mathbf{D}}^\top) = \bar{\mathbf{s}}^\top(\bar{\mathbf{A}}_{\mathsf{att}} \| \bar{\mathbf{D}}) - \mathbf{e}_{\mathsf{att}}^\top$. Thus, we have

$$
\begin{aligned}
\mathbf{A}_{i,\mathsf{fhe}} &= \begin{bmatrix} \bar{\mathbf{A}}_{\mathsf{fhe}} \\ \mathbf{b}_{\mathsf{fhe}}^\top \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{A}}_{\mathsf{fhe}} \\ \bar{\mathbf{s}}^\top \bar{\mathbf{A}}_{\mathsf{fhe}} + \mathbf{e}_{\mathsf{fhe}}^\top \end{bmatrix} \\
\mathbf{c}_{i,\mathsf{att}}^\top &= -(\mathbf{b}_{\mathsf{att}}^\top - \underline{\mathbf{a}}_{\mathsf{att}}^\top \| \mathbf{b}_{\mathbf{D}}^\top - \underline{\mathbf{d}}^\top) + \mathbf{s}_{\ell+1}^\top(\mathbf{0}_{n\times(\ell_{\mathbf{X}}+1)m} \| \mathbf{G}) \\
&= -\left(\bar{\mathbf{s}}^\top(\bar{\mathbf{A}}_{\mathsf{att}} \| \bar{\mathbf{D}}) - \mathbf{e}_{\mathsf{att}}^\top - (\underline{\mathbf{a}}_{\mathsf{att}}^\top \| \underline{\mathbf{d}}^\top)\right) + \mathbf{s}_{\ell+1}^\top(\mathbf{0}_{n\times(\ell_{\mathbf{X}}+1)m} \| \mathbf{G}) \\
&= -(\bar{\mathbf{s}}^\top, -1) \begin{bmatrix} \bar{\mathbf{A}}_{\mathsf{att}} & \| & \bar{\mathbf{D}} \\ \underline{\mathbf{a}}_{\mathsf{att}}^\top & \| & \underline{\mathbf{d}}^\top \end{bmatrix} + \mathbf{e}_{\mathsf{att}}^\top + \mathbf{s}_{\ell+1}^\top(\mathbf{0}_{n\times(\ell_{\mathbf{X}}+1)m} \| \mathbf{G}) \\
&= -(\bar{\mathbf{s}}^\top, -1)(\mathbf{A}_{\mathsf{att}} - \mathsf{bits}(1, \mathbf{X}_i) \otimes \mathbf{G} \| \mathbf{D}) + \mathbf{s}_{\ell+1}^\top(\mathbf{0}_{n\times(\ell_{\mathbf{X}}+1)m} \| \mathbf{G}) + \mathbf{e}_{\mathsf{att}}^\top .
\end{aligned}
$$

For the latter case, we note that if $\mathbf{b}_{\mathsf{fhe}} \xleftarrow{\$} \mathbb{Z}_q^{(\ell_{\mathsf{pri}}+\lambda)m}$, $\mathbf{b}_{\mathsf{att}} \xleftarrow{\$} \mathbb{Z}_q^{(\ell_{\mathbf{X}}+1)m}$ and $\mathbf{b}_{\mathbf{D}} \xleftarrow{\$} \mathbb{Z}_q^m$, then

- the randomness of $\mathbf{b}_{\mathsf{fhe}}$ implies the randomness of $\mathbf{A}_{\mathsf{fhe}} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, and
- the randomness of $(\mathbf{b}_{\mathsf{att}}^\top \| \mathbf{b}_{\mathbf{D}}^\top)$ implies the randomness of $\mathbf{c}_{i,\mathsf{att}} \xleftarrow{\$} \mathbb{Z}_q^{(\ell_{\mathbf{X}}+2)m}$.

□

4.5 Construction for Unbounded Depth Circuits

In this section, we construct a two-stage Sel-prCT secure sRFE scheme for the class $\mathcal{C}_{\ell_{\mathsf{pub}},\ell_{\mathsf{pri}}} = \{C : \mathcal{X}_{\mathsf{pub}} \times \mathcal{X}_{\mathsf{pri}} \to \{0,1\}\}$ containing all circuits with public input space $\mathcal{X}_{\mathsf{pub}} = \{0,1\}^{\ell_{\mathsf{pub}}}$, private input space $\mathcal{X}_{\mathsf{pri}} = \{0,1\}^{\ell_{\mathsf{pri}}}$ and unbounded depth. This generalizes Construction 4.4 which does not consider public inputs and only supports circuits whose depth is bounded by some fixed parameter ℓ_{dep}.

Construction 4.8 (Two-Stage Sel-prCT Secure sRFE for Unbounded Depth Circuits)**.** The construction uses the following building blocks:

- A blind garbling scheme $\mathsf{bGC} = \mathsf{bGC}.(\mathsf{Garble}, \mathsf{Eval}, \mathsf{Sim})$ with decomposability (Definition 3.16). We assume that the labels and the random coins used by $\{\mathsf{bGC}.\mathsf{Garble}_k\}_k$ and $\{\mathsf{bGC}.\mathsf{Garble}_{\mathsf{inp},k}\}_k$ are in $\{0,1\}^\lambda$. The former is guaranteed by Definition 3.12 and the latter can be achieved without loss of generality by using a PRF to derive longer (pseudo-)random coins if needed. We can instantiate bGC with the required properties assuming one-way functions (Fact 3.17).
- A pseudorandom function $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ with key space, input space and output space being $\{0,1\}^\lambda$. PRF can be instantiated assuming one-way functions.
- A laconic pPRIO scheme $\mathsf{LprIO} = \mathsf{LprIO}.(\mathsf{Setup}, \mathsf{Digest}, \mathsf{ObfOff}, \mathsf{ObfOn}, \mathsf{Eval})$ with global setup and a deterministic LprIO.Digest algorithm. Without loss of generality, the state output by LprIO.ObfOff and the random coins of LprIO.ObfOn are in $\{0,1\}^\lambda$. To achieve the former, we let the state be the random coins used by LprIO.ObfOff which can in turn be replaced with a string in $\{0,1\}^\lambda$ using a PRF. For the latter, we can also use a PRF. We note that Definition 3.43 guarantees that the length of digests is bounded by a fixed polynomial $\ell_{\mathsf{dig}} = \ell_{\mathsf{dig}}(\lambda)$ in the security parameter. We can instantiate LprIO with the desired properties assuming prFE and LWE (Construction B.1, Theorem B.4).
- A Sel-prCT secure sRFE scheme $\mathsf{sRFE} = \mathsf{sRFE}.(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$ for the class $\mathcal{C}_{\ell'_{\mathsf{pri}},\ell'_{\mathsf{dep}},\ell'_{\mathsf{out}}}$ consisting of circuits with public input length $\ell'_{\mathsf{pub}} = 0$, private input length $\ell'_{\mathsf{pri}} = \ell_{\mathsf{pri}} + \ell_{\mathsf{dig}} + 4\lambda$, maximum depth ℓ'_{dep} and output length ℓ'_{out}, where we set ℓ'_{dep} and ℓ'_{out} so that the circuit class contains $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$ defined in Figure 4. We denote the information specifying the circuit class by $\mathsf{param}' = (1^{\ell'_{\mathsf{pri}}}, 1^{\ell'_{\mathsf{dep}}}, 1^{\ell'_{\mathsf{out}}})$. We can construct sRFE with the desired properties assuming prFE and LWE (Construction 4.4).

The details of the two-stage Sel-prCT secure sRFE scheme for the circuit class $\mathcal{C}_{\ell_{\mathsf{pub}},\ell_{\mathsf{pri}}}$ are as follows.

$\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{pub}}}, 1^{\ell_{\mathsf{pri}}})$**:** Run

$$\mathsf{LprIO.crs} \leftarrow \mathsf{LprIO.Setup}(1^\lambda)\,, \qquad \mathsf{sRFE.crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda, \mathsf{param}')\,,$$

and output $\mathsf{crs} := (\mathsf{LprIO.crs}, \mathsf{sRFE.crs})$.

$\mathsf{Gen}(\mathsf{crs})$**:** Parse $\mathsf{crs} = (\mathsf{LprIO.crs}, \mathsf{sRFE.crs})$, run

$$\mathsf{sRFE.}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{sRFE.Gen}(\mathsf{sRFE.crs})\,,$$

and output the key pair $(\mathsf{pk} := \mathsf{sRFE.pk}, \mathsf{sk} := \mathsf{sRFE.sk})$.

$\mathsf{Agg}(\mathsf{crs}, \{\mathsf{pk}_i, C_i\}_{i\in[L]})$**:** Parse $\mathsf{crs} = (\mathsf{LprIO.crs}, \mathsf{sRFE.crs})$ and $\{\mathsf{pk}_i = \mathsf{sRFE.pk}_i\}_{i\in[L]}$, then run

$$\begin{aligned}\left\{\mathsf{dig}_{C_i} \leftarrow \mathsf{LprIO.Digest}\left(\mathsf{LprIO.crs}, \{(k, C_i^{(k)})\}_{k\in[|C_i|]}\right)\right\}_{i\in[L]}\\ \left(\mathsf{sRFE.mpk}, \{\mathsf{sRFE.hsk}_i\}_{i\in[L]}\right) \leftarrow \mathsf{sRFE.Agg}\left(\mathsf{sRFE.crs}, \{(\mathsf{sRFE.pk}_i, C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}])\}_{i\in[L]}\right)\,,\end{aligned}$$

where $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$ is defined in Figure 4. Then output $\mathsf{mpk} := (\mathsf{LprIO.crs}, \mathsf{sRFE.mpk})$ and $\mathsf{hsk}_i := (\mathsf{LprIO.crs}, \mathsf{sRFE.hsk}_i)$ for all $i \in [L]$.

Input:
- a LprIO digest $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}$ and a LprIO state st
- a private input vector $\mathbf{x}_{\mathsf{pri}} \in \{0,1\}^{\ell_{\mathsf{pri}}}$
- PRF seeds $\mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{bgc}} \in \{0,1\}^\lambda$

Output:
- a set of labels $\{\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}$
- LprIO online obfuscations $\widehat{C}_{\mathsf{on,cir}}$ and $\widehat{C}_{\mathsf{on,pub}}$

Hardwired Values:
- a slot index $i \in [L]$
- a CRS LprIO.crs and a LprIO digest dig_{C_i}

- Compute $R_{\mathsf{str}} = \mathsf{PRF}(\mathsf{sd}_{\mathsf{str}}, i)$ for $\mathsf{str} \in \{\mathsf{cir}, \mathsf{pub}, \mathsf{bgc}\}$.
- Run

$$\begin{aligned}\left\{(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) \leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R_{\mathsf{bgc}})\right\}_{k\in[\ell_{\mathsf{pub}}+1;\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}\\ \widehat{C}_{\mathsf{on,cir}} \leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_{C_i}, E_{\mathsf{cir}}[R_{\mathsf{bgc}}]; R_{\mathsf{cir}})\\ \widehat{C}_{\mathsf{on,pub}} \leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, E_{\mathsf{pub}}[R_{\mathsf{bgc}}]; R_{\mathsf{pub}})\,,\end{aligned}$$

where E_{pub} and E_{cir} are defined in Figure 5 and 6.

- Output $\left(\{\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}, \widehat{C}_{\mathsf{on,cir}}, \widehat{C}_{\mathsf{on,pub}}\right)$.

Figure 4: Definition of the circuit $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})$**:** The encryption algorithm proceeds in two steps.

$\mathsf{EncOff}(\mathsf{mpk})$**:** Parse $\mathsf{mpk} = (\mathsf{LprIO.crs}, \mathsf{sRFE.mpk})$ and run

$$(\widehat{C}_{\mathsf{off}}, \mathsf{LprIO.st}) \leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S)\,,$$

where S is the maximum size of the circuits $E_{\mathsf{cir}}[R_{\mathsf{bgc}}]$ and $E_{\mathsf{pub}}[R_{\mathsf{bgc}}]$ defined in Figure 5 and 6. Output $\mathsf{ct}_{\mathsf{off}} := \widehat{C}_{\mathsf{off}}$ and $\mathsf{st} = (\mathsf{LprIO.st}, \mathsf{LprIO.crs}, \mathsf{sRFE.mpk})$.

Input: a tuple $(k, G) \in [|C|] \times \{0,1\}^{4\lambda}$ encoding the information of a gate G in a circuit C
Output: a garbled gate $\widetilde{G}$
Hardwired Values: random coins R_{bgc}

- Compute $\widetilde{G} \leftarrow \mathsf{bGC.Garble}_k(1^\lambda, G; R_{\mathsf{bgc}})$
- Output $\widetilde{G}$.

Figure 5: Definition of the circuit $E_{\mathsf{cir}}[R_{\mathsf{bgc}}]$

Input: a tuple $(k, b) \in [\ell_{\mathsf{pub}}] \times \{0,1\}$ encoding an input
Output: a label $\mathsf{lab}_{k,b}$
Hardwired Values: random coins R_{bgc}

1. Compute $(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) \leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R_{\mathsf{bgc}})$
2. Output $\mathsf{lab}_{k,b}$.

Figure 6: Definition of the circuit $E_{\mathsf{pub}}[R_{\mathsf{bgc}}]$

$\mathsf{EncOn}(\mathsf{st}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})$**:** Parse $\mathsf{st} = (\mathsf{LprIO.st}, \mathsf{LprIO.crs}, \mathsf{sRFE.mpk})$, sample PRF seeds $\mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{bgc}} \xleftarrow{\$} \{0,1\}^\lambda$ and run

$$\begin{aligned} \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} &\leftarrow \mathsf{LprIO.Digest}\big(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k \in [\ell_{\mathsf{pub}}]}\big) \\ \mathsf{sRFE.ct} &\leftarrow \mathsf{sRFE.Enc}\big(\mathsf{sRFE.mpk}, \mathbf{x}'_{\mathsf{pri}}\big)\,, \end{aligned}$$

where $\mathbf{x}'_{\mathsf{pri}} = (\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{bgc}}) \in \{0,1\}^{\ell'_{\mathsf{pri}}}$. Output $\mathsf{ct}_{\mathsf{on}} := \mathsf{sRFE.ct}$.

The final output of $\mathsf{Enc}(\mathsf{mpk}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})$ is $\mathsf{ct} = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}})$.

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, C_{i^*}, \mathsf{ct}, \mathbf{x}_{\mathsf{pub}})$**:** Parse $\mathsf{sk}_{i^*} = \mathsf{sRFE.sk}_{i^*}$, $\mathsf{hsk}^*_i = (\mathsf{LprIO.crs}, \mathsf{sRFE.hsk}_{i^*})$ and $\mathsf{ct} = (\widehat{C}_{\mathsf{off}}, \mathsf{sRFE.ct})$. Run

$$\begin{pmatrix} \{\mathsf{lab}_k\}_{k \in [\ell_{\mathsf{pub}}+1; \ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}, \\ \widehat{C}_{\mathsf{on,cir}}, \widehat{C}_{\mathsf{on,pub}} \end{pmatrix} \leftarrow \mathsf{sRFE.Dec}\big(\mathsf{sRFE.sk}_{i^*}, \mathsf{sRFE.hsk}_{i^*}, C_{\mathsf{reg}}[i^*, \mathsf{LprIO.crs}, \mathsf{dig}_{C_{i^*}}], \mathsf{sRFE.ct}\big)$$

$$\begin{aligned} \widetilde{C}_{i^*} = \{\widetilde{C}^{(k)}_{i^*}\}_{k \in [|C_{i^*}|]} &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, C^{(k)}_{i^*})\}_{k \in [|f_{i^*}|]}, \widehat{C}_{\mathsf{cir}} = (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,cir}})\big) \\ \{\mathsf{lab}_k\}_{k \in [\ell_{\mathsf{pub}}]} &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k \in [\ell_{\mathsf{pub}}]}, \widehat{C}_{\mathsf{pub}} = (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,pub}})\big)\,, \end{aligned}$$

where $C_{\mathsf{reg}}[i^*, \mathsf{LprIO.crs}, \mathsf{dig}_{C_{i^*}}]$ is defined in Figure 4. Output $z \leftarrow \mathsf{bGC.Eval}(\widetilde{C}_{i^*}, \{\mathsf{lab}_k\}_{k \in [\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]})$.

Proposition 4.9 (Correctness and Compactness)**.** *The sRFE scheme for the circuit class $\mathcal{C}_{\ell_{\mathsf{pub}}, \ell_{\mathsf{pri}}}$ in Construction 4.8 is correct and compact. Specifically, it has the following parameters:*

$$\begin{aligned} |\mathsf{crs}| &= \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda) & |\mathsf{mpk}| &= \log\log L + \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda) \\ |\mathsf{hsk}_i| &= \log L \cdot \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda) & |\mathsf{ct}| &= \log L \cdot \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)\,. \end{aligned}$$

Proposition 4.10 (Security)**.** *If* bGC *is simulation secure (Definition 3.14) and blind (Definition 3.15),* PRF *is secure,* LprIO *is secure (Definiton 3.44) and* sRFE *is* Sel-prCT *secure (Definition 4.2), then the sRFE scheme in Construction 4.8 satisfies two-stage* Sel-prCT *security.*

The proofs of Propositions 4.9 and 4.10 can be found in Sections 4.6 and 4.7, respectively.

4.6 Proof of Correctness and Compactness

Proof of Proposition 4.9. We argue that Construction 4.8 is correct and compact.

Correctness. Let $\lambda, L \in \mathbb{N}$, $i^* \in [L]$, $\{C_i\}_{i\in[L]} \subseteq \mathcal{C}_{\ell_{\mathsf{pub}},\ell_{\mathsf{pri}}}$ and $(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}}) \in \{0,1\}^{\ell_{\mathsf{pub}}} \times \{0,1\}^{\ell_{\mathsf{pri}}}$. Then we have

$$\begin{aligned}
\mathsf{crs} := (\mathsf{LprIO.crs}, \mathsf{sRFE.crs}) &\leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param}) \\
\left\{(\mathsf{pk}_i := \mathsf{sRFE.pk}_i, \mathsf{sk}_i := \mathsf{sRFE.sk}_i) \leftarrow \mathsf{Gen}(\mathsf{crs})\right\}_{i\in[L]} & \\
\begin{pmatrix} \mathsf{mpk} := (\mathsf{LprIO.crs}, \mathsf{sRFE.mpk}), \\ \left\{\mathsf{hsk}_i := (\mathsf{LprIO.crs}, \mathsf{sRFE.hsk}_i)\right\}_{i\in[L]} \end{pmatrix} &\leftarrow \mathsf{Agg}(\mathsf{crs}, (\mathsf{pk}_i, C_i)_{i\in[L]}) \,,
\end{aligned}$$

where Agg registers the public key $\mathsf{sRFE.pk}_i$ with respect to $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$ as defined in Figure 4 for $\mathsf{dig}_{C_i} \leftarrow \mathsf{LprIO.Digest}(\mathsf{LprIO.crs}, \{(k, C_i^{(k)})\}_{k\in[|C_i|]})$. The encryption algorithm $\mathsf{Enc}(\mathsf{mpk}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})$ samples PRF seeds $\mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{bgc}} \xleftarrow{\$} \{0,1\}^\lambda$ and runs

$$\begin{aligned}
(\widehat{C}_{\mathsf{off}}, \mathsf{LprIO.st}) &\leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S) \\
\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} &\leftarrow \mathsf{LprIO.Digest}\big(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}\big) \\
\mathsf{sRFE.ct} &\leftarrow \mathsf{sRFE.Enc}\big(\mathsf{sRFE.mpk}, (\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{bgc}})\big) \,,
\end{aligned}$$

where S is the maximum size of the circuits $E_{\mathsf{cir}}[R_{\mathsf{bgc}}]$ and $E_{\mathsf{pub}}[R_{\mathsf{bgc}}]$ defined in Figure 5 and 6. Decryption starts by running sRFE.Dec. From the correctness of sRFE and the definition of $C_{\mathsf{reg}}[i^*, \mathsf{LprIO.crs}, \mathsf{dig}_{C_{i^*}}]$, we obtain that

$$\begin{aligned}
&\mathsf{sRFE.Dec}\big(\mathsf{sRFE.sk}_{i^*}, \mathsf{sRFE.hsk}_{i^*}, C_{\mathsf{reg}}[i^*, \mathsf{LprIO.crs}, \mathsf{dig}_{C_{i^*}}], \mathsf{sRFE.ct}\big) \\
&= C_{\mathsf{reg}}[i^*, \mathsf{LprIO.crs}, \mathsf{dig}_{C_{i^*}}](\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{bgc}}) \\
&= \big(\{\mathsf{lab}_k\}_{k\in[\ell_{\mathsf{pub}}+1;\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}, \widehat{C}_{\mathsf{on,cir}}, \widehat{C}_{\mathsf{on,pub}}\big) \,,
\end{aligned}$$

where

$$\begin{aligned}
\left\{(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) \leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R_{\mathsf{bgc}})\right\}_{k\in[\ell_{\mathsf{pub}}+1;\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]} & \\
\widehat{C}_{\mathsf{on,cir}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_{C_{i^*}}, E_{\mathsf{cir}}[R_{\mathsf{bgc}}]; R_{\mathsf{cir}}) \\
\widehat{C}_{\mathsf{on,pub}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, E_{\mathsf{pub}}[R_{\mathsf{bgc}}]; R_{\mathsf{pub}}) \,.
\end{aligned}$$

Subsequently, decryption evaluates

$$\begin{aligned}
\widetilde{C}_{i^*} = \{\widetilde{C}_{i^*}^{(k)}\}_{k\in[|C_{i^*}|]} &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, C_{i^*}^{(k)})\}_{k\in[|f_{i^*}|]}, \widehat{C}_{\mathsf{cir}} = (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,cir}})\big) \\
\{\mathsf{lab}_k\}_{k\in[\ell_{\mathsf{pub}}]} &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}, \widehat{C}_{\mathsf{pub}} = (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,pub}})\big).
\end{aligned}$$

From the correctness of LprIO it follows that

$$\begin{aligned}
\widetilde{C}_{i^*}^{(k)} = E_{\mathsf{cir}}[R_{\mathsf{bgc}}](k, C_{i^*}^{(k)}) &\leftarrow \mathsf{bGC.Garble}_k(1^\lambda, C_{i^*}^{(k)}; R_{\mathsf{bgc}}) \\
\mathsf{lab}_k = E_{\mathsf{pub}}[R_{\mathsf{bgc}}](k, \mathbf{x}_{\mathsf{pub}}[k]) &= \mathsf{lab}_{k,\mathbf{x}_{\mathsf{pub}}[k]} \,,
\end{aligned}$$

where $(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) \leftarrow \mathsf{bGC.Garble}_k(1^\lambda; R_{\mathsf{bgc}})$. Finally, decryption computes

$$z \leftarrow \mathsf{bGC.Eval}(\widetilde{C}_{i^*}, \{\mathsf{lab}_k\}_{k\in[\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]})$$

and we conclude from the correctness of bGC that $z = C_{i^*}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})$.

Compactness. We start by analyzing the size of the circuit $C_{\mathsf{reg}}[i^*, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$.

- The evaluations of PRF can be performed by circuits of size $\mathsf{poly}(\lambda)$.
- The ℓ_{pri} computations of $\mathsf{bGC.Garble}_{\mathsf{inp},k}$ can each be implemented by a circuit of size $\mathsf{poly}(\lambda)$, so the overall computation can be performed in size $\mathsf{poly}(\lambda, \ell_{\mathsf{pri}})$.
- To bound the size of the first execution of $\mathsf{LprIO.ObfOn}$, let us first bound the size of $E_{\mathsf{cir}}[R_{\mathsf{bgc}}]$. This circuit receives as input an index $k \in [|C|]$ (in binary) and the encoding of a gate $G \in \{0,1\}^{4\lambda}$ and runs the (poly-time) algorithm Garble_k. Thus, it can be implemented by a circuit of size $\mathsf{poly}(\log|C|, \lambda) \le \mathsf{poly}(\lambda)$, where the inequality follows from the fact that $|C| \le 2^\lambda$. Coming back to the evaluation of $\mathsf{LprIO.ObfOn}$, we now observe that the overall input length is a fixed polynomial in λ since $|\mathsf{LprIO.crs}| = \mathsf{poly}(\lambda)$, $|\mathsf{LprIO.st}| = \lambda$ and $|\mathsf{dig}_{C_i}| = \mathsf{poly}(\lambda)$ (guaranteed by Definition 3.43 and Theorem B.4). Therefore, the overall size can be bounded by $\mathsf{poly}(\lambda)$.
- The size of a circuit performing the second execution of $\mathsf{LprIO.ObfOn}$ can be bounded similarly to the first one. First, on input $i \in [\ell_{\mathsf{pub}}]$ (in binary), $E_{\mathsf{pub}}[R_{\mathsf{bgc}}]$ performs the computation of $\mathsf{Garble}_{\mathsf{inp},k}$ which can be implemented by a circuit of size $\mathsf{poly}(\log \ell_{\mathsf{pub}}, \lambda) \le \mathsf{poly}(\lambda)$, where the inequality follows from the fact that $\ell_{\mathsf{pub}} \le 2^\lambda$. Then the overall input length of $\mathsf{LprIO.ObfOn}$ is a fixed polynomial in λ, and the size of the circuit is $\mathsf{poly}(\lambda)$.

Relying on this analysis, we can derive the size of the parameters.

- We have $|\mathsf{crs}| = |\mathsf{LprIO.crs}| + |\mathsf{sRFE.crs}| = \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)$. This follows from our instantiations of LprIO (Construction B.1) and sRFE (Construction 4.4) which guarantee $|\mathsf{LprIO.crs}| = \mathsf{poly}(\lambda)$ and $|\mathsf{sRFE.crs}| = \ell'_{\mathsf{pri}} \cdot \mathsf{poly}(\ell'_{\mathsf{dep}}, \lambda) = \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)$. To see the last equality, we recall that $\ell'_{\mathsf{pri}} = \ell_{\mathsf{pri}} + \ell_{\mathsf{dig}} + 4\lambda = \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)$ (by construction) and $\ell'_{\mathsf{dep}} = \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)$ (by the above analysis of $C_{\mathsf{reg}}[i^*, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$).
- We have $|\mathsf{mpk}| = |\mathsf{LprIO.crs}| + |\mathsf{sRFE.mpk}|$. Plugging in the parameters of our instantiations and using again our size bound on $C_{\mathsf{reg}}[i^*, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$, we obtain that $|\mathsf{LprIO.crs}| = \mathsf{poly}(\lambda)$ and $|\mathsf{sRFE.mpk}| = \log\log L + (\ell'_{\mathsf{pri}} + \ell'_{\mathsf{out}}) \cdot \mathsf{poly}(\ell'_{\mathsf{dep}}, \lambda) = \log\log L + \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)$.
- We have $|\mathsf{hsk}_i| = |\mathsf{LprIO.crs}| + |\mathsf{sRFE.hsk}_i| = \log L \cdot \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)$ which can be obtained in a similar manner as before. In particular, we recall that our instantiation of sRFE gives $|\mathsf{sRFE.hsk}_i| = \log L \cdot \ell'_{\mathsf{out}} \cdot \mathsf{poly}(\ell'_{\mathsf{dep}}, \lambda) = \log L \cdot \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)$.
- Finally, we have $|\mathsf{ct}| = |\widehat{C}_{\mathsf{off}}| + |\mathsf{sRFE.ct}| = \log L \cdot \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)$. For this, we recall that our instantiation of LprIO satisfies $|\widehat{C}_{\mathsf{off}}| = \mathsf{poly}(S, \lambda)$, where S is a size bound on $E_{\mathsf{cir}}[R_{\mathsf{bgc}}]$ and $E_{\mathsf{pub}}[R_{\mathsf{bgc}}]$. By our above analysis, we have $S = \mathsf{poly}(\lambda)$. Furthermore, our instantiation of sRFE satisfies $|\mathsf{sRFE.ct}| = (\log L + \ell'_{\mathsf{pri}} + \ell'_{\mathsf{out}}) \cdot \mathsf{poly}(\ell'_{\mathsf{dep}}, \lambda) = \log L \cdot \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)$.

□

4.7 Proof of Security

Proof of Proposition 4.10. To avoid a clash of variables, we will use the following convention throughout this proof. The sRFE scheme for unbounded depth circuits which is built in Construction 4.8 is denoted by sRFE. On the other hand, the sRFE scheme for bounded depth circuits which serves as a building block and which was previously denoted by sRFE is now denoted by sRFE'. All variables (e.g., keys, inputs, etc.) belonging either to sRFE or sRFE' will be distinguished in the same way. Consider a PPT sampler Samp in the security game $\mathbf{Exp}^{\mathsf{post\text{-}real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$. We recall this game to fix notations.

- *Setup.* Launch $\mathsf{Samp}(1^\lambda)$ and receive from it a challenge message $(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}}) \in \{0,1\}^{\ell_{\mathsf{pub}}} \times \{0,1\}^{\ell_{\mathsf{pri}}}$. Run

$$\mathsf{LprIO.crs} \leftarrow \mathsf{LprIO.Setup}(1^\lambda)\,, \qquad \mathsf{sRFE}'\mathsf{.crs} \leftarrow \mathsf{sRFE}'\mathsf{.Setup}(1^\lambda, \mathsf{param}')\,,$$

and send $\mathsf{crs} := (\mathsf{LprIO.crs}, \mathsf{sRFE'.crs})$ to Samp. Initialize an empty set $\mathcal{C} := \varnothing$, an empty dictionary $\mathcal{D}$ and a transcript $\mathsf{rec}_{\mathsf{post}} := (x_{\mathsf{pub}}, \mathsf{crs})$.

- *Query.* Repeat the following for arbitrarily many rounds determined by Samp. In each round, Samp has two options.
  - QGen(): run $\mathsf{sRFE'.(pk,sk)} \leftarrow \mathsf{sRFE'.Gen}(\mathsf{sRFE'.crs})$ and return $\mathsf{pk} := \mathsf{sRFE'.pk}$ to Samp. Set $\mathcal{D}[\mathsf{pk}] = \mathsf{sk} := \mathsf{sRFE'.sk}$ and $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| \mathsf{pk}$.
  - QCor(pk): upon Samp submitting a public key pk, return $\mathcal{D}[\mathsf{pk}]$ to Samp. Furthermore, add pk to $\mathcal{C}$ and set $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| (\mathsf{pk}, \mathsf{sk})$.
- *Challenge.* Samp submits aux and $\{(\mathsf{pk}_i^*, C_i)\}_{i\in[L]}$. Sample PRF seeds $\mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{bgc}} \xleftarrow{\$} \{0,1\}^\lambda$ and run

$$\begin{aligned}
\left\{\mathsf{dig}_{C_i} \leftarrow \mathsf{LprIO.Digest}\left(\mathsf{LprIO.crs}, \{(k, C_i^{(k)})\}_{k\in[|C_i|]}\right)\right\}_{i\in[L]} \\
\left(\mathsf{sRFE'.mpk}, \{\mathsf{sRFE'.hsk}_i\}_{i\in[L]}\right) \leftarrow \mathsf{sRFE'.Agg}\left(\mathsf{sRFE'.crs}, \{(\mathsf{sRFE'.pk}_i, C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}])\}_{i\in[L]}\right) \\
(\widehat{C}_{\mathsf{off}}, \mathsf{LprIO.st}) \leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S) \\
\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} \leftarrow \mathsf{LprIO.Digest}\left(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}\right) \\
\mathsf{sRFE'.ct} \leftarrow \mathsf{sRFE'.Enc}\left(\mathsf{sRFE'.mpk}, (\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{bgc}})\right).
\end{aligned}$$

  Let $\mathsf{ct}^* = (\mathsf{ct}_{\mathsf{off}} := \widehat{C}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}}^* := \mathsf{sRFE'.ct})$ and $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| (\mathsf{aux}, \{(\mathsf{pk}_i^*, C_i)\}_{i\in[L]}, \mathsf{ct}^*)$.
- *Guess.* Run the adversary $\mathcal{A}_{\mathsf{post}}$ on input $(1^\lambda, \mathsf{rec}_{\mathsf{post}})$ whose output $b \in \{0,1\}$ is also the outcome of the experiment.

We need to show that if

$$\left(\mathsf{aux}, \mathbf{x}_{\mathsf{pub}}, \{C_i\}_{i\in[L]}, \{\delta_i^* := C_i(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right) \approx_c \left(\mathsf{aux}, \mathbf{x}_{\mathsf{pub}}, \{C_i\}_{i\in[L]}, \{\delta_i^* \xleftarrow{\$} \{0,1\}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right), \tag{16}$$

then $\mathbf{Exp}^{\text{post-real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}} \approx_c \mathbf{Exp}^{\text{post-rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$, where $\mathbf{Exp}^{\text{post-rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$ proceeds in the same fashion as the experiment $\mathbf{Exp}^{\text{post-real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$ recalled above except that it replaces $\mathsf{ct}^* = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}}^* := \mathsf{sRFE'.ct})$ with $\mathsf{ct}^{\$} = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}}^{\$} \xleftarrow{\$} \mathsf{sRFE'}.\mathcal{CT}_{\mathsf{on}})$. Here, $\mathsf{sRFE'}.\mathcal{CT}_{\mathsf{on}}$ denotes the online part of the ciphertext space of sRFE′.

We invoke the Sel-prCT security of sRFE′ with respect to a sampler $\mathsf{Samp}_{\mathsf{sRFE'}}(1^\lambda)$ which works as follows.

- *Setup.* Launch $\mathsf{Samp}(1^\lambda)$ and receive from it a challenge message $(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})$. Run

$$\begin{aligned}
\mathsf{LprIO.crs} \leftarrow \mathsf{LprIO.Setup}(1^\lambda) \\
(\widehat{C}_{\mathsf{off}}, \mathsf{LprIO.st}) \leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S) \\
\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} \leftarrow \mathsf{LprIO.Digest}\left(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}\right),
\end{aligned}$$

  sample $\mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{bgc}} \xleftarrow{\$} \{0,1\}^\lambda$ and send $\mathbf{x}'_{\mathsf{pri}} := (\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{bgc}}) \in \{0,1\}^{\ell'_{\mathsf{pri}}}$ to the challenger of sRFE′ to obtain sRFE′.crs in return. Send $\mathsf{crs} = (\mathsf{LprIO.crs}, \mathsf{sRFE'.crs})$ to Samp.
- *Query.* Upon receiving a query from Samp, $\mathsf{Samp}_{\mathsf{sRFE'}}$ makes the same query to the corresponding oracle of the sRFE challenger and forwards the response to Samp.
- *Challenge.* Upon Samp outputting aux and $\{(\mathsf{pk}_i^*, C_i)\}_{i\in[L]}$, $\mathsf{Samp}_{\mathsf{sRFE'}}$ runs

$$\left\{\mathsf{dig}_{C_i} \leftarrow \mathsf{LprIO.Digest}\left(\mathsf{LprIO.crs}, \{(k, C_i^{(k)})\}_{k\in[|C_i|]}\right)\right\}_{i\in[L]}.$$

  Then it outputs $\mathsf{aux}_{\mathsf{sRFE'}} = (\mathsf{aux}, \mathbf{x}_{\mathsf{pub}}, \{C_i\}_{i\in[L]}, \widehat{C}_{\mathsf{off}})$ and $\{(\mathsf{pk}_i^*, C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}])\}_{i\in[L]}$.

Under the security of sRFE′, it suffices to show that

$$\begin{aligned}
&\left(\mathsf{aux}_{\mathsf{sRFE'}}, \{C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]\}_{i\in[L]}, \{\Delta_i^* := C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}](\mathbf{x}'_{\mathsf{pri}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right) \\
&\approx_c \left(\mathsf{aux}_{\mathsf{sRFE'}}, \{C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]\}_{i\in[L]}, \{\Delta_i^* \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{reg}}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right),
\end{aligned} \tag{17}$$

where ℓ_{reg} denotes the output length of $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$. When unfolding the definition of the circuit $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}]$, we can note that

$$C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{C_i}](\mathbf{x}'_{\mathsf{pri}}) = \left(\{\mathsf{lab}_{i,k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}, \widehat{C}_{i,\mathsf{on,cir}}, \widehat{C}_{i,\mathsf{on,pub}}\right),$$

where

$$\begin{aligned}
\left\{(\mathsf{lab}_{i,k,0}, \mathsf{lab}_{i,k,1}) \leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R_{i,\mathsf{bgc}})\right\}_{k\in[\ell_{\mathsf{pub}}+1;\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]} \\
\widehat{C}_{i,\mathsf{on,cir}} \leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_{C_i}, E_{\mathsf{cir}}[R_{i,\mathsf{bgc}}]; R_{i,\mathsf{cir}}) \\
\widehat{C}_{i,\mathsf{on,pub}} \leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, E_{\mathsf{pub}}[R_{i,\mathsf{bgc}}]; R_{i,\mathsf{pub}})
\end{aligned}$$

and $R_{i,\mathsf{bgc}} = \mathsf{PRF}(\mathsf{sd}_{\mathsf{bgc}}, i)$, $R_{i,\mathsf{cir}} = \mathsf{PRF}(\mathsf{sd}_{\mathsf{cir}}, i)$, $R_{i,\mathsf{pub}} = \mathsf{PRF}(\mathsf{sd}_{\mathsf{pub}}, i)$. As the PRF seeds $\mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{bgc}}$ do not appear anywhere else, we can replace these PRF outputs by truly random strings under the security of PRF. Hence, it suffices to prove a variant of Equation (17), where $R_{i,\mathsf{bgc}}$, $R_{i,\mathsf{cir}}$ and $R_{i,\mathsf{pub}}$ for $i \in [L]$ are all replaced by independent uniformly random strings over $\{0,1\}^\lambda$. To show this variant of Equation (17) with truly random coins, we rely on the security of LprIO with respect to a sampler $\mathsf{Samp}_{\mathsf{LprIO}}(\mathsf{LprIO.crs})$ which does the following.

- *Setup.* On input LprIO.crs, $\mathsf{Samp}_{\mathsf{LprIO}}$ launches $\mathsf{Samp}(1^\lambda)$ to receive a challenge message $(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pri}})$ in return. Run

  $$\mathsf{sRFE}'.\mathsf{crs} \leftarrow \mathsf{sRFE}'.\mathsf{Setup}(1^\lambda, \mathsf{param}'),$$

  and send $\mathsf{crs} = (\mathsf{LprIO.crs}, \mathsf{sRFE}'.\mathsf{crs})$ to Samp.
- *Query.* Upon receiving a QGen or QCor query from Samp, $\mathsf{Samp}_{\mathsf{LprIO}}$ simulates the oracles by running the real $\mathsf{sRFE}'.\mathsf{Gen}$ algorithm.
- *Challenge.* Upon Samp outputting aux and $\{(\mathsf{pk}_i^*, C_i)\}_{i\in[L]}$, $\mathsf{Samp}_{\mathsf{LprIO}}$ outputs

  $$\left(\mathsf{aux}_{\mathsf{LprIO}} = (\mathsf{aux}_{\mathsf{LprIO},1}, \mathsf{aux}_{\mathsf{LprIO},2}), 1^S, \{X_{i,\mathsf{cir}}, X_{i,\mathsf{pub}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \{E_{i,\mathsf{cir}}, E_{i,\mathsf{pub}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right),$$

  where $S = \max\left\{|E_{\mathsf{cir}}[R_{i,\mathsf{bgc}}]|, |E_{\mathsf{pub}}[R_{i,\mathsf{bgc}}]|\right\}_{i\in[L]}$ and

  $$\begin{aligned}
  \mathsf{aux}_{\mathsf{LprIO},1} &= \{\mathsf{lab}_{i,k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}, k\in[\ell_{\mathsf{pub}}+1;\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}, & \mathsf{aux}_{\mathsf{LprIO},2} &= \left(\mathsf{aux}, \mathbf{x}_{\mathsf{pub}}, \{C_i\}_{i\in[L]}\right), \\
  X_{i,\mathsf{cir}} &= \{X_{i,k,\mathsf{cir}} = (k, C_i^{(k)})\}_{k\in[|C_i|]}, & E_{i,\mathsf{cir}} &= E_{\mathsf{cir}}[R_{i,\mathsf{bgc}}], \\
  X_{i,\mathsf{pub}} &= \{X_{i,k,\mathsf{pub}} = (k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}, & E_{i,\mathsf{pub}} &= E_{\mathsf{pub}}[R_{i,\mathsf{bgc}}].
  \end{aligned}$$

Let us consider the following equality.

$$\begin{aligned}
&\left(\begin{array}{c}\mathsf{aux}_{\mathsf{LprIO},1}, \mathsf{aux}_{\mathsf{LprIO},2}, 1^S, \{X_{i,\mathsf{cir}}, X_{i,\mathsf{pub}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \\ \{E_{i,\mathsf{cir}}(X_{i,k,\mathsf{cir}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}, k\in[|C_i|]} \cup \{E_{i,\mathsf{pub}}(X_{i,k,\mathsf{pub}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}, k\in[\ell_{\mathsf{pub}}]}\end{array}\right) \\
&\approx_c \left(\begin{array}{c}\{\Gamma_{i,k,\mathsf{pri}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}, k\in[\ell_{\mathsf{pri}}]}, \mathsf{aux}_{\mathsf{LprIO},2}, 1^S, \{X_{i,\mathsf{cir}}, X_{i,\mathsf{pub}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \\ \{\Gamma_{i,k,\mathsf{cir}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}, k\in[|C_i|]} \cup \{\Gamma_{i,k,\mathsf{pub}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}, k\in[\ell_{\mathsf{pub}}]}\end{array}\right),
\end{aligned} \tag{18}$$

where $\Gamma_{i,k,\mathsf{cir}} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{cir}}}$ for $(i,k) \in (\mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}) \times [|C_i|]$, $\Gamma_{i,k,\mathsf{pub}} \xleftarrow{\$} \{0,1\}^\lambda$ for $(i,k) \in (\mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}) \times [\ell_{\mathsf{pub}}]$ and $\Gamma_{i,k,\mathsf{pri}} \xleftarrow{\$} \{0,1\}^\lambda$ for $(i,k) \in (\mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}) \times [\ell_{\mathsf{pri}}]$. Here, ℓ_{cir} denotes the output length of $E_{\mathsf{cir}}[R]$ for any $R \in \{0,1\}^\lambda$ (note that the choice of R is irrelevant for the output length). We can observe that Equation (18) implies Equation (17) under the security of LprIO with respect to $\mathsf{Samp}_{\mathsf{LprIO}}$ and Lemma 3.46.

The rest of the proof is dedicated to proving Equation (18). Unrolling the definitions and applying the decomposability property of bGC (Definition 3.16), we observe that Equation (18) is equivalent to the following

$$\begin{aligned}
&\left(\left\{(\{\mathsf{lab}_{i,k,\mathbf{x}[k]}\}_{k\in[\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}, \widetilde{C}_i)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \mathsf{aux}, \mathbf{x}_{\mathsf{pub}}, \{C_i\}_{i\in[L]}\right) \\
&\approx_c \left(\left\{(\{\Gamma_{i,k,\mathsf{pub}}\}_{k\in[\ell_{\mathsf{pub}}]} \cup \{\Gamma_{i,k,\mathsf{pri}}\}_{k\in[\ell_{\mathsf{pri}}]}, \{\Gamma_{i,k,\mathsf{cir}}\}_{k\in[|C_i|]})\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \mathsf{aux}, \mathbf{x}_{\mathsf{pub}}, \{C_i\}_{i\in[L]}\right),
\end{aligned} \tag{19}$$

where $\mathbf{x}^\top = (\mathbf{x}_{\mathsf{pub}}^\top, \mathbf{x}_{\mathsf{pri}}^\top) \in \{0,1\}^{\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}}$ and $(\{\mathsf{lab}_{i,k,b}\}_{k\in[\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}],b\in\{0,1\}}, \widetilde{C}_i) \leftarrow \mathsf{bGC.Garble}(1^\lambda, C_i; R_{i,\mathsf{bgc}})$ for all $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$.

G_0**:** This is the L.H.S. distribution of Equation (19).

G_1**:** This is the same as G_0 except that the garbling algorithm is replaced with the simulator, i.e., the adversary's view is

$$\left(\left\{(\{\mathsf{lab}_{i,k}\}_{k\in[\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}, \widetilde{C}_i)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \mathsf{aux}, \mathbf{x}_{\mathsf{pub}}, \{C_i\}_{i\in[L]}\right)$$

where $(\{\mathsf{lab}_{i,k}\}_{k\in[\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}, \widetilde{C}_i) \leftarrow \mathsf{bGC.Sim}(1^\lambda, 1^{\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}}, \delta_i^* = C_i(\mathbf{x}))$ for all $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$. We have $\mathsf{G}_0 \approx_c \mathsf{G}_1$ from the simulation security of bGC (Definition 3.14).

G_2**:** This is the same as G_1 except that bGC.Sim is run on random inputs, i.e.,

$$\left(\{\mathsf{lab}_{i,k}\}_{k\in[\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}, \widetilde{C}_i\right) \leftarrow \mathsf{bGC.Sim}\left(1^\lambda, 1^{\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}}, \delta_i^{\$}\right),$$

where $\delta_i^{\$} \xleftarrow{\$} \{0,1\}$ for all $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$. We have $\mathsf{G}_1 \approx_c \mathsf{G}_2$ which is implied by Equation (16) and the fact that adding independent terms does not make the task of distinguishing the two distributions any easier.

G_3**:** This is the R.H.S. distribution of Equation (19). We have $\mathsf{G}_2 \approx_c \mathsf{G}_3$ which follows from the blindness of bGC (Definition 3.15).

□

4.8 Construction for TMs with Bounded-Length Private Inputs

In this section, we construct a Sel-prCT secure sRFE scheme for the class $\mathcal{T}_{\ell_{\mathsf{pri}}}$ of Turing machines with public input space $\mathcal{X}_{\mathsf{pub}} = \{0,1\}^*$ and private input space $\mathcal{X}_{\mathsf{pri}} = \{0,1\}^{\ell_{\mathsf{pri}}}$.

Construction 4.11 (Two-Stage Sel-prCT Secure sRFE for TMs)**.** The construction uses the following building blocks:

- A blind garbling scheme $\mathsf{bGC} = \mathsf{bGC}.(\mathsf{Garble}, \mathsf{Eval}, \mathsf{Sim})$ with decomposability (Definition 3.16). We assume that the labels and the random coins used by $\{\mathsf{bGC.Garble}_k\}_k$ and $\{\mathsf{bGC.Garble}_{\mathsf{inp},k}\}_k$ are in $\{0,1\}^\lambda$. The former is guaranteed by Definition 3.12 and the latter can be achieved without loss of generality by using a PRF to derive longer (pseudo-)random coins if needed. We can instantiate bGC with the required properties assuming one-way functions (Fact 3.17).
- A pseudorandom function $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ with key space, input space and output space being $\{0,1\}^\lambda$. PRF can be instantiated assuming one-way functions.
- A laconic pPRIO scheme $\mathsf{LprIO} = \mathsf{LprIO}.(\mathsf{Setup}, \mathsf{Digest}, \mathsf{ObfOff}, \mathsf{ObfOn}, \mathsf{Eval})$ with global setup and a deterministic LprIO.Digest algorithm. Without loss of generality, the state output by LprIO.ObfOff and the random coins of LprIO.ObfOn are in $\{0,1\}^\lambda$. To achieve the former, we let the state be the random coins used by LprIO.ObfOff which can in turn be replaced with a string in $\{0,1\}^\lambda$ using a PRF. For the latter, we can also use a PRF. We note that Definition 3.43 guarantees that the length of digests is bounded by a fixed polynomial $\ell_{\mathsf{dig}} = \ell_{\mathsf{dig}}(\lambda)$ in the security parameter. We can instantiate LprIO with the desired properties assuming prFE and LWE (Construction B.1, Theorem B.4).
- A Sel-prCT secure sRFE scheme $\mathsf{sRFE} = \mathsf{sRFE}.(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$ for the class $\mathcal{C}_{\ell'_{\mathsf{pri}},\ell'_{\mathsf{dep}},\ell'_{\mathsf{out}}}$ consisting of circuits with public input length $\ell'_{\mathsf{pub}} = 0$, private input length $\ell'_{\mathsf{pri}} = \ell_{\mathsf{pri}} + \ell_{\mathsf{dig}} + 9\lambda$, maximum depth ℓ'_{dep} and output length ℓ'_{out}, where we set ℓ'_{dep} and ℓ'_{out} so that the circuit class contains $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]$ defined in Figure 9. We denote the information specifying the circuit class by $\mathsf{param}' = (1^{\ell'_{\mathsf{pri}}}, 1^{\ell'_{\mathsf{dep}}}, 1^{\ell'_{\mathsf{out}}})$. We can construct sRFE with the desired properties assuming prFE and LWE (Construction 4.4).

The details of the Sel-prCT secure sRFE scheme for the class $\mathcal{T}_{\ell_{\mathsf{pri}}}$ are as follows.

$\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{pri}}})$**:** Run

$$\mathsf{LprIO.crs} \leftarrow \mathsf{LprIO.Setup}(1^\lambda)\,, \qquad \mathsf{sRFE.crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda, \mathsf{param}')\,,$$

and output $\mathsf{crs} := (\mathsf{LprIO.crs}, \mathsf{sRFE.crs})$.

$\mathsf{Gen}(\mathsf{crs})$**:** Parse $\mathsf{crs} = (\mathsf{LprIO.crs}, \mathsf{sRFE.crs})$, run

$$\mathsf{sRFE.}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{sRFE.Gen}(\mathsf{sRFE.crs})\,,$$

and output the key pair $(\mathsf{pk} := \mathsf{sRFE.pk}, \mathsf{sk} := \mathsf{sRFE.sk})$.

$\mathsf{Agg}(\mathsf{crs}, \{\mathsf{pk}_i, M_i\}_{i\in[L]})$**:** Parse $\mathsf{crs} = (\mathsf{LprIO.crs}, \mathsf{sRFE.crs})$ and $\{\mathsf{pk}_i = \mathsf{sRFE.pk}_i\}_{i\in[L]}$. Run

$$\begin{aligned}\left\{\mathsf{dig}_{M_i} \leftarrow \mathsf{LprIO.Digest}\left(\{(k, C^{(k)}_{\mathsf{tm},i})\}_{k\in[|C_{\mathsf{tm},i}|]}\right)\right\}_{i\in[L]}\\ \left(\mathsf{sRFE.mpk}, \{\mathsf{sRFE.hsk}_i\}_{i\in[L]}\right) \leftarrow \mathsf{sRFE.Agg}\left(\mathsf{sRFE.crs}, \{(\mathsf{sRFE.pk}_i, C_{\mathsf{reg},i})\}_{i\in[L]}\right)\,,\end{aligned}$$

where $C_{\mathsf{tm},i} = C_{\mathsf{tm}}[\mathsf{LprIO.crs}, M_i]$ and $C_{\mathsf{reg},i} = C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]$ are defined in Figure 7 and Figure 9, respectively. Output $\mathsf{mpk} := (\mathsf{LprIO.crs}, \mathsf{sRFE.mpk})$ and $\mathsf{hsk}_i := (\mathsf{LprIO.crs}, \mathsf{sRFE.hsk}_i)$ for all $i \in [L]$.

Input:
- two (fixed-length) bit strings $\mathsf{bits}(\ell), \mathsf{bits}(t) \in \{0,1\}^\lambda$
- a LprIO state LprIO.st
- random coins $R_{\mathsf{dig}}, R_{\mathsf{bgc}}, R_{\mathsf{cir}} \in \{0,1\}^\lambda$

Output: an LprIO online obfuscation $\widehat{C}_{\mathsf{on,cir}}$

Hardwired Values: a Turing machine M

- Run

$$\begin{aligned}\mathsf{dig}_T &\leftarrow \mathsf{LprIO.Digest}(\mathsf{LprIO.crs}, \{(k, T^{(k)})\}_{k\in[|T|]}; R_{\mathsf{dig}})\\ \widehat{C}_{\mathsf{on,cir}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_T, E_{\mathsf{cir}}[R_{\mathsf{bgc}}]; R_{\mathsf{cir}})\,,\end{aligned}$$

  where $T = T_\ell[M, t]$ and E_{cir} are defined in Figure 8 and 5, respectively.
- Output $\widehat{C}_{\mathsf{on,cir}}$.

Figure 7: Definition of the circuit $C_{\mathsf{tm}}[\mathsf{LprIO.crs}, M]$

Input: a vector $\mathbf{x} \in \{0,1\}^\ell$

Output: a bit $b \in \{0,1\}$

Hardwired Values: a Turing machine M and a runtime $t = \mathsf{poly}(\lambda)$

- Run M on input $\mathbf{x}$ for t steps.
- Output 1 if M is in an accepting state and 0 otherwise.

Figure 8: Definition of the circuit $T_\ell[M, t]\colon \{0,1\}^\ell \to \{0,1\}$

$\mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}})$**:** The encryption algorithm proceeds in two steps.

Input:
- a bit string $\mathsf{bits}(t) \in \{0,1\}^\lambda$
- a LprIO digest $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}$; we implicitly assume that $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}$ contains the length $\ell_{\mathsf{pub}} := |\mathbf{x}_{\mathsf{pub}}|$
- a private input vector $\mathbf{x}_{\mathsf{pri}} \in \{0,1\}^{\ell_{\mathsf{pri}}}$
- two LprIO states LprIO.st and LprIO.st$'$
- PRF seeds $\mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}} \in \{0,1\}^\lambda$

Output:
- two sets of labels $\{\mathsf{lab}'_{k,\mathbf{y}[k]}\}_{k\in[6\lambda]}$ and $\{\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}$
- LprIO online obfuscations $\widehat{C}'_{\mathsf{on},\mathsf{cir}}$ and $\widehat{C}_{\mathsf{on},\mathsf{pub}}$

Hardwired Values:
- a slot index $i \in [L]$
- a CRS LprIO.crs
- a LprIO digest dig_{M_i}

- Compute $R_{\mathsf{str}} = \mathsf{PRF}(\mathsf{sd}_{\mathsf{str}}, i)$ and $R'_{\mathsf{str}'} = \mathsf{PRF}(\mathsf{sd}'_{\mathsf{str}'}, i)$ for $\mathsf{str} \in \{\mathsf{dig},\mathsf{pub},\mathsf{cir},\mathsf{bgc}\}$ and $\mathsf{str}' \in \{\mathsf{cir},\mathsf{bgc}\}$.
- Define $\mathbf{y} = \big(\mathsf{bits}(\ell) \in \{0,1\}^\lambda, \mathsf{bits}(t), \mathsf{LprIO.st}, R_{\mathsf{dig}}, R_{\mathsf{bgc}}, R_{\mathsf{cir}}\big) \in \{0,1\}^{6\lambda}$, where $\ell = \ell_{\mathsf{pub}} + \ell_{\mathsf{pri}}$.
- Run

$$\begin{aligned}(\mathsf{lab}'_{k,0}, \mathsf{lab}'_{k,1}) &\leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R'_{\mathsf{bgc}}) \qquad \text{for } k \in [6\lambda]\\ (\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) &\leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R_{\mathsf{bgc}}) \qquad \text{for } k \in [\ell_{\mathsf{pub}}+1;\ell]\,.\end{aligned}$$

- Run

$$\begin{aligned}\widehat{C}'_{\mathsf{on},\mathsf{cir}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}', \mathsf{dig}_{M_i}, E_{\mathsf{cir}}[R'_{\mathsf{bgc}}]; R'_{\mathsf{cir}})\\ \widehat{C}_{\mathsf{on},\mathsf{pub}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, E_{\mathsf{pub}}[R_{\mathsf{bgc}}]; R_{\mathsf{pub}})\,,\end{aligned}$$

  where E_{cir} and E_{pub} are defined in Figure 5 and 6.
- Output $(\{\mathsf{lab}'_{k,\mathbf{y}[k]}\}_{k\in[6\lambda]}, \{\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}, \widehat{C}'_{\mathsf{on},\mathsf{cir}}, \widehat{C}_{\mathsf{on},\mathsf{pub}})$.

Figure 9: Definition of the circuit $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]$

$\mathsf{EncOff}(\mathsf{mpk})$**:** Parse $\mathsf{mpk} = (\mathsf{LprIO.crs}, \mathsf{sRFE.mpk})$ and run

$$\begin{aligned}(\widehat{C}_{\mathsf{off}}, \mathsf{LprIO.st}) &\leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S)\\ (\widehat{C}'_{\mathsf{off}}, \mathsf{LprIO.st}') &\leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S)\,,\end{aligned}$$

where S is the maximum size of the circuits $E_{\mathsf{cir}}[R_{\mathsf{bgc}}]$ and $E_{\mathsf{pub}}[R_{\mathsf{bgc}}]$ defined in Figure 5 and 6. Output $\mathsf{ct}_{\mathsf{off}} := (\widehat{C}_{\mathsf{off}}, \widehat{C}'_{\mathsf{off}})$ and $\mathsf{st} = (\mathsf{LprIO.st}, \mathsf{LprIO.st}', \mathsf{LprIO.crs}, \mathsf{sRFE.mpk})$.

$\mathsf{EncOn}(\mathsf{st}, x_{\mathsf{pub}}, x_{\mathsf{pri}})$**:** Parse $\mathsf{st} = (\mathsf{LprIO.st}, \mathsf{LprIO.crs}, \mathsf{sRFE.mpk})$, $x_{\mathsf{pub}} = (1^t, \mathbf{x}_{\mathsf{pub}}) \in \{0,1\}^*$ and $x_{\mathsf{pri}} = \mathbf{x}_{\mathsf{pri}} \in \{0,1\}^{\ell_{\mathsf{pri}}}$. Sample PRF seeds $\mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}} \xleftarrow{\$} \{0,1\}^\lambda$ and run

$$\begin{aligned}\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} &\leftarrow \mathsf{LprIO.Digest}\big(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}\big)\\ \mathsf{sRFE.ct} &\leftarrow \mathsf{sRFE.Enc}(\mathsf{sRFE.mpk}, \mathbf{x}'_{\mathsf{pri}})\,,\end{aligned}$$

where

$$\mathbf{x}'_{\mathsf{pri}} = \big(\mathsf{bits}(t), \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{LprIO.st}', \mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}}\big) \in \{0,1\}^{\ell'_{\mathsf{pri}}}\,.$$

Output $\mathsf{ct}_{\mathsf{on}} := \mathsf{sRFE.ct}$.

The final output of $\mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}})$ is $\mathsf{ct} = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}})$.

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, M_{i^*}, \mathsf{ct}, \mathbf{x}_{\mathsf{pub}})$**:** Parse $\mathsf{sk}_{i^*} = \mathsf{sRFE.sk}_{i^*}$, $\mathsf{hsk}_i^* = (\mathsf{LprIO.crs}, \mathsf{sRFE.hsk}_{i^*})$, $\mathsf{ct} = (\widehat{C}'_{\mathsf{off}}, \widehat{C}_{\mathsf{off}}, \mathsf{sRFE.ct})$ and $x_{\mathsf{pub}} = (1^t, \mathbf{x}_{\mathsf{pub}})$. Let $\ell_{\mathsf{pub}} = |\mathbf{x}_{\mathsf{pub}}|$ and $\ell = \ell_{\mathsf{pub}} + \ell_{\mathsf{pri}}$. Run

$$\begin{aligned}
\mathsf{dig}_{M_{i^*}} &\leftarrow \mathsf{LprIO.Digest}\big(\mathsf{LprIO.crs}, \{(k, C^{(k)}_{\mathsf{tm},i^*})\}_{k\in[|C_{\mathsf{tm},i^*}|]}\big)\\
\big(\{\mathsf{lab}'_k\}_{k\in[6\lambda]}, \{\mathsf{lab}_k\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}, \widehat{C}'_{\mathsf{on,cir}}, \widehat{C}_{\mathsf{on,pub}}\big) &\leftarrow \mathsf{sRFE.Dec}\big(\mathsf{sRFE.sk}_{i^*}, \mathsf{sRFE.hsk}_{i^*}, C_{\mathsf{reg},i^*}, \mathsf{sRFE.ct}\big)\,,
\end{aligned}$$

where $C_{\mathsf{tm},i^*} = C_{\mathsf{tm}}[\mathsf{LprIO.crs}, M_{i^*}]$ and $C_{\mathsf{reg},i^*} = C_{\mathsf{reg}}[i^*, \mathsf{LprIO.crs}, \mathsf{dig}_{M_{i^*}}]$ are defined in Figures 7 and 9. Compute

$$\begin{aligned}
\widetilde{C}_{\mathsf{tm},i^*} = \{\widetilde{C}^{(k)}_{\mathsf{tm},i^*}\}_{k\in[|C_{\mathsf{tm},i^*}|]} &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, C^{(k)}_{\mathsf{tm},i^*})\}_{k\in[|C_{\mathsf{tm},i^*}|]}, \widehat{C}'_{\mathsf{cir}} = (\widehat{C}'_{\mathsf{off}}, \widehat{C}'_{\mathsf{on,cir}})\big)\\
\widehat{C}_{\mathsf{on,cir}} &\leftarrow \mathsf{bGC.Eval}\big(\widetilde{C}_{\mathsf{tm},i^*}, \{\mathsf{lab}'_k\}_{k\in[6\lambda]}\big)\\
\widetilde{T}_{i^*} = \{\widetilde{T}^{(k)}_{i^*}\}_{k\in[|T_{i^*}|]} &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, T^{(k)}_{i^*})\}_{k\in[|T_{i^*}|]}, \widehat{C}_{\mathsf{cir}} = (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,cir}})\big)\\
\{\mathsf{lab}_k\}_{k\in[\ell_{\mathsf{pub}}]} &\leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}, \widehat{C}_{\mathsf{pub}} = (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,pub}})\big)\,,
\end{aligned}$$

where the circuit $T_{i^*} = T_\ell[M_{i^*}, t]$ is defined in Figure 8. Output $z \leftarrow \mathsf{bGC.Eval}(\widetilde{T}_{i^*}, \{\mathsf{lab}_k\}_{k\in[\ell]})$.

Proposition 4.12 (Correctness and Compactness)**.** *The sRFE scheme for the class $\mathcal{T}_{\ell_{\mathsf{pri}}}$ in Construction 4.11 is correct and compact. Specifically, it has the following parameters:*

$$\begin{aligned}
|\mathsf{crs}| &= \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda) & |\mathsf{mpk}| &= \log\log L + \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)\\
|\mathsf{hsk}_i| &= \log L \cdot \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda) & |\mathsf{ct}| &= \log L \cdot \mathsf{poly}(\ell_{\mathsf{pri}}, \lambda)\,.
\end{aligned}$$

Proposition 4.13 (Security)**.** *If* bGC *is simulation secure (Definition 3.14) and blind (Definition 3.15),* PRF *is secure,* LprIO *is secure (Definition 3.44) and* sRFE *is* Sel-prCT *secure (Definition 4.2), then the sRFE scheme in Construction 4.11 satisfies two-stage* Sel-prCT *security.*

The proofs of Propositions 4.12 and 4.13 can be found in Sections 4.9 and 4.10, respectively.

4.9 Proof of Correctness and Compactness

Proof of Proposition 4.12. We argue that Construction 4.11 is correct and compact.

Correctness. Let $\lambda, L \in \mathbb{N}$, $i^* \in [L]$, $\{M_i\}_{i\in[L]} \subseteq \mathcal{T}_{\ell_{\mathsf{pri}}}$ and $(x_{\mathsf{pub}}, x_{\mathsf{pri}}) \in \mathcal{X}_{\mathsf{pub}} \times \mathcal{X}_{\mathsf{pri}}$. Then we have

$$\begin{aligned}
\mathsf{crs} := (\mathsf{LprIO.crs}, \mathsf{sRFE.crs}) &\leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param})\\
\big\{(\mathsf{pk}_i := \mathsf{sRFE.pk}_i, \mathsf{sk}_i := \mathsf{sRFE.sk}_i) &\leftarrow \mathsf{Gen}(\mathsf{crs})\big\}_{i\in[L]}\\
\begin{pmatrix} \mathsf{mpk} := (\mathsf{LprIO.crs}, \mathsf{sRFE.mpk}),\\ \big\{\mathsf{hsk}_i := (\mathsf{LprIO.crs}, \mathsf{sRFE.hsk}_i)\big\}_{i\in[L]} \end{pmatrix} &\leftarrow \mathsf{Agg}(\mathsf{crs}, (\mathsf{pk}_i, M_i)_{i\in[L]})\,,
\end{aligned}$$

where Agg registers the public key $\mathsf{sRFE.pk}_i$ with respect to the circuit $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]$ which in turn hardwires the digest $\mathsf{dig}_{M_i} \leftarrow \mathsf{LprIO.Digest}(\mathsf{LprIO.crs}, \{(k, C^{(k)}_{\mathsf{tm},i})\}_{k\in[|C_{\mathsf{tm},i}|]})$ for $C_{\mathsf{tm},i} = C_{\mathsf{tm}}[\mathsf{LprIO.crs}, M_i]$. Subsequently, $\mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}})$ parses $x_{\mathsf{pub}} = (1^t, \mathbf{x}_{\mathsf{pub}})$ and $x_{\mathsf{pri}} = \mathbf{x}_{\mathsf{pri}}$, defines $\ell_{\mathsf{pub}} := |\mathbf{x}_{\mathsf{pub}}|$, samples PRF seeds $\mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}} \xleftarrow{\$} \{0,1\}^\lambda$ and runs

$$\begin{aligned}
(\widehat{C}_{\mathsf{off}}, \mathsf{LprIO.st}) &\leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S)\\
(\widehat{C}'_{\mathsf{off}}, \mathsf{LprIO.st}') &\leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S)\\
\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} &\leftarrow \mathsf{LprIO.Digest}\big(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}\big)\\
\mathsf{sRFE.ct} &\leftarrow \mathsf{sRFE.Enc}\big(\mathsf{sRFE.mpk}, \mathbf{x}'_{\mathsf{pri}}\big)\,,
\end{aligned}$$

where S is the maximum size of the circuits $E_{\mathsf{cir}}[R_{\mathsf{bgc}}]$ and $E_{\mathsf{pub}}[R_{\mathsf{bgc}}]$, and

$$\mathbf{x}'_{\mathsf{pri}} = \big(\mathsf{bits}(t), \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{LprIO.st}', \mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}}\big) \in \{0,1\}^{\ell'_{\mathsf{pri}}} .$$

Decryption starts by running sRFE.Dec. To do so, it first reconstructs $\mathsf{dig}_{M_{i^*}}$ by running

$$\mathsf{dig}_{M_{i^*}} \leftarrow \mathsf{LprIO.Digest}\big(\mathsf{LprIO.crs}, \{(k, C^{(k)}_{\mathsf{tm},i^*})\}_{k\in[|C_{\mathsf{tm},i^*}|]}\big) .$$

Note that the hardwired values $\mathsf{LprIO.crs}$ and M_{i^*} are given as input to the decryption algorithm and that LprIO.Digest is a deterministic algorithm, so the obtained digest is the same as during aggregation. Subsequently, $\mathsf{dig}_{M_{i^*}}$ is used to reconstruct $C_{\mathsf{reg},i^*} = C_{\mathsf{reg}}[i^*, \mathsf{LprIO.crs}, \mathsf{dig}_{M_{i^*}}]$, and we can observe that

$$\begin{aligned}
&\mathsf{sRFE.Dec}\big(\mathsf{sRFE.sk}_{i^*}, \mathsf{sRFE.hsk}_{i^*}, C_{\mathsf{reg},i^*}, \mathsf{sRFE.ct}\big)\\
&= C_{\mathsf{reg},i^*}(\mathsf{bits}(t), \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{LprIO.st}', \mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}})\\
&= \big(\{\mathsf{lab}'_{k,\mathbf{y}[k]}\}_{k\in[6\lambda]}, \{\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}, \widehat{C}'_{\mathsf{on,cir}}, \widehat{C}_{\mathsf{on,pub}}\big) ,
\end{aligned}$$

where

$$\begin{aligned}
\{(\mathsf{lab}'_{k,0}, \mathsf{lab}'_{k,1}) &\leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R'_{\mathsf{bgc}})\}_{k\in[6\lambda]}\\
\{(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) &\leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R_{\mathsf{bgc}})\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}\\
\widehat{C}'_{\mathsf{on,cir}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}', \mathsf{dig}_{M_i}, E_{\mathsf{cir}}[R'_{\mathsf{bgc}}]; R'_{\mathsf{cir}})\\
\widehat{C}_{\mathsf{on,pub}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, E_{\mathsf{pub}}[R_{\mathsf{bgc}}]; R_{\mathsf{pub}}) .
\end{aligned}$$

Next, the decryption algorithm evaluates the LprIO obfuscations. In the first step, we have

$$\widetilde{C}_{\mathsf{tm},i^*} = \{\widetilde{C}^{(k)}_{\mathsf{tm},i^*}\}_{k\in[|C_{\mathsf{tm},i^*}|]} \leftarrow \mathsf{LprIO.Eval}\big(\mathsf{LprIO.crs}, \{(k, C^{(k)}_{\mathsf{tm},i^*})\}_{k\in[|C_{\mathsf{tm},i^*}|]}, \widehat{C}'_{\mathsf{cir}} = (\widehat{C}'_{\mathsf{off}}, \widehat{C}'_{\mathsf{on,cir}})\big) .$$

From the correctness of LprIO, it follows that

$$\widetilde{C}^{(k)}_{\mathsf{tm},i^*} = E_{\mathsf{cir}}[R'_{\mathsf{bgc}}](k, C^{(k)}_{\mathsf{tm},i^*}) \leftarrow \mathsf{bGC.Garble}_k(1^\lambda, C^{(k)}_{\mathsf{tm},i^*}; R'_{\mathsf{bgc}}) .$$

A joint evaluation of $\widetilde{C}_{\mathsf{tm},i^*}$ with $\{\mathsf{lab}'_k\}_{k\in[6\lambda]}$ obtained from the sRFE decryption gives

$$\widehat{C}_{\mathsf{on,cir}} = C_{\mathsf{tm},i^*}(\mathbf{y}) \leftarrow \mathsf{bGC.Eval}\big(\widetilde{C}_{\mathsf{tm},i^*}, \{\mathsf{lab}'_k\}_{k\in[6\lambda]}\big) ,$$

where $\mathbf{y} = (\mathsf{bits}(\ell) \in \{0,1\}^\lambda, \mathsf{bits}(t), \mathsf{LprIO.st}, R_{\mathsf{dig}}, R_{\mathsf{bgc}}, R_{\mathsf{cir}})$. By the definition of C_{tm,i^*}, we have

$$\begin{aligned}
\mathsf{dig}_{T^*_i} &\leftarrow \mathsf{LprIO.Digest}(\mathsf{LprIO.crs}, \{(k, T^{(k)}_{i^*})\}_{k\in[|T_{i^*}|]}; R_{\mathsf{dig}})\\
\widehat{C}_{\mathsf{on,cir}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_{T_{i^*}}, E_{\mathsf{cir}}[R_{\mathsf{bgc}}]; R_{\mathsf{cir}}) ,
\end{aligned}$$

where $T_{i^*} = T_\ell[M_{i^*}, t]$ is a circuit that evaluates the Turing machine M_{i^*} for t steps on input a vector of length $\ell = \ell_{\mathsf{pub}} + \ell_{\mathsf{pri}}$. At this point, we have $(\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,cir}}, \widehat{C}_{\mathsf{on,pub}})$ and the remaining part of decryption is basically the same as in our sRFE scheme for unbounded depth circuits (Construction 4.8). The only difference is that $\widehat{C}_{\mathsf{on,cir}}$ does no longer obfuscate the circuit C_{i^*} with fixed input length but the circuit $T_\ell[M_{i^*}, t]$ which is dynamically adapted to the varying length ℓ_{pub} of $\mathbf{x}_{\mathsf{pub}}$ and the desired number t of evaluation steps of M_{i^*}. Please see the proof of Proposition 4.9 for further details about the last part of the decryption.

Compactness. We start by analyzing the size of the circuit $C_{\mathsf{reg}}[i^*, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]$.

- The evaluations of PRF can be performed by circuits of size $\mathsf{poly}(\lambda)$.
- The $(6\lambda + \ell_{\mathsf{pri}})$ computations of $\mathsf{bGC.Garble}_{\mathsf{inp},k}$ can each be implemented by a circuit of size $\mathsf{poly}(\lambda)$, so the overall computation can be performed in size $\mathsf{poly}(\lambda, \ell_{\mathsf{pri}})$.

- To bound the size of the first execution of $\mathsf{LprIO.ObfOn}$, we observe that the overall input length is a fixed polynomial in λ since $|\mathsf{LprIO.crs}| = \mathsf{poly}(\lambda)$, $|\mathsf{LprIO.st}'| = \lambda$ and $|\mathsf{dig}_{M_i}| = \mathsf{poly}(\lambda)$ (guaranteed by Definition 3.43 and Theorem B.4) as well as $|E_{\mathsf{cir}}[R'_{\mathsf{bgc}}]| = \mathsf{poly}(\lambda)$ (argued in the proof of Proposition 4.9). Since $\mathsf{LprIO.ObfOn}$ is a poly-time algorithm, this implies that the overall size can be bounded by $\mathsf{poly}(\lambda)$.
- The size of a circuit performing the second execution of $\mathsf{LprIO.ObfOn}$ can be bounded similarly. First, we observe that the overall input length of $\mathsf{LprIO.ObfOn}$ is a fixed polynomial in λ. In particular, we recall from the proof of Proposition 4.9 that $|E_{\mathsf{pub}}[R_{\mathsf{bgc}}]| = \mathsf{poly}(\lambda)$. Thus, the size of the circuit computing $\mathsf{LprIO.ObfOn}$ is also $\mathsf{poly}(\lambda)$.

Plugging this size bound into the parameters of our sRFE instantiation (Construction 4.4) will give the parameters as stated. For a detailed analysis, please see the proof of Proposition 4.9. □

4.10 Proof of Security

Proof of Proposition 4.13. To avoid a clash of variables, we will use the following convention throughout this proof. The sRFE scheme for Turing machines which is built in Construction 4.11 is denoted by sRFE. On the other hand, the sRFE scheme for bounded depth circuits which serves as a building block and which was previously denoted by sRFE is now denoted by sRFE'. All variables (e.g., keys, inputs, etc.) belonging either to sRFE or sRFE' will be distinguished in the same way. Consider a PPT sampler Samp in the security game $\mathbf{Exp}^{\mathsf{post\text{-}real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$. We recall this game to fix notations.

- *Setup.* Launch $\mathsf{Samp}(1^\lambda)$ and receive from it the challenge $x_{\mathsf{pub}} = (1^t, \mathbf{x}_{\mathsf{pub}}) \in \{0,1\}^*, x_{\mathsf{pri}} = \mathbf{x}_{\mathsf{pri}} \in \{0,1\}^{\ell_{\mathsf{pri}}}$. Define $\ell_{\mathsf{pub}} = |\mathbf{x}_{\mathsf{pub}}|$ and $\ell = \ell_{\mathsf{pub}} + \ell_{\mathsf{pri}}$ Run

$$\mathsf{LprIO.crs} \leftarrow \mathsf{LprIO.Setup}(1^\lambda)\,, \qquad \mathsf{sRFE}'.\mathsf{crs} \leftarrow \mathsf{sRFE}'.\mathsf{Setup}(1^\lambda, \mathsf{param}')\,,$$

  and send $\mathsf{crs} := (\mathsf{LprIO.crs}, \mathsf{sRFE}'.\mathsf{crs})$ to Samp. Initialize an empty set $\mathcal{C} := \varnothing$, an empty dictionary $\mathcal{D}$ and a transcript $\mathsf{rec}_{\mathsf{post}} := (x_{\mathsf{pub}}, \mathsf{crs})$.
- *Query.* Repeat the following for arbitrarily many rounds determined by Samp. In each round, Samp has two options.
  - $\mathsf{QGen}()$: run $\mathsf{sRFE}'.(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{sRFE}'.\mathsf{Gen}(\mathsf{sRFE}'.\mathsf{crs})$ and return $\mathsf{pk} := \mathsf{sRFE}'.\mathsf{pk}$ to Samp. Set $\mathcal{D}[\mathsf{pk}] = \mathsf{sk} := \mathsf{sRFE}'.\mathsf{sk}$ and $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \,\|\, \mathsf{pk}$.
  - $\mathsf{QCor}(\mathsf{pk})$: upon Samp submitting a public key pk, return $\mathcal{D}[\mathsf{pk}]$ to Samp. Furthermore, add pk to $\mathcal{C}$ and set $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \,\|\, (\mathsf{pk},\mathsf{sk})$.
- *Challenge.* Upon Samp submitting aux and $\{(\mathsf{pk}_i^*, M_i)\}_{i\in[L]}$ aggregate the public keys as follows

$$\begin{aligned}\left\{\mathsf{dig}_{M_i} \leftarrow \mathsf{LprIO.Digest}\left(\{(k, C^{(k)}_{\mathsf{tm},i})\}_{k\in[|C_{\mathsf{tm},i}|]}\right)\right\}_{i\in[L]}\\ \left(\mathsf{sRFE.mpk}, \{\mathsf{sRFE.hsk}_i\}_{i\in[L]}\right) \leftarrow \mathsf{sRFE.Agg}\left(\mathsf{sRFE.crs}, \{(\mathsf{sRFE.pk}_i, C_{\mathsf{reg},i})\}_{i\in[L]}\right)\,,\end{aligned}$$

  where we denote $C_{\mathsf{tm},i} = C_{\mathsf{tm}}[\mathsf{LprIO.crs}, M_i]$ and $C_{\mathsf{reg},i} = C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]$. Then sample PRF seeds $\mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}} \xleftarrow{\$} \{0,1\}^\lambda$ and compute the challenge ciphertext

$$\begin{aligned}(\widehat{C}_{\mathsf{off}}, \mathsf{LprIO.st}) &\leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S)\\ (\widehat{C}'_{\mathsf{off}}, \mathsf{LprIO.st}') &\leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S)\\ \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} &\leftarrow \mathsf{LprIO.Digest}\left(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}\right)\\ \mathsf{sRFE.ct} &\leftarrow \mathsf{sRFE.Enc}\left(\mathsf{sRFE.mpk}, \mathbf{x}'_{\mathsf{pri}}\right)\,,\end{aligned}$$

  where $\mathbf{x}'_{\mathsf{pri}} = (\mathsf{bits}(t), \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{LprIO.st}', \mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}}) \in \{0,1\}^{\ell'_{\mathsf{pri}}}$. Let $\mathsf{ct}^* = (\mathsf{ct}_{\mathsf{off}} := (\widehat{C}_{\mathsf{off}}, \widehat{C}'_{\mathsf{off}}), \mathsf{ct}^*_{\mathsf{on}} := \mathsf{sRFE}'.\mathsf{ct})$ and $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \,\|\, (\mathsf{aux}, \{(\mathsf{pk}_i^*, M_i)\}_{i\in[L]}, \mathsf{ct}^*)$.

- *Guess.* Run the adversary $\mathcal{A}_{\mathsf{post}}$ on input $(1^\lambda, \mathsf{rec}_{\mathsf{post}})$ whose output $b \in \{0,1\}$ is also the outcome of the experiment.

We need to show that if

$$\left(\mathsf{aux}, x_{\mathsf{pub}}, \{M_i\}_{i\in[L]}, \{\delta_i^* := M_i(x_{\mathsf{pub}}, x_{\mathsf{pri}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right) \approx_c \left(\mathsf{aux}, x_{\mathsf{pub}}, \{M_i\}_{i\in[L]}, \{\delta_i^* \xleftarrow{\$} \{0,1\}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right), \tag{20}$$

then $\mathbf{Exp}^{\text{post-real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}} \approx_c \mathbf{Exp}^{\text{post-rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$, where $\mathbf{Exp}^{\text{post-rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$ proceeds in the same fashion as the experiment $\mathbf{Exp}^{\text{post-real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$ recalled above except that it replaces $\mathsf{ct}^* = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}^*_{\mathsf{on}} := \mathsf{sRFE}'.\mathsf{ct})$ with $\mathsf{ct}^{\$} = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}^{\$}_{\mathsf{on}} \xleftarrow{\$} \mathsf{sRFE}'.\mathcal{CT}_{\mathsf{on}})$. Here, $\mathsf{sRFE}'.\mathcal{CT}_{\mathsf{on}}$ denotes the online part of the ciphertext space of sRFE'.

We invoke the Sel-prCT security of sRFE' with respect to a sampler $\mathsf{Samp}_{\mathsf{sRFE}'}(1^\lambda)$ which works as follows.

- *Setup.* Launch $\mathsf{Samp}(1^\lambda)$ and receive from it a challenge message $x_{\mathsf{pub}} = (1^t, \mathbf{x}_{\mathsf{pub}}) \in \{0,1\}^*, x_{\mathsf{pri}} = \mathbf{x}_{\mathsf{pri}} \in \{0,1\}^{\ell_{\mathsf{pri}}}$. Run

$$\begin{aligned}
\mathsf{LprIO.crs} &\leftarrow \mathsf{LprIO.Setup}(1^\lambda)\\
(\widehat{C}_{\mathsf{off}}, \mathsf{LprIO.st}) &\leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S)\\
(\widehat{C}'_{\mathsf{off}}, \mathsf{LprIO.st}') &\leftarrow \mathsf{LprIO.ObfOff}(\mathsf{LprIO.crs}, 1^\lambda, 1^S)\\
\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} &\leftarrow \mathsf{LprIO.Digest}\left(\mathsf{LprIO.crs}, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}\right),
\end{aligned}$$

  sample PRF seeds $\mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}} \xleftarrow{\$} \{0,1\}^\lambda$ and send the challenge message $\mathbf{x}'_{\mathsf{pri}} = (\mathsf{bits}(t), \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{LprIO.st}', \mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}}) \in \{0,1\}^{\ell'_{\mathsf{pri}}}$ to the challenger of sRFE' to obtain $\mathsf{sRFE}'.\mathsf{crs}$ in return. Send $\mathsf{crs} = (\mathsf{LprIO.crs}, \mathsf{sRFE}'.\mathsf{crs})$ to Samp.

- *Query.* Upon receiving a query from Samp, $\mathsf{Samp}_{\mathsf{sRFE}'}$ makes the same query to the corresponding oracle of the sRFE challenger and forwards the response to Samp.

- *Challenge.* Upon Samp outputting aux and $\{(\mathsf{pk}_i^*, M_i)\}_{i\in[L]}$, $\mathsf{Samp}_{\mathsf{sRFE}'}$ runs

$$\left\{\mathsf{dig}_{M_i} \leftarrow \mathsf{LprIO.Digest}\left(\{(k, C^{(k)}_{\mathsf{tm},i})\}_{k\in[|C_{\mathsf{tm},i}|]}\right)\right\}_{i\in[L]}.$$

  Then it outputs $\mathsf{aux}_{\mathsf{sRFE}'} = (\mathsf{aux}, x_{\mathsf{pub}}, \{M_i\}_{i\in[L]}, \widehat{C}_{\mathsf{off}}, \widehat{C}'_{\mathsf{off}})$ and $\{(\mathsf{pk}_i^*, C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}])\}_{i\in[L]}$.

Under the security of sRFE', it suffices to show that

$$\begin{aligned}
&\left(\mathsf{aux}_{\mathsf{sRFE}'}, \{C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]\}_{i\in[L]}, \{\Delta_i^* := C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}](\mathbf{x}'_{\mathsf{pri}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right)\\
&\approx_c \left(\mathsf{aux}_{\mathsf{sRFE}'}, \{C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]\}_{i\in[L]}, \{\Delta_i^* \xleftarrow{\$} \{0,1\}^{\ell'_{\mathsf{out}}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right),
\end{aligned} \tag{21}$$

where we recall that ℓ'_{out} denotes the output length of the circuit $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]$. Unfolding the definition of $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]$, we can note that

$$C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}](\mathbf{x}'_{\mathsf{pri}}) = (\{\mathsf{lab}'_{i,k,\mathbf{y}_i[k]}\}_{k\in[6\lambda]}, \{\mathsf{lab}_{i,k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}, \widehat{C}'_{i,\mathsf{on},\mathsf{cir}}, \widehat{C}_{i,\mathsf{on},\mathsf{pub}}),$$

where $R_{i,\mathsf{str}} = \mathsf{PRF}(\mathsf{sd}_{\mathsf{str}}, i)$ and $R'_{i,\mathsf{str}'} = \mathsf{PRF}(\mathsf{sd}'_{\mathsf{str}'}, i)$ for $i \in [L]$, $\mathsf{str} \in \{\mathsf{dig}, \mathsf{pub}, \mathsf{cir}, \mathsf{bgc}\}$ and $\mathsf{str}' \in \{\mathsf{cir}, \mathsf{bgc}\}$; $\mathbf{y}_i = \left(\mathsf{bits}(\ell) \in \{0,1\}^\lambda, \mathsf{bits}(t), \mathsf{LprIO.st}, R_{i,\mathsf{dig}}, R_{i,\mathsf{bgc}}, R_{i,\mathsf{cir}}\right)$; and

$$\begin{aligned}
\left\{(\mathsf{lab}'_{i,k,0}, \mathsf{lab}'_{i,k,1}) \leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R'_{i,\mathsf{bgc}})\right\}_{k\in[6\lambda]}\\
\left\{(\mathsf{lab}_{i,k,0}, \mathsf{lab}_{i,k,1}) \leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R_{i,\mathsf{bgc}})\right\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}\\
\widehat{C}'_{i,\mathsf{on},\mathsf{cir}} \leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}', \mathsf{dig}_{M_i}, E_{\mathsf{cir}}[R'_{i,\mathsf{bgc}}]; R'_{i,\mathsf{cir}})\\
\widehat{C}_{i,\mathsf{on},\mathsf{pub}} \leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, E_{\mathsf{pub}}[R_{i,\mathsf{bgc}}]; R_{i,\mathsf{pub}}).
\end{aligned}$$

As neither of the PRF seeds appear anywhere else in the scheme, we can rely on the security of PRF to replace $\{R_{i,\mathsf{str}}\}_{i\in[L],\mathsf{str}\in\{\mathsf{dig},\mathsf{pub},\mathsf{cir},\mathsf{bgc}\}}$ and $\{R'_{i,\mathsf{str}'}\}_{i\in[L],\mathsf{str}'\in\{\mathsf{cir},\mathsf{bgc}\}}$ by truly random strings. Hence, it suffices to prove a

variant of Equation (21), where all $R_{i,\mathsf{str}}$ and $R'_{i,\mathsf{str}'}$ are replaced by strings chosen independently and uniformly at random over $\{0,1\}^\lambda$. To show this variant of Equation (21) with truly random coins, we rely on the security of LprIO. In the first step, we deal with online obfuscations $\{\widetilde{C}'_{i,\epsilon,\mathsf{cir}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}$. For this, we consider a sampler $\mathsf{Samp}'_{\mathsf{LprIO}}(\mathsf{LprIO.crs})$ which does the following.

- *Setup.* On input LprIO.crs, $\mathsf{Samp}'_{\mathsf{LprIO}}$ launches $\mathsf{Samp}(1^\lambda)$ to receive challenge messages $x_{\mathsf{pub}} = (1^t, \mathbf{x}_{\mathsf{pub}})$, $x_{\mathsf{pri}} = \mathbf{x}_{\mathsf{pri}}$ in return. Run

$$\mathsf{sRFE}'.\mathsf{crs} \leftarrow \mathsf{sRFE}'.\mathsf{Setup}(1^\lambda, \mathsf{param}')\,,$$

  and send $\mathsf{crs} = (\mathsf{LprIO.crs}, \mathsf{sRFE}'.\mathsf{crs})$ to Samp.

- *Query.* Upon receiving a QGen or QCor query from Samp, $\mathsf{Samp}'_{\mathsf{LprIO}}$ simulates the oracles by running the real $\mathsf{sRFE}'.\mathsf{Gen}$ algorithm.

- *Challenge.* Upon Samp outputting aux and $\{(\mathsf{pk}^*_i, M_i)\}_{i\in[L]}$, $\mathsf{Samp}'_{\mathsf{LprIO}}$ outputs

$$\left(\mathsf{aux}'_{\mathsf{LprIO}} = (\mathsf{aux}'_{\mathsf{LprIO},1}, \mathsf{aux}'_{\mathsf{LprIO},2}), 1^S, \{X'_i\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \{E'_i\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right),$$

  where

$$\begin{aligned}
\mathsf{aux}'_{\mathsf{LprIO},1} &= \left\{\left(\{\mathsf{lab}'_{i,k,\mathbf{y}_i[k]}\}_{k\in[6\lambda]}, \{\mathsf{lab}_{i,k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}, \widehat{C}_{i,\mathsf{on},\mathsf{pub}}\right)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\\
\mathsf{aux}'_{\mathsf{LprIO},2} &= \left(\mathsf{aux}, x_{\mathsf{pub}}, \{M_i\}_{i\in[L]}, \mathsf{LprIO.crs}, \widehat{C}_{\mathsf{off}}\right)\\
S &= \max\left\{|E_{\mathsf{cir}}[R'_{i,\mathsf{bgc}}]|\right\}_{i\in[L]}\\
X'_i &= \{X'_{i,k} = (k, C^{(k)}_{\mathsf{tm},i})\}_{k\in[|C_{\mathsf{tm},i}|]}\\
E'_i &= E_{\mathsf{cir}}[R'_{i,\mathsf{bgc}}]\,.
\end{aligned}$$

Let $\ell_{i,\mathsf{on},\mathsf{pub}} := |\widehat{C}_{i,\mathsf{on},\mathsf{pub}}|$ for $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$ and let ℓ_{cir} denote the output length of $E_{\mathsf{cir}}[R]$ for any $R \in \{0,1\}^\lambda$ (note that the concrete choice of R is irrelevant for the output length). Then we consider the following equation

$$\begin{aligned}
&\left(\mathsf{aux}'_{\mathsf{LprIO},1}, \mathsf{aux}'_{\mathsf{LprIO},2}, 1^S, \{X'_i\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \{E'_i(X'_{i,k})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}},k\in[|C_{\mathsf{tm},i}|]}\right)\\
&\approx_c \left(\begin{array}{c}\left\{\left(\{\Gamma'_{i,k,\mathsf{pri}} \xleftarrow{\$} \{0,1\}^\lambda\}_{k\in[6\lambda]}, \{\Gamma_{i,k,\mathsf{pri}} \xleftarrow{\$} \{0,1\}^\lambda\}_{k\in[\ell_{\mathsf{pri}}]}, O_{i,\mathsf{pub}} \xleftarrow{\$} \{0,1\}^{\ell_{i,\mathsf{on},\mathsf{pub}}}\right)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}},\\ \mathsf{aux}_{\mathsf{LprIO},2}, 1^S, \{X'_i\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \{\Gamma'_{i,k,\mathsf{cir}} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{cir}}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}},k\in[|C_{\mathsf{tm},i}|]}\end{array}\right),
\end{aligned} \tag{22}$$

where we note that the part in the gray box has the same distribution as $\Gamma'_{\mathsf{aux}} \xleftarrow{\$} \{0,1\}^{\ell'_{\mathsf{aux}}}$ for $\ell'_{\mathsf{aux}} := |\mathsf{aux}'_{\mathsf{LprIO},1}|$. We can observe that Equation (22) implies Equation (21) under the security of LprIO with respect to $\mathsf{Samp}'_{\mathsf{LprIO}}$ and Lemma 3.46, so it suffices to prove Equation (22). Unrolling the definitions and applying the decomposability property of bGC (Definition 3.16), we observe that Equation (22) is equivalent to the following

$$\begin{aligned}
&\left(\left\{\left(\{\mathsf{lab}_{i,k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}, \widehat{C}_{i,\mathsf{on},\mathsf{pub}}\right)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \mathsf{aux}'_{\mathsf{LprIO},2}, \boxed{\left\{\left(\{\mathsf{lab}'_{i,k,\mathbf{y}_i[k]}\}_{k\in[6\lambda]}, \widetilde{C}_{\mathsf{tm},i}\right)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}}\right)\\
&\approx_c \left(\left\{\left(\{\Gamma_{i,k,\mathsf{pri}}\}_{k\in[\ell_{\mathsf{pri}}]}, O_{i,\mathsf{pub}}\right)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}, \mathsf{aux}'_{\mathsf{LprIO},2}, \left\{\left(\{\Gamma'_{i,k,\mathsf{pri}}\}_{k\in[6\lambda]}, \{\Gamma'_{i,k,\mathsf{cir}}\}_{k\in[|C_{\mathsf{tm},i}|]}\right)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right),
\end{aligned} \tag{23}$$

where $\boxed{\left(\{\mathsf{lab}'_{i,k,b}\}_{k\in[6\lambda],b\in\{0,1\}}, \widetilde{C}_{\mathsf{tm},i}\right)} \leftarrow \mathsf{bGC.Garble}(1^\lambda, C_{\mathsf{tm},i}; R'_{i,\mathsf{bgc}})$ for all $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$. We note that under the simulation security of bGC (Definition 3.14), it suffices to prove a variant of Equation 23, where the output of the real garbling algorithm is replaced with the output of the simulator $\boxed{\left(\{\mathsf{lab}'_{i,k}\}_{k\in[6\lambda]}, \widetilde{C}_{\mathsf{tm},i}\right)} \leftarrow \mathsf{bGC.Sim}(1^\lambda, 1^{6\lambda}, C_{\mathsf{tm},i}(\mathbf{y}_i))$ for all $i \in [L]$. Unrolling the definition of $C_{\mathsf{tm},i}$, we have that $C_{\mathsf{tm},i}(\mathbf{y}_i) = \widehat{C}_{i,\mathsf{on},\mathsf{cir}}$, where $T_i = T_\ell[M_i, t]$ and

$$\begin{aligned}
\mathsf{dig}_{T_i} &\leftarrow \mathsf{LprIO.Digest}(\mathsf{LprIO.crs}, \{(k, T^{(k)}_i)\}_{k\in[|T_i|]}; R_{i,\mathsf{dig}})\\
\widehat{C}_{i,\mathsf{on},\mathsf{cir}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.crs}, \mathsf{LprIO.st}, \mathsf{dig}_{T_i}, E_{\mathsf{cir}}[R_{i,\mathsf{bgc}}]; R_{i,\mathsf{cir}})\,.
\end{aligned}$$

Using the blindness of bGC (Definition 3.15), we observe that in order to show Equation 23, it is sufficient to prove the following

$$\begin{aligned}&\left(\left\{\left(\{\mathsf{lab}_{i,k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]},\widehat{C}_{i,\mathsf{on},\mathsf{pub}}\right)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}},\mathsf{aux}'_{\mathsf{LprIO},2},\{\widehat{C}_{i,\mathsf{on},\mathsf{cir}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right)\\&\approx_c\left(\left\{\left(\{\Gamma_{i,k,\mathsf{pri}}\}_{k\in[\ell_{\mathsf{pri}}]},O_{i,\mathsf{pub}}\right)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}},\mathsf{aux}'_{\mathsf{LprIO},2},\{O_{i,\mathsf{cir}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right),\end{aligned}\tag{24}$$

where $O_{i,\mathsf{cir}}\xleftarrow{\$}\{0,1\}^{\ell_{i,\mathsf{on},\mathsf{cir}}}$ for $\ell_{i,\mathsf{on},\mathsf{cir}}:=|\widehat{C}_{i,\mathsf{on},\mathsf{cir}}|$. For this, we invoke again the security of LprIO with respect to the following sampler $\mathsf{Samp}_{\mathsf{LprIO}}(\mathsf{LprIO.crs})$.

- *Setup.* On input LprIO.crs, $\mathsf{Samp}_{\mathsf{LprIO}}$ launches $\mathsf{Samp}(1^\lambda)$ to receive challenge messages $x_{\mathsf{pub}}=(1^t,\mathbf{x}_{\mathsf{pub}})$, $x_{\mathsf{pri}}=\mathbf{x}_{\mathsf{pri}}$ in return. Run

$$\mathsf{sRFE}'.\mathsf{crs}\leftarrow\mathsf{sRFE}'.\mathsf{Setup}(1^\lambda,\mathsf{param}')\,,$$

  and send $\mathsf{crs}=(\mathsf{LprIO.crs},\mathsf{sRFE}'.\mathsf{crs})$ to Samp.

- *Query.* Upon receiving a QGen or QCor query from Samp, $\mathsf{Samp}_{\mathsf{LprIO}}$ simulates the oracles by running the real $\mathsf{sRFE}'.\mathsf{Gen}$ algorithm.

- *Challenge.* Upon Samp outputting aux and $\{(\mathsf{pk}_i^*,M_i)\}_{i\in[L]}$, $\mathsf{Samp}_{\mathsf{LprIO}}$ outputs

$$\left(\mathsf{aux}_{\mathsf{LprIO}}=(\mathsf{aux}_{\mathsf{LprIO},1},\mathsf{aux}_{\mathsf{LprIO},2}),1^S,\{X_{i,\mathsf{cir}},X_{i,\mathsf{pub}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}},\{E_{i,\mathsf{cir}},E_{i,\mathsf{pub}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right),$$

  where $S=\max\left\{|E_{\mathsf{cir}}[R_{i,\mathsf{bgc}}]|,|E_{\mathsf{pub}}[R_{i,\mathsf{bgc}}]|\right\}_{i\in[L]}$ and

$$\begin{aligned}\mathsf{aux}_{\mathsf{LprIO},1}&=\{\mathsf{lab}_{i,k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}},k\in[\ell_{\mathsf{pub}}+1;\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}\,,&\quad\mathsf{aux}_{\mathsf{LprIO},2}&=\left(\mathsf{aux},\mathbf{x}_{\mathsf{pub}},\{M_i\}_{i\in[L]}\right),\\X_{i,\mathsf{cir}}&=\{X_{i,k,\mathsf{cir}}=(k,T_i^{(k)})\}_{k\in[|T_i|]}\,,&\quad E_{i,\mathsf{cir}}&=E_{\mathsf{cir}}[R_{i,\mathsf{bgc}}]\,,\\X_{i,\mathsf{pub}}&=\{X_{i,k,\mathsf{pub}}=(k,x_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}\,,&\quad E_{i,\mathsf{pub}}&=E_{\mathsf{pub}}[R_{i,\mathsf{bgc}}]\,.\end{aligned}$$

Applying Lemma 3.46 and the security of LprIO with respect to $\mathsf{Samp}_{\mathsf{LprIO}}$, we observe that Equation (24) is implied by the following

$$\begin{aligned}&\left(\begin{array}{c}\mathsf{aux}_{\mathsf{LprIO},1},\mathsf{aux}_{\mathsf{LprIO},2},1^S,\{X_{i,\mathsf{cir}},X_{i,\mathsf{pub}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}},\\\{E_{i,\mathsf{cir}}(X_{i,k,\mathsf{cir}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}},k\in[|T_i|]}\cup\{E_{i,\mathsf{pub}}(X_{i,k,\mathsf{pub}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}},k\in[\ell_{\mathsf{pub}}]}\end{array}\right)\\&\approx_c\left(\begin{array}{c}\{\Gamma_{i,k,\mathsf{pri}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}},k\in[\ell_{\mathsf{pri}}]},\mathsf{aux}_{\mathsf{LprIO},2},1^S,\{X_{i,\mathsf{cir}},X_{i,\mathsf{pub}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}},\\\{\Gamma_{i,k,\mathsf{cir}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}},k\in[|T_i|]}\cup\{\Gamma_{i,k,\mathsf{pub}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}},k\in[\ell_{\mathsf{pub}}]}\end{array}\right),\end{aligned}\tag{25}$$

where $\Gamma_{i,k,\mathsf{cir}}\xleftarrow{\$}\{0,1\}^{\ell_{\mathsf{cir}}}$ for $(i,k)\in(\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}})\times[|T_i|]$ and $\Gamma_{i,k,\mathsf{pub}}\xleftarrow{\$}\{0,1\}^\lambda$ for $(i,k)\in(\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}})\times[\ell_{\mathsf{pub}}]$. Unrolling the definitions and applying the decomposability property of bGC, we observe that Equation (25) is equivalent to the following

$$\begin{aligned}&\left(\left\{\left(\{\mathsf{lab}_{i,k,\mathbf{x}[k]}\}_{k\in[\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]},\widetilde{T}_i\right)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}},\mathsf{aux},x_{\mathsf{pub}},\{M_i\}_{i\in[L]}\right)\\&\approx_c\left(\left\{\left(\{\Gamma_{i,k,\mathsf{pub}}\}_{k\in[\ell_{\mathsf{pub}}]}\cup\{\Gamma_{i,k,\mathsf{pri}}\}_{k\in[\ell_{\mathsf{pri}}]},\{\Gamma_{i,k,\mathsf{cir}}\}_{k\in[|T_i|]}\right)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}},\mathsf{aux},x_{\mathsf{pub}},\{M_i\}_{i\in[L]}\right),\end{aligned}\tag{26}$$

where $\mathbf{x}^\top=(\mathbf{x}_{\mathsf{pub}}^\top,\mathbf{x}_{\mathsf{pri}}^\top)\in\{0,1\}^{\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}}$ and $(\{\mathsf{lab}_{i,k,b}\}_{k\in[\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}],b\in\{0,1\}},\widetilde{T}_i)\leftarrow\mathsf{bGC.Garble}(1^\lambda,T_i;R_{i,\mathsf{bgc}})$ for all $i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}$. To prove Equation 26, we consider the following sequence of hybrids.

G_0**:** This is the L.H.S. distribution of Equation (26).

G_1**:** This is the same as G_0 except that the garbling algorithm is replaced with the simulator, i.e., the adversary's view is

$$\left(\left\{\left(\{\mathsf{lab}_{i,k}\}_{k\in[\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]},\widetilde{T}_i\right)\right\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}},\mathsf{aux},x_{\mathsf{pub}},\{M_i\}_{i\in[L]}\right)$$

where $(\{\mathsf{lab}_{i,k}\}_{k\in[\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]},\widetilde{T}_i)\leftarrow\mathsf{bGC.Sim}(1^\lambda,1^{\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}},\delta_i^*=M_i(x_{\mathsf{pub}},x_{\mathsf{pri}}))$ for all $i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}$. We have $\mathsf{G}_0\approx_c\mathsf{G}_1$ from the simulation security of bGC (Definition 3.14).

G_2**:** This is the same as G_1 except that bGC.Sim is run on random inputs, i.e.,

$$(\{\mathsf{lab}_{i,k}\}_{k\in[\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}]}, \widetilde{T}_i) \leftarrow \mathsf{bGC.Sim}(1^\lambda, 1^{\ell_{\mathsf{pub}}+\ell_{\mathsf{pri}}}, \delta_i^{\$}) ,$$

where $\delta_i^{\$} \xleftarrow{\$} \{0,1\}$ for all $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$. We have $\mathsf{G}_1 \approx_c \mathsf{G}_2$ which is implied by Equation (20).

G_3**:** This is the R.H.S. distribution of Equation (26). We have $\mathsf{G}_2 \approx_c \mathsf{G}_3$ by the blindness of bGC.

□

4.11 Improving Asymptotic Parameters and Achieving Unbounded-Length Private Inputs

The sizes of the crs, the master public key, helper secret keys and ciphertexts in the sRFE schemes for unbounded depth circuits (Construction 4.8) and Turing machines (Construction 4.11) are already independent of public input lengths and function sizes/depths. However, they still depend on the lengths of private inputs. In this section, we provide a simple generic compiler that (1) completely removes this dependency from the sizes of the CRS, the master public key and the helper secret keys, and (2) reduces the ciphertext size to an additive dependency on the private input length. For the case of Turing machines, we also enable private inputs of unbounded length. Moreover, we generalize the previous constructions to functionalities outputting more than one bit. That is, we build sRFE schemes for

- the class $\boxed{\mathcal{C}_{\ell_{\mathsf{pub}},\ell_{\mathsf{pri}},\ell_{\mathsf{out}}}}$ of all (unbounded depth) circuits with public input space $\mathcal{X}_{\mathsf{pub}} = \{0,1\}^{\ell_{\mathsf{pub}}}$, private input space $\mathcal{X}_{\mathsf{pub}} = \{0,1\}^{\ell_{\mathsf{pri}}}$ and output space $\mathcal{Y} = \{0,1\}^{\ell_{\mathsf{out}}}$, and
- the class $\boxed{\mathcal{T}_{\ell_{\mathsf{out}}}}$ of all Turing machine with public/private input space $\mathcal{X}_{\mathsf{pub}} = \mathcal{X}_{\mathsf{pri}} = \{0,1\}^*$ and output space $\mathcal{Y} = \{0,1\}^{\ell_{\mathsf{out}}}$.

Construction 4.14 (Two-Stage Sel-prCT Secure sRFE With Nearly Optimal Parameters)**.** The construction uses the following ingredients:

- An INDr secure SKE scheme $\mathsf{SKE} = \mathsf{SKE}.(\mathsf{Setup},\mathsf{Enc},\mathsf{Dec})$ with message space $\mathsf{SKE}.\mathcal{M} = \{0,1\}^{\ell_{\mathsf{pri}}}$, key space $\mathsf{SKE}.\mathcal{K} = \{0,1\}^\lambda$ and ciphertext space $\mathsf{SKE}.\mathcal{CT} = \{0,1\}^{\ell_{\mathsf{pri}}+\lambda}$. We can construct a SKE scheme with these properties assuming one-way functions.
- A Sel-prCT secure sRFE scheme $\mathsf{sRFE} = \mathsf{sRFE}.(\mathsf{Setup},\mathsf{Gen},\mathsf{Agg},\mathsf{Enc},\mathsf{Dec})$ for the class $\boxed{\mathcal{C}_{\ell'_{\mathsf{pub}},\ell'_{\mathsf{pri}}}}$ $\boxed{\mathcal{T}_{\ell'_{\mathsf{pri}}}}$, where $\ell'_{\mathsf{pub}} = \ell_{\mathsf{pub}} + \ell_{\mathsf{pri}} + \lambda$ and $\ell'_{\mathsf{pri}} = \lambda$. We can construct such sRFE schemes assuming prFE and LWE (see Construction $\boxed{4.8}$ $\boxed{4.11}$).

The details of the sRFE schemes for $\boxed{\mathcal{C}_{\ell_{\mathsf{pub}},\ell_{\mathsf{pri}},\ell_{\mathsf{out}}}}$ $\boxed{\mathcal{T}_{\ell_{\mathsf{out}}}}$ are as follows. Throughout the description, we use the letter F to generically refer to a function of the considered function class, i.e., F denotes either a $\boxed{\text{circuit}}$ or a $\boxed{\text{Turing machine}}$.

$\mathsf{Setup}(1^\lambda, \boxed{1^{\ell_{\mathsf{pub}}}, 1^{\ell_{\mathsf{pri}}}}, 1^{\ell_{\mathsf{out}}})$**:** Run

$$\mathsf{sRFE.crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda, \boxed{1^{\ell'_{\mathsf{pub}}}}, 1^{\ell_{\mathsf{pri}}'}) ,$$

and output $\mathsf{crs} := \mathsf{sRFE.crs}$.

$\mathsf{Gen}(\mathsf{crs})$**:** Parse $\mathsf{crs} = \mathsf{sRFE.crs}$, run

$$\mathsf{sRFE.}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{sRFE.Gen}(\mathsf{sRFE.crs}) ,$$

and output the key pair ($\mathsf{pk} := \mathsf{sRFE.pk}, \mathsf{sk} := \mathsf{sRFE.sk}$).

Public Input: a public input x_{pub} and a SKE ciphertext SKE.ct
Private Input: a SKE secret key SKE.sk
Output: a bit $b \in \{0,1\}$
Hardwired Values: a circuit/Turing machine F

- Compute $x_{\mathsf{pri}} \leftarrow \mathsf{SKE.Dec}(\mathsf{SKE.sk}, \mathsf{SKE.ct})$.
- Output $F(x_{\mathsf{pub}}, x_{\mathsf{pri}})$.

Figure 10: Definition of the circuit/Turing machine $F_{\mathsf{reg}}[F]$

$\mathsf{Agg}(\mathsf{crs}, \{\mathsf{pk}_i, F_i\}_{i\in[L]})$**:** Parse $\mathsf{crs} = \mathsf{sRFE.crs}$, $\{\mathsf{pk}_i = \mathsf{sRFE.pk}_i\}_{i\in[L]}$ and $F_i = (F_{i,1}, \ldots, F_{i,\ell_{\mathsf{out}}})$. Run

$$\big(\mathsf{sRFE.mpk}, \{\mathsf{sRFE.hsk}_i\}_{i\in[L]}\big) \leftarrow \mathsf{sRFE.Agg}\big(\mathsf{sRFE.crs}, \{(\mathsf{sRFE.pk}_i, F_{\mathsf{reg}}[F_{i,j}])\}_{(i,j)\in[L]\times[\ell_{\mathsf{out}}]}\big)\,,$$

where $F_{\mathsf{reg}}[F_{i,j}]$ is defined in Figure 10. Then output $\mathsf{mpk} := \mathsf{sRFE.mpk}$ and $\mathsf{hsk}_i := \{\mathsf{sRFE.hsk}_{i,j}\}_{j\in[\ell_{\mathsf{out}}]}$ for all $i \in [L]$.

$\mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}})$**:** The encryption algorithm proceeds in two steps.

$\mathsf{EncOff}(\mathsf{mpk})$**:** Parse $\mathsf{mpk} = \mathsf{sRFE.mpk}$, run

$$(\mathsf{sRFE.ct}_{\mathsf{off}}, \mathsf{sRFE.st}) \leftarrow \mathsf{sRFE.EncOff}(\mathsf{sRFE.mpk})$$

and output $\mathsf{ct}_{\mathsf{off}} := \mathsf{sRFE.ct}_{\mathsf{off}}$ and $\mathsf{st} = \mathsf{sRFE.st}$.

$\mathsf{EncOn}(\mathsf{st}, x_{\mathsf{pub}}, x_{\mathsf{pri}})$**:** Parse $\mathsf{st} = \mathsf{sRFE.st}$ and $x_{\mathsf{pri}} = \mathbf{x}_{\mathsf{pri}}$, define $\ell_{\mathsf{pri}} := |\mathbf{x}_{\mathsf{pri}}|$ and run

$$\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda, 1^{\ell_{\mathsf{pri}}})\,, \qquad \mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, \mathbf{x}_{\mathsf{pri}})\,.$$

Then set $x'_{\mathsf{pub}} := (x_{\mathsf{pub}}, \mathsf{SKE.ct})$ and $x'_{\mathsf{pri}} = \mathsf{SKE.sk}$, compute

$$\mathsf{sRFE.ct}_{\mathsf{on}} \leftarrow \mathsf{sRFE.EncOn}(\mathsf{sRFE.st}, x'_{\mathsf{pub}}, x'_{\mathsf{pri}})\,,$$

and output $\mathsf{ct}_{\mathsf{on}} = (\mathsf{SKE.ct}, \mathsf{sRFE.ct}_{\mathsf{on}})$.

The final output of $\mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}})$ is $\mathsf{ct} = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}})$.

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, F_{i^*}, \mathsf{ct}, x_{\mathsf{pub}})$**:** Parse $\mathsf{sk}_{i^*} = \mathsf{sRFE.sk}_{i^*}$, $\mathsf{hsk}^*_i = \mathsf{sRFE.hsk}_{i^*}$, $F_{i^*} = (F_{i^*,1}, \ldots, F_{i^*,\ell_{\mathsf{out}}})$ as well as $\mathsf{ct} = (\mathsf{ct}_{\mathsf{off}} = \mathsf{sRFE.ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}} = (\mathsf{SKE.ct}, \mathsf{sRFE.ct}_{\mathsf{on}}))$. Set $\mathsf{sRFE.ct} := (\mathsf{sRFE.ct}_{\mathsf{off}}, \mathsf{sRFE.ct}_{\mathsf{on}})$ and define $x'_{\mathsf{pub}} := (x_{\mathsf{pub}}, \mathsf{SKE.ct})$. Compute

$$\big\{z_j \leftarrow \mathsf{sRFE.Dec}(\mathsf{sRFE.sk}_{i^*}, \mathsf{sRFE.hsk}_{i^*}, F_{\mathsf{reg}}[F_{i,j}], \mathsf{sRFE.ct}, x'_{\mathsf{pub}})\big\}_{j\in[\ell_{\mathsf{out}}]}\,,$$

where $F_{\mathsf{reg}}[F_{i,j}]$ is defined in Figure 10, and output $\mathbf{z} = (z_1, \ldots, z_{\ell_{\mathsf{out}}})$.

Proposition 4.15 (Correctness and Compactness)**.** *The sRFE scheme in Construction 4.14 is correct and compact. Specifically, it has the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log\log L + \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = \log L \cdot \ell_{\mathsf{out}} \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = \log L \cdot \mathsf{poly}(\lambda) + \ell_{\mathsf{pri}}\,.$$

Proposition 4.16 (Security)**.** *If* SKE *is* INDr *secure (Definition 3.25) and* sRFE *is two-stage* Sel-prCT *secure (Definition 4.3), then the sRFE scheme in Construction 4.14 also satisfies two-stage* Sel-prCT *security.*

The proofs of Propositions 4.15 and 4.16 can be found in Sections 4.12 and 4.13, respectively. We summarize the results of Section 4 in the following theorem.

Theorem 4.17. *Assuming LWE and prFE, there exist* Sel-prCT *secure sRFE schemes supporting the function classes* $\mathcal{C}_{\ell_{\mathsf{pub}},\ell_{\mathsf{pri}},\ell_{\mathsf{out}}}$ *and* $\mathcal{T}_{\ell_{\mathsf{out}}}$ *with the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log\log L + \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = \log L \cdot \ell_{\mathsf{out}} \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = \log L \cdot \mathsf{poly}(\lambda) + \ell_{\mathsf{pri}} \,,$$

where ℓ_{pri} *denotes the length of the private message encrypted in* ct. *(Note that this length is not bounded at the time of setup in the case of Turing machines which is the reason why* ℓ_{pri} *does not appear as an index of the function class* $\mathcal{T}_{\ell_{\mathsf{out}}}$*).*

4.12 Proof of Correctness and Compactness

Proof of Proposition 4.15. We argue that Construction 4.14 is correct and compact.

Correctness. Pick $\lambda, L \in \mathbb{N}$, $i^* \in [L]$, $\{F_i\}_{i\in[L]}$ and $(x_{\mathsf{pub}}, x_{\mathsf{pri}}) \in \mathcal{X}_{\mathsf{pub}} \times \mathcal{X}_{\mathsf{pri}}$. Then we have

$$\begin{aligned}
\mathsf{crs} := \mathsf{sRFE.crs} &\leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param}) \\
\left\{(\mathsf{pk}_i := \mathsf{sRFE.pk}_i, \mathsf{sk}_i := \mathsf{sRFE.sk}_i) &\leftarrow \mathsf{Gen}(\mathsf{crs})\right\}_{i\in[L]} \\
\left(\mathsf{mpk} := \mathsf{sRFE.mpk}, \left\{\mathsf{hsk}_i := \{\mathsf{sRFE.hsk}_{i,j}\}_{j\in[\ell_{\mathsf{out}}]}\right\}_{i\in[L]}\right) &\leftarrow \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, F_i)\}_{i\in[L]}) \,,
\end{aligned}$$

where Agg registers the public key $\mathsf{sRFE.pk}_i$ with respect to $F_{\mathsf{reg}}[F_i]$. Encryption first computes

$$(\mathsf{sRFE.ct}_{\mathsf{off}}, \mathsf{sRFE.st}) \leftarrow \mathsf{sRFE.EncOff}(\mathsf{sRFE.mpk}) \,.$$

Then it parses $x_{\mathsf{pri}} = \mathbf{x}_{\mathsf{pri}}$, defines $\ell_{\mathsf{pri}} := |\mathbf{x}_{\mathsf{pri}}|$, generates

$$\begin{aligned}
\mathsf{SKE.sk} &\leftarrow \mathsf{SKE.Setup}(1^\lambda, 1^{\ell_{\mathsf{pri}}}) \\
\mathsf{SKE.ct} &\leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, \mathbf{x}_{\mathsf{pri}}) \\
\mathsf{sRFE.ct}_{\mathsf{on}} &\leftarrow \mathsf{sRFE.EncOn}\big(\mathsf{sRFE.st}, x'_{\mathsf{pub}} = (x_{\mathsf{pub}}, \mathsf{SKE.ct}), x'_{\mathsf{pri}} = \mathsf{SKE.sk}\big)
\end{aligned}$$

and outputs the ciphertext $\mathsf{ct} = (\mathsf{ct}_{\mathsf{off}} = \mathsf{sRFE.ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}} = (\mathsf{SKE.ct}, \mathsf{sRFE.ct}_{\mathsf{on}}))$. Decryption reconstructs x'_{pub}, sets $\mathsf{sRFE.ct} = (\mathsf{sRFE.ct}_{\mathsf{off}}, \mathsf{sRFE.ct}_{\mathsf{on}})$ and runs

$$\left\{z_j \leftarrow \mathsf{sRFE.Dec}(\mathsf{sRFE.sk}_{i^*}, \mathsf{sRFE.hsk}_{i^*}, F_{\mathsf{reg}}[F_{i,j}], \mathsf{sRFE.ct}, x'_{\mathsf{pub}})\right\}_{j\in[\ell_{\mathsf{out}}]} \,.$$

By the definition of $F_{\mathsf{reg}}[F_{i,j}]$, we conclude that $z_j = F_{i^*,j}(x_{\mathsf{pub}}, x_{\mathsf{pri}})$.

Compactness. The parameters follow immediately by plugging the values $\ell'_{\mathsf{pub}} = \ell_{\mathsf{pub}} + \ell_{\mathsf{pri}} + \lambda$ and $\ell'_{\mathsf{pri}} = \lambda$ into the parameters of sRFE (as stated in Propositions 4.9 and 4.12) and the fact that $\mathsf{SKE}.\mathcal{CT} = \{0,1\}^{\ell_{\mathsf{pri}}+\lambda}$. □

4.13 Proof of Security

Proof of Proposition 4.16. To avoid a clash of variables, we will use the following convention throughout this proof. The sRFE scheme which is built in Construction 4.14 is denoted by sRFE. On the other hand, the sRFE scheme which serves as a building block and which was previously denoted by sRFE is now denoted by sRFE′. All variables (e.g., keys, inputs, etc.) belonging either to sRFE or sRFE′ will be distinguished in the same way. Consider a PPT sampler Samp in the security game $\mathbf{Exp}^{\mathsf{post\text{-}real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$. We briefly recall this game.

- *Setup.* Launch $\mathsf{Samp}(1^\lambda)$ and receive from it the challenge $(x_{\mathsf{pub}}, x_{\mathsf{pri}})$. Parse $x_{\mathsf{pri}} = \mathbf{x}_{\mathsf{pri}}$ and define $\ell_{\mathsf{pri}} = |\mathbf{x}_{\mathsf{pri}}|$. Run

$$\mathsf{sRFE.crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda, \boxed{1^{\ell'_{\mathsf{pub}}}}, 1^{\ell_{\mathsf{pri}}'}) ,$$

  and send $\mathsf{crs} := \mathsf{sRFE}'.\mathsf{crs}$ to Samp. Initialize an empty set $\mathcal{C} := \varnothing$, an empty dictionary $\mathcal{D}$ and a transcript $\mathsf{rec}_{\mathsf{post}} := (x_{\mathsf{pub}}, \mathsf{crs})$.

- *Query.* Repeat the following for arbitrarily many rounds determined by Samp. In each round, Samp has two options.

  - $\mathsf{QGen}()$: run $\mathsf{sRFE}'.(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{sRFE}'.\mathsf{Gen}(\mathsf{sRFE}'.\mathsf{crs})$ and return $\mathsf{pk} := \mathsf{sRFE}'.\mathsf{pk}$ to Samp. Set $\mathcal{D}[\mathsf{pk}] = \mathsf{sk} := \mathsf{sRFE}'.\mathsf{sk}$ and $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| \mathsf{pk}$.
  - $\mathsf{QCor}(\mathsf{pk})$: upon Samp submitting a public key pk, return $\mathcal{D}[\mathsf{pk}]$ to Samp. Furthermore, add pk to $\mathcal{C}$ and set $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| (\mathsf{pk},\mathsf{sk})$.

- *Challenge.* Upon Samp submitting aux and $\{(\mathsf{pk}_i^*, F_i = \{F_{i,j}\}_{j\in[\ell_{\mathsf{out}}]})\}_{i\in[L]}$ aggregate the public keys by running

$$\big(\mathsf{sRFE.mpk}, \{\mathsf{sRFE.hsk}_i\}_{i\in[L]}\big) \leftarrow \mathsf{sRFE.Agg}\big(\mathsf{sRFE.crs}, \{(\mathsf{sRFE.pk}_i, F_{\mathsf{reg}}[F_{i,j}])\}_{(i,j)\in[L]\times[\ell_{\mathsf{out}}]}\big) .$$

  Then compute the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}_{\mathsf{off}} = \mathsf{sRFE.ct}_{\mathsf{off}}, \mathsf{ct}^*_{\mathsf{on}} = (\mathsf{SKE.ct}, \mathsf{sRFE.ct}_{\mathsf{on}}))$ as follows

$$\begin{aligned}(\mathsf{sRFE.ct}_{\mathsf{off}}, \mathsf{sRFE.st}) &\leftarrow \mathsf{sRFE.EncOff}(\mathsf{sRFE.mpk})\\ \mathsf{SKE.sk} &\leftarrow \mathsf{SKE.Setup}(1^\lambda, 1^{\ell_{\mathsf{pri}}})\\ \mathsf{SKE.ct} &\leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, \mathbf{x}_{\mathsf{pri}})\\ \mathsf{sRFE.ct}_{\mathsf{on}} &\leftarrow \mathsf{sRFE.EncOn}\big(\mathsf{sRFE.st}, x'_{\mathsf{pub}} = (x_{\mathsf{pub}}, \mathsf{SKE.ct}), x'_{\mathsf{pri}} = \mathsf{SKE.sk}\big) .\end{aligned}$$

  Set $\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \| (\mathsf{aux}, \{(\mathsf{pk}_i^*, F_i)\}_{i\in[L]}, \mathsf{ct}^*)$.

- *Guess.* Run the adversary $\mathcal{A}_{\mathsf{post}}$ on input $(1^\lambda, \mathsf{rec}_{\mathsf{post}})$ whose output $b \in \{0,1\}$ is also the outcome of the experiment.

We need to show that if

$$\big(\mathsf{aux}, x_{\mathsf{pub}}, \{F_i\}_{i\in[L]}, \{\delta_i^* := F_i(x_{\mathsf{pub}}, x_{\mathsf{pri}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\big) \approx_c \big(\mathsf{aux}, x_{\mathsf{pub}}, \{F_i\}_{i\in[L]}, \{\delta_i^{\$} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{out}}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\big) , \tag{27}$$

then $\mathbf{Exp}^{\text{post-real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}} \approx_c \mathbf{Exp}^{\text{post-rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$, where $\mathbf{Exp}^{\text{post-rand}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$ proceeds in the same fashion as the experiment $\mathbf{Exp}^{\text{post-real}}_{\mathsf{sRFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}$ recalled above except that it replaces $\mathsf{ct}^* = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}^*_{\mathsf{on}} := (\mathsf{SKE.ct}, \mathsf{sRFE}'.\mathsf{ct}_{\mathsf{on}}))$ with

$$\mathsf{ct}^{\$} = \big(\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}^{\$}_{\mathsf{on}} \xleftarrow{\$} (\mathsf{SKE}.\mathcal{CT} \times \mathsf{sRFE}'.\mathcal{CT}_{\mathsf{on}})\big) .$$

Here, $\mathsf{SKE}.\mathcal{CT}$ denotes the ciphertext space of SKE and $\mathsf{sRFE}'.\mathcal{CT}_{\mathsf{on}}$ denotes the online part of the ciphertext space of sRFE'. We first deal with $\mathsf{sRFE}'.\mathsf{ct}_{\mathsf{on}}$. To prove its pseudorandomness, we invoke the two-stage Sel-prCT security of sRFE' with respect to a sampler $\mathsf{Samp}_{\mathsf{sRFE}'}(1^\lambda)$ which works as follows.

- *Setup.* Launch $\mathsf{Samp}(1^\lambda)$ and receive from it a challenge message $x_{\mathsf{pub}}, x_{\mathsf{pri}}$. Run

$$\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda, 1^{\ell_{\mathsf{pri}}}) \qquad \text{and} \qquad \mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, \mathbf{x}_{\mathsf{pri}}) .$$

  Then send the challenge message $(x'_{\mathsf{pub}} := (x_{\mathsf{pub}}, \mathsf{SKE.ct}), x'_{\mathsf{pri}} := \mathsf{SKE.sk})$ to the challenger of sRFE' to obtain $\mathsf{sRFE}'.\mathsf{crs}$ in return. Forward $\mathsf{crs} := \mathsf{sRFE}'.\mathsf{crs}$ to Samp.

- *Query.* Upon receiving a query from Samp, $\mathsf{Samp}_{\mathsf{sRFE}'}$ makes the same query to the corresponding oracle of the sRFE challenger and forwards the response to Samp.

- *Challenge.* Upon Samp outputting aux and $\{(\mathsf{pk}_i^*, F_i = \{F_{i,j}\}_{j\in[\ell_{\mathsf{out}}]})\}_{i\in[L]}$, $\mathsf{Samp}_{\mathsf{sRFE}'}$ outputs $\mathsf{aux}' = (\mathsf{aux}, x_{\mathsf{pub}}, \mathsf{SKE.ct}, \{F_i\}_{i\in[L]})$ and $\{(\mathsf{pk}_i^*, F_{\mathsf{reg}}[F_{i,j}])\}_{i\in[L],j\in[\ell_{\mathsf{out}}]}$.

Under the security of sRFE′, it suffices to show that

$$\begin{aligned}&\left(\mathsf{aux}_{\mathsf{sRFE}'}, \{F_{\mathsf{reg}}[F_{i,j}]\}_{i\in[L],j\in[\ell_{\mathsf{out}}]}, \{\Delta_{i,j}^* := F_{\mathsf{reg}}[F_{i,j}](x'_{\mathsf{pub}}, x'_{\mathsf{pri}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}},j\in[\ell_{\mathsf{out}}]}\right)\\ &\approx_c \left(\mathsf{aux}_{\mathsf{sRFE}'}, \{F_{\mathsf{reg}}[F_{i,j}]\}_{i\in[L],j\in[\ell_{\mathsf{out}}]}, \{\Delta_{i,j}^{\$} \xleftarrow{\$} \{0,1\}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}},j\in[\ell_{\mathsf{out}}]}\right).\end{aligned} \tag{28}$$

Unrolling the definitions, we observe that Equation (28) is equivalent to the following

$$\begin{aligned}&\left(\mathsf{aux}, x_{\mathsf{pub}}, \mathsf{SKE.ct}, \{F_i\}_{i\in[L]}, \{\mathbf{\Delta}_i^* := F_i(x_{\mathsf{pub}}, x_{\mathsf{pri}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right)\\ &\approx_c \left(\mathsf{aux}, x_{\mathsf{pub}}, \mathsf{SKE.ct}, \{F_i\}_{i\in[L]}, \{\mathbf{\Delta}_i^{\$} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{out}}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right).\end{aligned} \tag{29}$$

But this follows immediately from Equation (27) and the INDr security of SKE. Specifically, we have

$$\begin{aligned}&\left(\mathsf{aux}, x_{\mathsf{pub}}, \mathsf{SKE.ct}, \{F_i\}_{i\in[L]}, \{\mathbf{\Delta}_i^* := F_i(x_{\mathsf{pub}}, x_{\mathsf{pri}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right)\\ &\approx_c \left(\mathsf{aux}, x_{\mathsf{pub}}, \mathbf{\Gamma} \xleftarrow{\$} \mathsf{SKE}.\mathcal{CT}, \{F_i\}_{i\in[L]}, \{\mathbf{\Delta}_i^* := F_i(x_{\mathsf{pub}}, x_{\mathsf{pri}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right)\\ &\approx_c \left(\mathsf{aux}, x_{\mathsf{pub}}, \mathbf{\Gamma} \xleftarrow{\$} \mathsf{SKE}.\mathcal{CT}, \{F_i\}_{i\in[L]}, \{\mathbf{\Delta}_i^{\$} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{out}}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right)\\ &\approx_c \left(\mathsf{aux}, x_{\mathsf{pub}}, \mathsf{SKE.ct}, \{F_i\}_{i\in[L]}, \{\mathbf{\Delta}_i^{\$} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{out}}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right),\end{aligned}$$

where

- the first and the third indistinguishability follows from the INDr security of SKE and the fact that the secret key sk is freshly sampled and not used anywhere else, and
- the second indistinguishability follows from Equation (27) and the observation that adding a string $\mathbf{\Gamma}$ chosen independently at random does not simplify the task of distinguishing the two distributions.

Finally, using the pseudorandomness of $\mathsf{sRFE}'.\mathsf{ct}_{\mathsf{on}}$, the pseudorandomness of SKE.ct is immediate. Indeed, we can observe that at this point the secret key SKE.sk does not appear anywhere in the scheme anymore, thus we conclude from the INDr security of SKE that $\mathsf{SKE.ct} \approx_c \mathbf{\Gamma} \xleftarrow{\$} \mathsf{SKE}.\mathcal{CT}$. □

5 Applications to sRABE and sRPE with (Nearly) Optimal Parameters

In this section, we demonstrate how Sel-prCT secure sRABE can be used to build slotted Registered ABE (sRABE) and slotted Registered Predicate Encryption (sRPE).

5.1 Definition

The definitions for sRABE and sRPE are nearly identical. We therefore give the formal definitions only for the case of sRABE and mention differences compared to sRPE along the way.

Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$, $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be sequences of message, ciphertext attribute and key attribute spaces, respectively. Furthermore, let $R = \{R_\lambda\}_{\lambda\in\mathbb{N}}$ be a sequence of relations, where $R_\lambda : \mathcal{X}_\lambda \times \mathcal{Y}_\lambda \to \{0,1\}$ for all $\lambda \in \mathbb{N}$.

Definition 5.1 (Syntax of sRABE)**.** A sRABE scheme for message space $\mathcal{M}$ and relation R consists of five efficient algorithms:

$\mathsf{Setup}(1^\lambda, \mathsf{param}) \to \mathsf{crs}$**.** On input the security parameter 1^λ and some parameter param specifying R, this algorithm outputs a common reference string crs.

$\mathsf{Gen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$**.** On input the crs, this algorithm outputs a pair of a public and a secret key $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Agg}(\mathsf{crs}, (\mathsf{pk}_i, y_i)_{i\in[L]}) \to (\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]})$**.** On input the crs and L pairs (pk_i, y_i) with key attributes $y_i \in \mathcal{Y}_\lambda$, this algorithm outputs a master public key mpk and L helper secret keys $\{\mathsf{hsk}_i\}_{i\in[L]}$. We require Agg to be deterministic.

$\mathsf{Enc}(\mathsf{mpk}, x, \mu) \to \mathsf{ct}$**.** The encryption algorithm proceeds in two steps.

$\mathsf{EncOff}(\mathsf{mpk}) \to (\mathsf{ct}_{\mathsf{off}}, \mathsf{st})$**.** On input the master public key mpk, this algorithm outputs an offline ciphertext $\mathsf{ct}_{\mathsf{off}}$ and a state st.

$\mathsf{EncOn}(\mathsf{st}, x, \mu) \to \mathsf{ct}_{\mathsf{on}}$**.** On input a state st, a ciphertext attribute $x \in \mathcal{X}_\lambda$ and a message $\mu \in \mathcal{M}_\lambda$, this algorithm outputs an online ciphertext ct.

The final output of $\mathsf{Enc}(\mathsf{mpk}, x, \mu)$ is $\mathsf{ct} = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}})$.

$\mathsf{Dec}(\mathsf{sk}_i, \mathsf{hsk}_i, y_i, \mathsf{ct}, x) \to \mu' \vee \bot$**.** On input a secret key sk_i with corresponding helper secret key hsk_i and registered attribute y_i as well as a ciphertext ct with corresponding attribute x, this algorithm outputs a message $\mu' \in \mathcal{M}_\lambda$ or a special symbol $\bot$ indicating failure. We require Dec to be deterministic. In the case of sRPE, the algorithm does not take x as input.

Correctness. A sRABE scheme is correct if for all $\lambda, L \in \mathbb{N}$, $i^* \in [L]$, $\mu \in \mathcal{M}_\lambda$, $x \in \mathcal{X}_\lambda$ and $\{y_i\}_{i\in[L]} \subseteq \mathcal{Y}_\lambda$ such that $R(x, y_{i^*}) = 0$, it holds that

$$\Pr\left[\mu' = \mu \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param}) \\ \{(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{crs})\}_{i\in[L]} \\ (\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]}) := \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, y_i)\}_{i\in[L]}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, \mu) \\ \mu' := \mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, y_{i^*}, \mathsf{ct}, x) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of the algorithms Setup, Gen and Enc.

Compactness. A sRABE scheme is *compact* if for all $\lambda, L \in \mathbb{N}$ and $i \in [L]$, it holds that

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

Security. We define the notion of Sel-INDr security for schemes with two-stage encryption algorithms.

Definition 5.2 (Two-Stage Sel-INDr Security for sRABE)**.** A sRABE scheme sRABE is two-stage Sel-INDr secure if $\mathbf{Exp}^{\mathsf{srabe\text{-}real}}_{\mathsf{sRABE},\mathcal{A}}(1^\lambda) \approx_c \mathbf{Exp}^{\mathsf{srabe\text{-}rand}}_{\mathsf{sRABE},\mathcal{A}}(1^\lambda)$ for all PPT adversaries $\mathcal{A}$, where $\mathbf{Exp}^{\mathsf{srabe\text{-}str}}_{\mathsf{sRABE},\mathcal{A}}$ for $\mathsf{str} \in \{\mathsf{real}, \mathsf{rand}\}$ proceeds as follows.

- *Setup.* Launch $\mathcal{A}(1^\lambda)$ and receive from it the challenge $(x^*, \mu^*) \in \mathcal{X}_\lambda \times \mathcal{M}_\lambda$. Run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param})$ and send crs to $\mathcal{A}$. Initialize an empty set $\mathcal{C} := \varnothing$ and an empty dictionary $\mathcal{D}$.
- *Query.* Repeat the following for arbitrarily many rounds determined by $\mathcal{A}$. In each round, $\mathcal{A}$ has two options.
  - $\mathsf{QGen}()$: run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{crs}, i)$ and return pk to $\mathcal{A}$. Set $\mathcal{D}[\mathsf{pk}] := \mathsf{sk}$.
  - $\mathsf{QCor}(\mathsf{pk})$: upon $\mathcal{A}$ submitting a public key pk, return $\mathcal{D}[\mathsf{pk}]$ to $\mathcal{A}$. Add pk to $\mathcal{C}$.
- *Challenge.* The adversary submits L tuples $\{(\mathsf{pk}_i^*, y_i)\}_{i\in[L]}$ with $y_1, \ldots, y_L \in \mathcal{Y}_\lambda$ for some $L \in \mathbb{N}$ determined by $\mathcal{A}$. Run

$$\big(\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]}\big) \leftarrow \mathsf{Agg}(\mathsf{crs}, (\mathsf{pk}_i^*, y_i)_{i\in[L]}) \ , \qquad (\mathsf{ct}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{EncOff}(\mathsf{mpk}) \ ,$$

$$\begin{aligned} \text{in } \mathbf{Exp}^{\mathsf{srabe\text{-}real}}_{\mathsf{sRABE},\mathcal{A}}\text{:} &\quad \mathsf{ct}^*_{\mathsf{on}} \leftarrow \mathsf{EncOn}(\mathsf{st}, x^*, \mu^*) \ , \ \ \mathsf{ct} := (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}^*_{\mathsf{on}}) \\ \text{in } \mathbf{Exp}^{\mathsf{srabe\text{-}rand}}_{\mathsf{sRABE},\mathcal{A}}\text{:} &\quad \mathsf{ct}^{\$}_{\mathsf{on}} \xleftarrow{\$} \mathcal{CT}_{\lambda,\mathsf{on}} \ , \ \ \mathsf{ct} := (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}^{\$}_{\mathsf{on}}) \end{aligned}$$

  and return ct to $\mathcal{A}$.

- *Guess.* The adversary outputs a guess $\mathsf{str}' \in \{\mathsf{real}, \mathsf{rand}\}$. The outcome of the experiment is str' if $R(x^*, y_i) = 1$ for all $i \in [L]$ such that $\mathsf{pk}_i^* \in \mathcal{C}$ or $\mathcal{D}_i[\mathsf{pk}_i^*] = \bot$. Otherwise, the outcome is set to $\bot$.

We note that our notion of two-stage Sel-INDr security implies the traditional notion of selective IND-CPA security (defined below) for both sRABE and sRPE since the pseudorandom ciphertext component $\mathsf{ct}_{\mathsf{on}}^{\$}$ erases all information about the challenge (x^*, μ^*). Therefore, the only difference between our sRABE and sRPE definition is the efficiency requirement. For sRABE, we give the attribute x to the Dec algorithm and require that the ciphertext size is independent of the length of the attribute. On the other hand, for sRPE we require ciphertexts to hide their attribute x, so we cannot give x (in the clear) to the decryption algorithm. In this case, the ciphertext size necessarily grows linearly with the size of the attribute.

For completeness, we also recall the notion of classical Sel-IND security.

Definition 5.3 (Sel-IND Security for sRABE)**.** A sRABE scheme sRABE is Sel-IND secure if $\mathbf{Exp}^{\text{srabe-0}}_{\mathsf{sRABE},\mathcal{A}}(1^\lambda) \approx_c \mathbf{Exp}^{\text{srabe-1}}_{\mathsf{sRABE},\mathcal{A}}(1^\lambda)$ for all PPT adversaries $\mathcal{A}$, where $\mathbf{Exp}^{\text{srabe-}b}_{\mathsf{sRABE},\mathcal{A}}$ for $b \in \{0,1\}$ proceeds as follows.

- *Setup.* Launch $\mathcal{A}(1^\lambda)$ and receive from it the challenge input $(x^*, \mu_0^*, \mu_1^*) \in \mathcal{X}_\lambda \times \mathcal{M}_\lambda \times \mathcal{M}_\lambda$. Run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param})$ and send crs to $\mathcal{A}$. Initialize an empty set $\mathcal{C} := \varnothing$ and an empty dictionary $\mathcal{D}$.
- *Query.* This is identical to the query phase in Definition 5.2.
- *Challenge.* The adversary submits L tuples $\{(\mathsf{pk}_i^*, y_i)\}_{i\in[L]}$ with $y_1, \ldots, y_L \in \mathcal{Y}_\lambda$ for some $L \in \mathbb{N}$ determined by $\mathcal{A}$. Compute the challenge ciphertext ct as follows and return it to $\mathcal{A}$:

$$\big(\mathsf{mpk}, \{\mathsf{hsk}_i\}_{i\in[L]}\big) \leftarrow \mathsf{Agg}(\mathsf{crs}, (\mathsf{pk}_i^*, y_i)_{i\in[L]}) \,, \qquad \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{st}, x^*, \mu_b^*) \,.$$

- *Guess.* The adversary outputs a guess $b' \in \{0,1\}$. The outcome of the experiment is b' if $R(x^*, y_i) = 1$ for all $i \in [L]$ such that $\mathsf{pk}_i^* \in \mathcal{C}$ or $\mathcal{D}_i[\mathsf{pk}_i^*] = \bot$. Otherwise, the outcome is set to $\bot$.

Policy Classes. We consider sRABE for the same policy classes as plain ABE (see paragraph **Policy Classes** in Section 3.11).

5.2 Construction of KP-sRABE and sRPE for Unbounded Depth Circuits and TMs

We build KP-sRABE and sRPE schemes for the class $\mathcal{C}_\ell$ of all (unbounded depth) circuits $C\colon \{0,1\}^\ell \to \{0,1\}$ and the class $\mathcal{T}$ of all Turing machines. As all constructions are very similar, we prefer to present them at once. We use solid (resp. dashed) boxes to indicate that a component appears only in the case of $\mathcal{C}_\ell$ (resp. $\mathcal{T}$). Similarly, we use gray (resp. blue) to indicate that a component is only used in the case of KP-sRABE (resp. sRPE).

For notational simplicity, we describe all constructions for the fixed message space $\mathcal{M} = \{0,1\}^\lambda$. We note that this restriction is without loss of generality. Indeed, using the hybrid encryption framework we can upgrade these schemes to support arbitrary message spaces while preserving our asymptotic efficiency parameters.

Construction 5.4 (KP-sRABE and sRPE for $\mathcal{C}_\ell$ and $\mathcal{T}$)**.** The construction uses the following ingredients:

- A Sel-prCT secure sRFE scheme $\mathsf{sRFE} = \mathsf{sRFE}.(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Agg}, \mathsf{EncOff}, \mathsf{EncOn}, \mathsf{Dec})$ for the following function class $\mathcal{F}$, where

KP-sRABE for $\mathcal{C}_\ell$:	$\mathcal{F} = \mathcal{C}_{\ell_{\mathsf{pub}}, \ell_{\mathsf{pri}}, \ell_{\mathsf{out}}}$	with $\ell_{\mathsf{pub}} = \ell$, $\ell_{\mathsf{pri}} = 2\lambda$ and $\ell_{\mathsf{out}} = \lambda$,
KP-sRABE for $\mathcal{T}$:	$\mathcal{F} = \mathcal{T}_{\ell_{\mathsf{pri}}, \ell_{\mathsf{out}}}$	with $\ell_{\mathsf{pri}} = 2\lambda$ and $\ell_{\mathsf{out}} = \lambda$,
sRPE for $\mathcal{C}_\ell$:	$\mathcal{F} = \mathcal{C}_{\ell_{\mathsf{pub}}, \ell_{\mathsf{pri}}, \ell_{\mathsf{out}}}$	with $\ell_{\mathsf{pub}} = 0$, $\ell_{\mathsf{pri}} = \ell + 2\lambda$ and $\ell_{\mathsf{out}} = \lambda$,
sRPE for $\mathcal{T}$:	$\mathcal{F} = \mathcal{T}_{\ell_{\mathsf{out}}}$	with $\ell_{\mathsf{out}} = \lambda$.

Such sRFE schemes exist assuming prFE and LWE (see Construction 4.14, Theorem 4.17).

- A pseudorandom function $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ with key space, input space and output space being $\{0,1\}^\lambda$. PRF can be instantiated assuming one-way functions.

The details of the construction are as follows.

$\mathsf{Setup}(1^\lambda, \boxed{1^\ell})$. Run

$$\mathsf{sRFE.crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda, \boxed{1^{\ell_{\mathsf{pub}}}}, 1^{\ell_{\mathsf{pri}}}, 1^{\ell_{\mathsf{out}}}) ,$$

and output $\mathsf{crs} := \mathsf{sRFE.crs}$.

$\mathsf{Gen}(\mathsf{crs})$. Parse $\mathsf{crs} = \mathsf{sRFE.crs}$, run

$$\mathsf{sRFE.}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{sRFE.Gen}(\mathsf{sRFE.crs}) ,$$

and output the key pair ($\mathsf{pk} := \mathsf{sRFE.pk}, \mathsf{sk} := \mathsf{sRFE.sk}$).

$\mathsf{Agg}(\mathsf{crs}, \{\mathsf{pk}_i, f_i\}_{i\in[L]})$. Parse $\mathsf{crs} = \mathsf{sRFE.crs}$ and $\mathsf{pk}_i = \mathsf{sRFE.pk}_i$ for $i \in [L]$. Run

$$\big(\mathsf{sRFE.mpk}, \{\mathsf{sRFE.hsk}_i\}_{i\in[L]}\big) \leftarrow \mathsf{sRFE.Agg}\big(\mathsf{sRFE.crs}, \{(\mathsf{sRFE.pk}_i, F_{\mathsf{reg}}[i, f_i])\}_{i\in[L]}\big) ,$$

where $F_{\mathsf{reg}}[i, f_i]$ is defined in Figure 11. Then output $\mathsf{mpk} := \mathsf{sRFE.mpk}$ and $\mathsf{hsk}_i := \mathsf{sRFE.hsk}_i$ for all $i \in [L]$.

Public Input: an attribute $x \in \mathcal{X}$

Private Input:
- an attribute $x \in \mathcal{X}$
- a message $\mu \in \mathcal{M}$
- a PRF seed sd

Output: a bit string in $\{0,1\}^\lambda$

Hardwired Values:
- a slot index $i \in [L]$
- a policy f_i

- Compute $b = f_i(x)$.
- Output μ if $b = 0$ and $\mathsf{PRF}(\mathsf{sd}, i)$ if $b = 1$.

Figure 11: Definition of the function $F_{\mathsf{reg}}[i, f_i]$

$\mathsf{Enc}(\mathsf{mpk}, x, \mu)$. The encryption algorithm proceeds in two steps.

$\mathsf{EncOff}(\mathsf{mpk})$. Parse $\mathsf{mpk} = \mathsf{sRFE.mpk}$, run

$$\mathsf{sRFE.}(\mathsf{ct}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{sRFE.EncOff}(\mathsf{sRFE.mpk})$$

and return ($\mathsf{ct}_{\mathsf{off}} := \mathsf{sRFE.ct}_{\mathsf{off}}, \mathsf{st} = \mathsf{sRFE.st}$)

$\mathsf{EncOn}(\mathsf{st}, x, \mu)$. Parse $\mathsf{st} = \mathsf{sRFE.st}$, sample a PRF seed $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$ and set $x_{\mathsf{pub}} := x,\ x_{\mathsf{pri}} = (\mu, \mathsf{sd})$ (resp. $x_{\mathsf{pub}} := \varepsilon,\ x_{\mathsf{pri}} = (x, \mu, \mathsf{sd})$). Then run

$$\mathsf{sRFE.ct} \leftarrow \mathsf{sRFE.Enc}(\mathsf{sRFE.mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}) ,$$

and output $\mathsf{ct}_{\mathsf{on}} = \mathsf{sRFE.ct}_{\mathsf{on}}$.

The final output of $\mathsf{Enc}(\mathsf{mpk}, x, \mu)$ is $\mathsf{ct} = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}})$.

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, f_{i^*}, \mathsf{ct}, x)$. Parse $\mathsf{sk}_{i^*} = \mathsf{sRFE.sk}_{i^*}$, $\mathsf{hsk}_{i^*} = \mathsf{sRFE.hsk}_{i^*}$ and $\mathsf{ct} = \mathsf{sRFE.ct}$. Set $x_{\mathsf{pub}} := x$ (resp. $x_{\mathsf{pub}} := \varepsilon$), run

$$z \leftarrow \mathsf{sRFE.Dec}(\mathsf{sRFE.sk}_{i^*}, \mathsf{sRFE.hsk}_{i^*}, F_{\mathsf{reg}}[i^*, f_{i^*}], \mathsf{sRFE.ct}, x_{\mathsf{pub}}) ,$$

and output z, where $F_{\mathsf{reg}}[i^*, f_{i^*}]$ is defined in Figure 11.

Proposition 5.5 (Correctness and Compactness)**.** *The KP-sRABE and sRPE schemes described in Construction 5.4 are correct and compact. Specifically, they have the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log\log L + \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = \log L \cdot \mathsf{poly}(\lambda) + \ell .$$

Proposition 5.6 (Security)**.** *If* sRFE *is two-stage* Sel-prCT *secure and* PRF *is secure, then the Construction 5.4 is two-stage* Sel-INDr *secure.*

The proofs of Propositions 5.5 and 5.6 can be found in Sections 5.3 and 5.4, respectively. Summarizing our results we obtain the following theorem. Specifically, we consider a ciphertext ct encrypting a message μ of length ℓ_{in} (using the hybrid encryption framework) with respect to an attribute x of length $\ell_{\mathsf{att}} = \ell$.

Theorem 5.7. *Assuming LWE and prFE, there exist* Sel-INDr *secure KP-sRABE and sRPE schemes supporting the policy classes $\mathcal{C}_\ell$ and $\mathcal{T}$ with the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log\log L + \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = \log L \cdot \mathsf{poly}(\lambda) + \ell_{\mathsf{in}} + \ell_{\mathsf{att}} .$$

5.3 Proof of Correctness and Compactness

Proof of Proposition 5.5. We argue that Construction 5.4 is correct and compact.

Correctness. The correctness readily follows from the correctness of sRFE and the definition of $F_{\mathsf{reg}}[i^*, f_{i^*}]$. Specifically, for $\mathsf{sk}_{i^*} = \mathsf{sRFE.sk}_{i^*}$, $\mathsf{hsk}_{i^*} = \mathsf{sRFE.hsk}_{i^*}$, $\mathsf{ct} = \mathsf{sRFE.ct}$ and $x_{\mathsf{pub}} := x$ (resp. $x_{\mathsf{pub}} := \varepsilon$), we have

$$\begin{aligned} z &:= \mathsf{sRFE.Dec}\big(\mathsf{sRFE.sk}_{i^*}, \mathsf{sRFE.hsk}_{i^*}, F_{\mathsf{reg}}[i^*, f_{i^*}], \mathsf{sRFE.ct}, x_{\mathsf{pub}}\big) \\ &= F_{\mathsf{reg}}[i^*, f_{i^*}](x, \mu, \mathsf{sd}) \\ &= \begin{cases} \mu & \text{if } f_i(x) = 0 \\ \mathsf{PRF}(\mathsf{sd}, i^*) & \text{if } f_i(x) = 1 . \end{cases} \end{aligned}$$

Compactness. The parameters can be obtained by plugging the values of ℓ_{pub}, ℓ_{pri} and ℓ_{out} into the parameters of sRFE (Construction 4.14, Theorem 4.17). □

5.4 Proof of Security

Proof of Proposition 5.6. We first give the proof for the case of KP-sRABE and briefly mention the case of sRPE at the end. Let sRABE denote the KP-sRABE (resp. sRPE) scheme in Construction 5.4 and $\mathcal{A}$ a PPT adversary. We recall the experiments $\mathbf{Exp}^{\mathsf{srabe\text{-}real}}_{\mathsf{sRABE},\mathcal{A}}$ and $\mathbf{Exp}^{\mathsf{srabe\text{-}rand}}_{\mathsf{sRABE},\mathcal{A}}$.

- *Setup.* Upon launching $\mathcal{A}(1^\lambda; \mathsf{coins}_{\mathcal{A}})$, the adversary outputs the challenge input (x^*, μ^*). The challenger sends $\mathsf{crs} := \mathsf{sRFE.crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda, \boxed{1^{\ell_{\mathsf{pub}}}}, 1^{\ell_{\mathsf{pri}}}, 1^{\ell_{\mathsf{out}}})$ to $\mathcal{A}$ and initializes an empty set $\mathcal{C} := \varnothing$ and an empty dictionary $\mathcal{D}$.
- *Query.* $\mathcal{A}$ has access to the following two oracles.

– QGen(): run $\mathsf{sRFE.(pk,sk)} \leftarrow \mathsf{sRFEGen(sRFE.crs)}$ and send $\mathsf{pk} := \mathsf{sRFE.pk}$ to $\mathcal{A}$. Set $\mathcal{D}[\mathsf{pk}] := \mathsf{sRFE.sk}$.

– QCor(pk): return $\mathcal{D}[\mathsf{pk}]$ to $\mathcal{A}$. Add pk to $\mathcal{C}$.

- *Challenge.* Upon $\mathcal{A}$ submitting $\{(\mathsf{pk}_i^*, f_i)\}_{i\in[L]}$, the challenger parses $\mathsf{pk}_i^* = \mathsf{sRFE.pk}_i^*$, samples $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$ and returns ct to $\mathcal{A}$ as follows:

$$\begin{aligned}\big(\mathsf{sRFE.mpk}, \{\mathsf{sRFE.hsk}_i\}_{i\in[L]}\big) &\leftarrow \mathsf{sRFE.Agg}\big(\mathsf{sRFE.crs}, \{(\mathsf{sRFE.pk}_i^*, F_{\mathsf{reg}}[i,f_i])\}_{i\in[L]}\big)\\ \mathsf{sRFE.}(\mathsf{ct}_{\mathsf{off}}, \mathsf{st}) &\leftarrow \mathsf{sRFE.EncOff(sRFE.mpk)}\end{aligned}$$

$$\begin{aligned}\text{in } \mathbf{Exp}^{\mathsf{srabe\text{-}real}}_{\mathsf{sRABE},\mathcal{A}}\text{:} &\quad \mathsf{sRFE.ct}_{\mathsf{on}}^* \leftarrow \mathsf{sRFE.EncOn}(\mathsf{sRFE.st}, x^*, \mu^*)\ ,\ \ \mathsf{ct} := (\mathsf{sRFE.ct}_{\mathsf{off}}, \mathsf{sRFE.ct}_{\mathsf{on}}^*)\\ \text{in } \mathbf{Exp}^{\mathsf{srabe\text{-}rand}}_{\mathsf{sRABE},\mathcal{A}}\text{:} &\quad \mathsf{sRFE.ct}_{\mathsf{on}}^{\$} \xleftarrow{\$} \mathsf{sRFE.}\mathcal{CT}_{\lambda,\mathsf{on}}\ ,\ \ \mathsf{ct} := (\mathsf{sRFE.ct}_{\mathsf{off}}, \mathsf{sRFE.ct}_{\mathsf{on}}^{\$})\ ,\end{aligned}$$

where $\mathsf{sRFE.}\mathcal{CT}_{\mathsf{on}}$ denotes the ciphertext space of sRFE.EncOn.

- *Guess.* $\mathcal{A}$ outputs a guess $\mathsf{str}' \in \{\mathsf{real}, \mathsf{rand}\}$. The outcome of the experiment is str' if $f_i(x^*) = 1$ for all $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$, where $\mathcal{I}_{\mathsf{cor}} = \{i \in [L] : \mathsf{pk}_i^* \in \mathcal{C}\}$ and $\mathcal{I}_{\mathsf{mal}} = \{i \in [L] : \mathcal{D}[\mathsf{pk}_i^*] = \bot\}$. Otherwise, the outcome is set to $\bot$.

To prove $\mathbf{Exp}^{\mathsf{srabe\text{-}real}}_{\mathsf{sRABE},\mathcal{A}} \approx_c \mathbf{Exp}^{\mathsf{srabe\text{-}rand}}_{\mathsf{sRABE},\mathcal{A}}$, we invoke the security of sRFE with respect to the following sampler $\mathsf{Samp}_{\mathsf{sRFE}}(1^\lambda)$.

- *Setup.* Sample $\mathsf{sd}, \mathsf{coins}_{\mathcal{A}} \xleftarrow{\$} \{0,1\}^\lambda$ and launch $\mathcal{A}(1^\lambda; \mathsf{coins}_{\mathcal{A}})$ to obtain (x^*, μ^*). Output $x_{\mathsf{pub}} = x^*$, $x_{\mathsf{pri}} = (\mu^*, \mathsf{sd})$ and receive $\mathsf{crs} := \mathsf{sRFE.crs}$ in return. Forward sRFE.crs to $\mathcal{A}$.

- *Query.* Upon $\mathcal{A}$ submitting a query QGen() or QCor(pk), $\mathsf{Samp}_{\mathsf{sRFE}}$ makes the same query to its own challenger and forwards the response to $\mathcal{A}$.

- *Challenge.* Upon $\mathcal{A}$ submitting $\{(\mathsf{pk}_i^*, f_i)\}_{i\in[L]}$, $\mathsf{Samp}_{\mathsf{sRFE}}$ outputs $\mathsf{aux}_{\mathsf{sRFE}} = (\mathsf{coins}_{\mathcal{A}}, \{f_i\}_{i\in[L]})$ and L tuples $\{(\mathsf{pk}_i^*, F_{\mathsf{reg}}[i, f_i])\}_{i\in[L]}$.

From the security of sRFE with $\mathsf{Samp}_{\mathsf{sRFE}}$, we obtain that $\mathbf{Exp}^{\mathsf{srabe\text{-}real}}_{\mathsf{sRABE},\mathcal{A}} \approx_c \mathbf{Exp}^{\mathsf{srabe\text{-}rand}}_{\mathsf{sRABE},\mathcal{A}}$ if $\mathsf{rec}^*_{\mathsf{pre}} \approx_c \mathsf{rec}^{\$}_{\mathsf{pre}}$, where

$$\begin{aligned}\mathsf{rec}^*_{\mathsf{pre}} &:= \big(\mathsf{aux}_{\mathsf{sRFE}}, x_{\mathsf{pub}}, \{F_{\mathsf{reg}}[i,f_i]\}_{i\in[L]}, \{\delta_i^* := F_{\mathsf{reg}}[i,f_i](x_{\mathsf{pub}}, x_{\mathsf{pri}})\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\big)\\ \mathsf{rec}^{\$}_{\mathsf{pre}} &:= \big(\mathsf{aux}_{\mathsf{sRFE}}, x_{\mathsf{pub}}, \{F_{\mathsf{reg}}[i,f_i]\}_{i\in[L]}, \{\delta_i^{\$} \xleftarrow{\$} \{0,1\}^\lambda\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\big)\ .\end{aligned}$$

To see this, we recall that

$$F_{\mathsf{reg}}[i,f_i]\big(x_{\mathsf{pub}} = x^*, x_{\mathsf{pri}} = (\mu^*, \mathsf{sd})\big) = \begin{cases}\mu & \text{if } f_i(x^*) = 0\ ,\\ \mathsf{PRF}(\mathsf{sd}, i) & \text{if } f_i(x^*) = 1\ .\end{cases}$$

From the admissibility of the adversary in $\mathbf{Exp}^{\mathsf{srabe\text{-}real}}_{\mathsf{sRABE},\mathcal{A}}$ and $\mathbf{Exp}^{\mathsf{srabe\text{-}rand}}_{\mathsf{sRABE},\mathcal{A}}$, we have that $f_i(x^*) = 1$ for all $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$ in which case the security of PRF implies that

$$\delta_i^* := F_{\mathsf{reg}}[i,f_i]\big(x_{\mathsf{pub}} = x^*, x_{\mathsf{pri}} = (\mu^*, \mathsf{sd})\big) = \mathsf{PRF}(\mathsf{sd}, i) \approx_c \delta_i^{\$}\ .$$

The argument for the case of sRPE is almost identical. In particular, $\mathsf{Samp}_{\mathsf{sRFE}}$ defines $x_{\mathsf{pub}} = \varepsilon$ and $x_{\mathsf{pri}} = (x^*, \mu^*, \mathsf{sd})$. Then replacing the ciphertext by a random string hides the information of both x and μ and, thus, satisfies the security requirement for a sRPE scheme. □

5.5 Construction of CP-sRABE for Unbounded Depth Circuits and TMs

We build CP-sRABE schemes for the class $\mathcal{C}_\ell$ of all (unbounded depth) circuits $C\colon \{0,1\}^\ell \to \{0,1\}$ and the class $\mathcal{T}$ of all Turing machines. As in the case of KP-sRABE, we use solid (resp. dashed) boxes to indicate that a component appears only in the case of $\mathcal{C}_\ell$ (resp. $\mathcal{T}$). Also, we describe the construction for the fixed message space $\mathcal{M} = \{0,1\}^\lambda$ which can be generically upgraded to support arbitrary message spaces using the hybrid encryption framework.

Construction 5.8 (CP-sRABE for $\boxed{\mathcal{C}_\ell}$ and $\boxed{\mathcal{T}}$)**.** The construction uses the following building blocks:

- A KP-ABE scheme $\mathsf{kpABE} = \mathsf{kpABE}.(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc} = (\mathsf{EncOff}, \mathsf{EncOn}), \mathsf{Dec})$ for policy class $\boxed{\mathcal{C}_\ell}$ $\boxed{\mathcal{T}}$ satisfying reusable VerSel-INDr security. We denote the output space of $\mathsf{kpABE}.\mathsf{EncOn}$ by $\mathsf{kpABE}.\mathcal{CT}_{\mathsf{on}} = \{0,1\}^{\ell_{\mathsf{on}}}$ and require its size to be some fixed polynomial $\ell_{\mathsf{on}} = \ell_{\mathsf{on}}(\lambda)$ in λ. Without loss of generality, we assume that the state output by $\mathsf{kpABE}.\mathsf{EncOff}$ and the random coins of $\mathsf{kpABE}.\mathsf{EncOn}$ be in $\{0,1\}^\lambda$. To achieve the former, we let the state be the random coins used by $\mathsf{kpABE}.\mathsf{EncOff}$ which can in turn be replaced with a string in $\{0,1\}^\lambda$ using a PRF. For the latter, we can also use a PRF to derive longer (pseudo-)random coins if needed. KP-ABE schemes with the desired properties exist assuming LWE, pPRIO and reusable VerSel-prCT secure FE for bounded depth circuits (Fact 3.32, Construction 7.5, Theorem 7.12).
- A two-stage Sel-prCT secure sRFE scheme $\mathsf{sRFE} = \mathsf{sRFE}.(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Agg}, \mathsf{EncOff}, \mathsf{EncOn}, \mathsf{Dec})$ for the circuit class $\mathcal{C}_{\ell_{\mathsf{pub}},\ell_{\mathsf{pri}},\ell_{\mathsf{out}}}$, where $\ell_{\mathsf{pub}} = 0$, $\ell_{\mathsf{pri}} = 3\lambda$ and $\ell_{\mathsf{out}} = \ell_{\mathsf{on}}$. We can construct such an sRFE scheme assuming prFE and LWE (see Construction 5.8, Theorem 4.17).
- A pseudorandom function $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ with key space, input space and output space being $\{0,1\}^\lambda$. PRF can be instantiated assuming one-way functions.

The details of the CP-sRABE scheme are as follows.

$\mathsf{Setup}(1^\lambda, \boxed{1^\ell})$**:** Run

$$\mathsf{sRFE.crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda, 1^{\ell_{\mathsf{pub}}}, 1^{\ell_{\mathsf{pri}}}, 1^{\ell_{\mathsf{out}}}) ,$$

and output $\mathsf{crs} := \mathsf{sRFE.crs}$.

$\mathsf{Gen}(\mathsf{crs})$**:** Parse $\mathsf{crs} = \mathsf{sRFE.crs}$, run

$$\mathsf{sRFE}.(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{sRFE.Gen}(\mathsf{sRFE.crs}) ,$$

and output the key pair $(\mathsf{pk} := \mathsf{sRFE.pk}, \mathsf{sk} := \mathsf{sRFE.sk})$.

$\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, x_i)\}_{i\in[L]})$**:** Parse $\mathsf{crs} = \mathsf{sRFE.crs}$ and $\mathsf{pk}_i = \mathsf{sRFE.pk}_i$ for $i \in [L]$. Run

$$\big(\mathsf{sRFE.mpk}, \{\mathsf{sRFE.hsk}_i\}_{i\in[L]}\big) \leftarrow \mathsf{sRFE.Agg}\big(\mathsf{sRFE.crs}, \{(\mathsf{sRFE.pk}_i, C_{\mathsf{reg}}[i, x_i])\}_{i\in[L]}\big) ,$$

where $C_{\mathsf{reg}}[i, x_i]$ is defined in Figure 12. Output $\mathsf{mpk} := \mathsf{sRFE.mpk}$ and $\mathsf{hsk}_i := \mathsf{sRFE.hsk}_i$ for all $i \in [L]$.

(Private) Input:
- a message $\mu \in \mathcal{M}$
- a kpABE state kpABE.st
- a PRF seed sd

Output: a kpABE online ciphertext $\mathsf{kpABE.ct}_{\mathsf{on}} \in \mathsf{kpABE}.\mathcal{CT}_{\mathsf{on}}$

Hardwired Values:
- a slot index $i \in [L]$
- an attribute $x_i \in \mathcal{X}$

- Compute $R_i = \mathsf{PRF}(\mathsf{sd}, i)$.
- Output $\mathsf{kpABE.ct}_{\mathsf{on}} \leftarrow \mathsf{kpABE.EncOn}(\mathsf{kpABE.st}, x_i, \mu; R_i)$.

Figure 12: Definition of the circuit $C_{\mathsf{reg}}[i, x_i]$ in Construction 5.8

$\mathsf{Enc}(\mathsf{mpk}, f, \mu)$**:** Parse $\mathsf{mpk} = \mathsf{sRFE.mpk}$, sample $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$ and run

$$\begin{aligned}
\mathsf{kpABE}.(\mathsf{mpk}, \mathsf{msk}) &\leftarrow \mathsf{kpABE.Setup}(1^\lambda, \boxed{1^\ell}) \\
\mathsf{kpABE}.(\mathsf{ct}_{\mathsf{off}}, \mathsf{st}) &\leftarrow \mathsf{kpABE.EncOff}(\mathsf{kpABE.mpk}) \\
\mathsf{kpABE.sk}_f &\leftarrow \mathsf{kpABE.KeyGen}(\mathsf{kpABE.msk}, f) \\
\mathsf{sRFE.ct} &\leftarrow \mathsf{sRFE.Enc}\big(\mathsf{sRFE.mpk}, x_{\mathsf{pub}} = \varepsilon, x_{\mathsf{pri}} = (\mu, \mathsf{kpABE.st}, \mathsf{sd})\big) .
\end{aligned}$$

Output $\mathsf{ct} := (\mathsf{kpABE.ct}_{\mathsf{off}}, \mathsf{kpABE.sk}_f, \mathsf{sRFE.ct})$.

$\mathsf{Dec}(\mathsf{sk}_{i^*}, \mathsf{hsk}_{i^*}, x_{i^*}, \mathsf{ct}, f)$**:** Parse $\mathsf{sk}_{i^*} = \mathsf{sRFE.sk}_{i^*}$, $\mathsf{hsk}_{i^*} = \mathsf{sRFE.hsk}_{i^*}$, $\mathsf{ct} = (\mathsf{kpABE.ct}_{\mathsf{off}}, \mathsf{kpABE.sk}_f, \mathsf{sRFE.ct})$. Run

$$\mathsf{kpABE.ct}_{\mathsf{on}} \leftarrow \mathsf{sRFE.Dec}\big(\mathsf{sRFE.sk}_{i^*}, \mathsf{sRFE.hsk}_{i^*}, C_{\mathsf{reg}}[i^*, x_{i^*}], \mathsf{sRFE.ct}, \varepsilon\big)$$
$$z \leftarrow \mathsf{kpABE.Dec}\big(\mathsf{kpABE.msk}, \mathsf{kpABE.sk}_f, f, \mathsf{kpABE.}(\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}}), x_{i^*}\big)\,,$$

and output z, where $C_{\mathsf{reg}}[i^*, x_{i^*}]$ is defined in Figure 12.

Proposition 5.9 (Correctness and Compactness)**.** *The CP-sRABE scheme in Construction 5.8 is correct and compact. More specifically, it has the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log\log L + \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = \log L \cdot \mathsf{poly}(\lambda)\,.$$

Proposition 5.10 (Security)**.** *If* kpABE *satisfies reusable* VerSel-INDr *security,* sRFE *is two-stage* Sel-prCT *secure and* PRF *is secure, then the CP-sRABE scheme in Construction 5.8 is* Sel-IND *secure.*

The proofs of Propositions 5.9 and 5.10 can be found in Sections 5.6 and 5.7, respectively. We summarize our results in the following theorem. Specifically, we consider a ciphertext ct encrypting a message μ of length ℓ_{in} (using the hybrid encryption framework).

Theorem 5.11. *Assuming LWE and prFE, there exist* Sel-IND *secure CP-sRABE schemes supporting the policy classes $\mathcal{C}_\ell$ and $\mathcal{T}$ with the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda) \qquad |\mathsf{mpk}| = \log\log L + \mathsf{poly}(\lambda)$$
$$|\mathsf{hsk}_i| = \log L \cdot \mathsf{poly}(\lambda) \qquad |\mathsf{ct}| = \log L \cdot \mathsf{poly}(\lambda) + \ell_{\mathsf{in}}\,.$$

5.6 Proof of Correctness and Compactness

Proof of Proposition 5.9. We argue that Construction 5.8 is correct and compact.

Correctness. Correctness readily follows from the correctness of the building blocks and the definition of $C_{\mathsf{reg}}[i^*, x_{i^*}]$. Specifically, for $\mathsf{sk}_{i^*} = \mathsf{sRFE.sk}_{i^*}$, $\mathsf{hsk}_{i^*} = \mathsf{sRFE.hsk}_{i^*}$ and $\mathsf{ct} = (\mathsf{kpABE.ct}_{\mathsf{off}}, \mathsf{kpABE.sk}_f, \mathsf{sRFE.ct})$, we have

$$\begin{aligned}
&\mathsf{sRFE.Dec}\big(\mathsf{sRFE.sk}_{i^*}, \mathsf{sRFE.hsk}_{i^*}, C_{\mathsf{reg}}[i^*, x_{i^*}], \mathsf{sRFE.ct}, \varepsilon\big)\\
&= C_{\mathsf{reg}}[i^*, x_{i^*}](\mu, \mathsf{kpABE.st}, \mathsf{sd})\\
&= \mathsf{kpABE.EncOn}(\mu, \mathsf{kpABE.st}, x_{i^*}; \mathsf{PRF}(\mathsf{sd}, i^*)) =: \mathsf{kpABE.ct}_{\mathsf{on}}
\end{aligned}$$

Plugging this online ciphertext into the kpABE decryption algorithm yields

$$\mathsf{kpABE.Dec}\big(\mathsf{kpABE.msk}, \mathsf{kpABE.sk}_f, f, \mathsf{kpABE.}(\mathsf{ct}_{\mathsf{off}}.\mathsf{ct}_{\mathsf{on}}), x_{i^*}\big) = \mu \qquad \text{if } f(x_i^*) = 0\,.$$

Compactness. First, when instating kpABE as proposed (Fact 3.32, Theorem 7.12), we have

$$|\mathsf{kpABE.mpk}| = \mathsf{poly}(\lambda)\,, \quad |\mathsf{kpABE.ct}_{\mathsf{off}}| = \mathsf{poly}(\lambda)\,, \quad |\mathsf{kpABE.ct}_{\mathsf{on}}| = \mathsf{poly}(\lambda)\,, \quad |\mathsf{kpABE.sk}_f| = \mathsf{poly}(\lambda)\,.$$

Then, plugging the values of $\ell_{\mathsf{pub}} = 0$, $\ell_{\mathsf{pri}} = 3\lambda$ and $\ell_{\mathsf{out}} = \ell_{\mathsf{on}}$ into the parameters of sRFE (Construction 4.14, Theorem 4.17) gives

$$\begin{aligned}
|\mathsf{crs}| &= |\mathsf{sRFE.crs}| = \mathsf{poly}(\lambda)\\
|\mathsf{mpk}| &= |\mathsf{sRFE.mpk}| = \log\log L + \mathsf{poly}(\lambda)\\
|\mathsf{hsk}_i| &= |\mathsf{sRFE.hsk}_i| = \log L \cdot \ell_{\mathsf{out}} \cdot \mathsf{poly}(\lambda) = \log L \cdot \mathsf{poly}(\lambda)\\
|\mathsf{ct}| &= |\mathsf{kpABE.ct}_{\mathsf{off}}| + |\mathsf{kpABE.sk}_f| + \underbrace{|\mathsf{sRFE.ct}|}_{\log L \cdot \mathsf{poly}(\lambda) + \ell_{\mathsf{pri}}} = \log L \cdot \mathsf{poly}(\lambda)\,,
\end{aligned}$$

where we recall that $\ell_{\mathsf{on}}(\lambda) = \mathsf{poly}(\lambda)$. □

5.7 Proof of Security

Proof of Proposition 5.10. Let sRABE denote the CP-sRABE scheme in Construction 5.8 and $\mathcal{A}$ a PPT adversary. We recall the experiments $\mathbf{Exp}^{\mathsf{srabe}\text{-}b}_{\mathsf{sRABE},\mathcal{A}}$ for $b \in \{0,1\}$.

- *Setup.* Launch $\mathcal{A}(1^\lambda)$ to obtain the challenge input (f^*, μ_0^*, μ_1^*). The challenger sends $\mathsf{crs} := \mathsf{sRFE.crs} \leftarrow \mathsf{sRFE.Setup}(1^\lambda, 1^{\ell_{\mathsf{pub}}}, 1^{\ell_{\mathsf{pri}}}, 1^{\ell_{\mathsf{out}}})$, for $\ell_{\mathsf{pub}} = 0$, $\ell_{\mathsf{pri}} = 3\lambda$ and $\ell_{\mathsf{out}} = \ell_{\mathsf{on}}$, to $\mathcal{A}$ and initializes an empty set $\mathcal{C} := \varnothing$ and an empty dictionary $\mathcal{D}$.
- *Query.* $\mathcal{A}$ has access to the following two oracles.
  - $\mathsf{QGen}()$: run $\mathsf{sRFE.}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{sRFEGen}(\mathsf{sRFE.crs})$ and send $\mathsf{pk} := \mathsf{sRFE.pk}$ to $\mathcal{A}$. Set $\mathcal{D}[\mathsf{pk}] := \mathsf{sRFE.sk}$.
  - $\mathsf{QCor}(\mathsf{pk})$: return $\mathcal{D}[\mathsf{pk}]$ to $\mathcal{A}$. Add pk to $\mathcal{C}$.
- *Challenge.* Upon $\mathcal{A}$ and $\{(\mathsf{pk}_i^*, x_i)\}_{i\in[L]}$, the challenger parses $\mathsf{pk}_i^* = \mathsf{sRFE.pk}_i^*$, samples $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$ and returns $\mathsf{ct} := (\mathsf{kpABE.ct}_{\mathsf{off}}, \mathsf{kpABE.sk}_{f^*}, \mathsf{sRFE.ct})$ to $\mathcal{A}$ as follows:

$$
\begin{aligned}
\big(\mathsf{sRFE.mpk}, \{\mathsf{sRFE.hsk}_i\}_{i\in[L]}\big) &\leftarrow \mathsf{sRFE.Agg}\big(\mathsf{sRFE.crs}, \{(\mathsf{sRFE.pk}_i^*, C_{\mathsf{reg}}[i, x_i])\}_{i\in[L]}\big)\\
\mathsf{kpABE.}(\mathsf{mpk},\mathsf{msk}) &\leftarrow \mathsf{kpABE.Setup}(1^\lambda, \boxed{1^\ell})\\
\mathsf{kpABE.}(\mathsf{ct}_{\mathsf{off}},\mathsf{st}) &\leftarrow \mathsf{kpABE.EncOff}(\mathsf{kpABE.mpk})\\
\mathsf{kpABE.sk}_{f^*} &\leftarrow \mathsf{kpABE.KeyGen}(\mathsf{kpABE.msk}, f^*)\\
\mathsf{sRFE.ct} &\leftarrow \mathsf{sRFE.Enc}\big(\mathsf{sRFE.mpk}, x_{\mathsf{pub}} = \varepsilon, x_{\mathsf{pri}} = (\mu_b^*, \mathsf{kpABE.st}, \mathsf{sd})\big)\,.
\end{aligned}
$$

- *Guess.* $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$. The outcome of the experiment is b' if $f^*(x_i) = 1$ for all $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$, where $\mathcal{I}_{\mathsf{cor}} = \{i \in [L] : \mathsf{pk}_i^* \in \mathcal{C}\}$ and $\mathcal{I}_{\mathsf{mal}} = \{i \in [L] : \mathcal{D}[\mathsf{pk}_i^*] = \bot\}$. Otherwise, the outcome is set to $\bot$.

We recall that sRFE is a sRFE scheme with two-stage encryption, i.e., sRFE.ct is consits of two components $\mathsf{sRFE.ct}_{\mathsf{off}}$ and $\mathsf{sRFE.ct}_{\mathsf{on}}$. Note that $\Delta^* := \mathsf{sRFE.ct}_{\mathsf{on}}$ is the only component of the challenge ciphertext ct which depends on the challenge bit b. Hence, it suffices to argue that $\Delta^* \approx_c \Delta^{\$} \xleftarrow{\$} \mathsf{sRFE}.\mathcal{CT}_{\mathsf{on}}$, where $\mathsf{sRFE}.\mathcal{CT}_{\mathsf{on}}$ denotes the ciphertext space of sRFE.EncOn. We do so by relying on the two-stage Sel-prCT security of sRFE with respect to the following sampler $\mathsf{Samp}_{\mathsf{sRFE}}(1^\lambda)$.

- *Setup.* Sample $\mathsf{sd}, \mathsf{coins}_{\mathcal{A}} \xleftarrow{\$} \{0,1\}^\lambda$ and launch $\mathcal{A}(1^\lambda; \mathsf{coins}_{\mathcal{A}})$ to obtain (f^*, μ_0^*, μ_1^*). Run

$$
\begin{aligned}
\mathsf{kpABE.}(\mathsf{mpk},\mathsf{msk}) &\leftarrow \mathsf{kpABE.Setup}(1^\lambda, \boxed{1^\ell})\\
\mathsf{kpABE.}(\mathsf{ct}_{\mathsf{off}},\mathsf{st}) &\leftarrow \mathsf{kpABE.EncOff}(\mathsf{kpABE.mpk})\,.
\end{aligned}
$$

  Output $(x^*_{\mathsf{pub}} = \varepsilon, x^*_{\mathsf{pri}} = (\mathsf{kpABE.st}, \mathsf{sd}, \mu_b^*))$ and receive sRFE.crs in return. Forward sRFE.crs to $\mathcal{A}$.
- *Query.* Upon $\mathcal{A}$ submitting a query $\mathsf{QGen}()$ or $\mathsf{QCor}(\mathsf{pk})$, $\mathsf{Samp}_{\mathsf{sRFE}}$ makes the same query to its own challenger and forwards the response to $\mathcal{A}$.
- *Challenge.* Upon $\mathcal{A}$ submitting $\{(\mathsf{pk}_i^*, x_i)\}_{i\in[L]}$, $\mathsf{Samp}_{\mathsf{sRFE}}$ runs

$$
\mathsf{kpABE.sk}_{f^*} \leftarrow \mathsf{kpABE.KeyGen}(\mathsf{kpABE.msk}, f^*)
$$

  and outputs $\mathsf{aux}_{\mathsf{sRFE}} = (\mathsf{coins}_{\mathcal{A}}, \{x_i\}_{i\in[L]}, \mathsf{kpABE.ct}_{\mathsf{off}}, \mathsf{kpABE.sk}_{f^*})$ and $\{(\mathsf{pk}_i^*, C_{\mathsf{reg}}[i, x_i])\}_{i\in[L]}$.

Invoking the security of sRFE with $\mathsf{Samp}_{\mathsf{sRFE}}$, we obtain that $\Delta^* \approx_c \Delta^{\$}$ if

$$
\mathbf{Exp}^{\mathsf{pre\text{-}real}}_{\mathsf{sRFE},\mathsf{Samp}_{\mathsf{sRFE}},\mathcal{A}_{\mathsf{pre}}} \approx_c \mathbf{Exp}^{\mathsf{pre\text{-}rand}}_{\mathsf{sRFE},\mathsf{Samp}_{\mathsf{sRFE}},\mathcal{A}_{\mathsf{pre}}}
$$

for every PPT adversary $\mathcal{A}_{\mathsf{pre}}$. To show this, we consider the following sequence of hybrids $(\mathsf{G}_0, \mathsf{G}_1, \mathsf{G}_2)$.

Game G_0**:** This is $\mathbf{Exp}^{\text{pre-real}}_{\mathsf{sRFE},\mathsf{Samp}_{\mathsf{sRFE}},\mathcal{A}}$. Specifically, when unrolling $\mathsf{Samp}_{\mathsf{sRFE}}$, this game works as follows:

- *Setup.* Sample $\mathsf{sd},\mathsf{coins}_{\mathcal{A}} \xleftarrow{\$} \{0,1\}^\lambda$ and launch $\mathcal{A}(1^\lambda;\mathsf{coins}_{\mathcal{A}})$ to obtain (f^*,μ_0^*,μ_1^*). Run

$$\mathsf{kpABE}.(\mathsf{mpk},\mathsf{msk}) \leftarrow \mathsf{kpABE}.\mathsf{Setup}(1^\lambda,\boxed{1^\ell})$$
$$\mathsf{kpABE}.(\mathsf{ct}_{\mathsf{off}},\mathsf{st}) \leftarrow \mathsf{kpABE}.\mathsf{EncOff}(\mathsf{kpABE}.\mathsf{mpk})$$

  and set $(x^*_{\mathsf{pub}} = \varepsilon, x^*_{\mathsf{pri}} = (\mathsf{kpABE}.\mathsf{st},\mathsf{sd},\mu_b^*))$. Send $\mathsf{sRFE}.\mathsf{crs} \leftarrow \mathsf{sRFE}.\mathsf{Setup}(1^\lambda)$ to $\mathcal{A}$.
- *Query.* Repeat the following for arbitrarily many rounds determined by $\mathcal{A}$. In each round, $\mathcal{A}$ has two options.
  - $\mathsf{QGen}()$: run $\mathsf{sRFE}.(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{sRFE}.\mathsf{Gen}(\mathsf{sRFE}.\mathsf{crs},i)$ and return $\mathsf{pk} := \mathsf{sRFE}.\mathsf{pk}$ to $\mathcal{A}$. Set $\mathcal{D}[\mathsf{pk}] := \mathsf{sRFE}.\mathsf{sk}$.
  - $\mathsf{QCor}(\mathsf{pk})$: upon $\mathcal{A}$ submitting a public key pk, return $\mathcal{D}[\mathsf{pk}]$ to $\mathcal{A}$. Add pk to $\mathcal{C}$.
- *Challenge.* Upon $\mathcal{A}$ submitting $\{(\mathsf{pk}_i^*,x_i)\}_{i\in[L]}$, run

$$\mathsf{kpABE}.\mathsf{sk}_{f^*} \leftarrow \mathsf{kpABE}.\mathsf{KeyGen}(\mathsf{kpABE}.\mathsf{msk},f^*)\,,$$

  set $\mathsf{aux}_{\mathsf{sRFE}} = (\mathsf{coins}_{\mathcal{A}},\{x_i\}_{i\in[L]},\mathsf{kpABE}.\mathsf{ct}_{\mathsf{off}},\mathsf{kpABE}.\mathsf{sk}_{f^*})$ and define

$$\mathsf{rec}_{\mathsf{pre}} := \left(\mathsf{aux}_{\mathsf{sRFE}},x^*_{\mathsf{pub}},\{C_{\mathsf{reg}}[i,x_i]\}_{i\in[L]},\{\delta_i^*\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right),$$

  where we recall that

$$\{\delta_i^* := C_{\mathsf{reg}}[i,x_i](x^*_{\mathsf{pub}},x^*_{\mathsf{pri}}) = \mathsf{kpABE}.\mathsf{EncOn}(\mathsf{kpABE}.\mathsf{st},x_i,\mu_b^*;\mathsf{PRF}(\mathsf{sd},i))\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\,.$$

- *Guess.* Run the adversary $\mathcal{A}_{\mathsf{pre}}$ on input $(1^\lambda,\mathsf{rec}_{\mathsf{pre}})$ whose output $b'\in\{0,1\}$ is also the outcome of the experiment.

Game G_1**:** This is the same as G_0 except that the challenger computes

$$\{\delta_i^* := C_{\mathsf{reg}}[i,x_i](x^*_{\mathsf{pub}},x^*_{\mathsf{pri}}) = \mathsf{kpABE}.\mathsf{EncOn}(\mathsf{kpABE}.\mathsf{st},x_i,\mu_b^*)\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\,,$$

i.e., the $\mathsf{kpABE}.\mathsf{EncOn}$ is run with uniform random coins. As sd is not used anywhere else, we have $\mathsf{G}_0 \approx_c \mathsf{G}_1$ from the security of PRF.

Game G_2**:** This is the same as G_1 except that the challenger replaces the set $\{\delta_i^*\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}$ of function values with random values $\{\delta_i^{\$} \xleftarrow{\$} \mathsf{sRFE}.\mathcal{CT}_{\mathsf{on}}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}$. We note that $\mathsf{G}_2 = \mathbf{Exp}^{\text{pre-rand}}_{\mathsf{sRFE},\mathsf{Samp}_{\mathsf{sRFE}},\mathcal{A}}$. To prove $\mathsf{G}_1 \approx_c \mathsf{G}_2$, we argue that an adversary $\mathcal{A}_{\mathsf{pre}}$ who can distinguish G_1 and G_2 can be used to break the VerSel-INDr reusable security of kpABE. The reduction $\mathcal{B}$ works as follows:

- *Setup.* $\mathcal{B}$ samples $\mathsf{sd},\mathsf{coins}_{\mathcal{A}} \xleftarrow{\$} \{0,1\}^\lambda$, launches $\mathcal{A}(1^\lambda;\mathsf{coins}_{\mathcal{A}})$ to obtain the challenge (f^*,μ_0^*,μ_1^*) and sends $\mathsf{sRFE}.\mathsf{crs} \leftarrow \mathsf{sRFE}.\mathsf{Setup}(1^\lambda)$ to $\mathcal{A}$.
- *Query.* Repeat the following for arbitrarily many rounds determined by $\mathcal{A}$. In each round, $\mathcal{A}$ has two options.
  - $\mathsf{QGen}()$: run $\mathsf{sRFE}.(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{sRFE}.\mathsf{Gen}(\mathsf{sRFE}.\mathsf{crs},i)$ and return $\mathsf{pk} := \mathsf{sRFE}.\mathsf{pk}$ to $\mathcal{A}$.
  - $\mathsf{QCor}(\mathsf{pk})$: upon $\mathcal{A}$ submitting a public key pk, return $\mathcal{D}[\mathsf{pk}]$ to $\mathcal{A}$.
- *Challenge.* Upon $\mathcal{A}$ submitting $\{(\mathsf{pk}_i^*,x_i)\}_{i\in[L]}$, $\mathcal{B}$ sends $(\mathsf{coins}_{\mathcal{A}},\{f^*\},\{x_i\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}},\mu_b^*)$ to the kpABE challenger which returns

$$\left(\{\mathsf{kpABE}.\mathsf{sk}_{f^*}\},\mathsf{kpABE}.\mathsf{ct}_{\mathsf{off}},\{\mathsf{kpABE}.\mathsf{ct}_{\mathsf{on},i}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}}\right).$$

  $\mathcal{B}$ constructs the record

$$\mathsf{rec}_{\mathsf{pre}} := \begin{pmatrix} \mathsf{aux}_{\mathsf{sRFE}} = (\mathsf{coins}_{\mathcal{A}},\{x_i\}_{i\in[L]},\mathsf{kpABE}.\mathsf{ct}_{\mathsf{off}},\mathsf{kpABE}.\mathsf{sk}_{f^*}), \\ x^*_{\mathsf{pub}},\{C_{\mathsf{reg}}[i,x_i]\}_{i\in[L]},\{\mathsf{kpABE}.\mathsf{ct}_{\mathsf{on},i}\}_{i\in\mathcal{I}_{\mathsf{cor}}\cup\mathcal{I}_{\mathsf{mal}}} \end{pmatrix},$$

- *Guess.* $\mathcal{B}$ runs $b' \leftarrow \mathcal{A}_{\mathsf{pre}}(1^\lambda, \mathsf{rec}_{\mathsf{pre}})$ and forwards b' to the kpABE challenger.

Note that if the kpABE challenger provides real ciphertexts $\mathsf{kpABE.ct}_{\mathsf{on},i} \leftarrow \mathsf{kpABE.EncOn}(\mathsf{kpABE.st}, x_i, \mu_b^*)$, then $\mathcal{B}$ simulates G_1. If the kpABE challenger samples $\mathsf{kpABE.ct}_{\mathsf{on},i} \xleftarrow{\$} \mathsf{kpABE}.\mathcal{CT}_{\mathsf{on}}$, then $\mathcal{B}$ simulates G_2. Furthermore, by the admissibility of the adversary $\mathcal{A}$ in the game $\mathbf{Exp}^{\mathsf{srabe}\text{-}b}_{\mathsf{sRABE},\mathcal{A}}$, we have that $f^*(x_i) = 1$ for all $i \in \mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$. Thus the query sent by $\mathcal{B}$ to the kpABE challenger is also admissible.

□

6 Results in the Registration-Based Setting

In this section, we state our results in the (non-slotted) RFE and RABE setting. For RABE, we use the same definitions as previous works (see Section 3.12). For RFE, we introduce the new notion of pseudorandom ciphertext security which we present next.

Definition of RFE. We provide our notion of RFE with Sel-prCT security. Let $\{\mathcal{X}_{\lambda,\mathsf{pub}}\}_{\lambda\in\mathbb{N}}$, $\{\mathcal{X}_{\lambda,\mathsf{pri}}\}_{\lambda\in\mathbb{N}}$ and $\{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be sequences of public input spaces, private input spaces and output spaces, respectively. We consider a functionality $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ where each $\mathcal{F}_\lambda$ contains functions $f_\lambda : \mathcal{X}_{\lambda,\mathsf{pub}} \times \mathcal{X}_{\lambda,\mathsf{pri}} \to \mathcal{Y}_\lambda$.

Definition 6.1 (Syntax of RFE)**.** A RFE scheme for the functionality $\mathcal{F}$ consists of six efficient algorithms:

$\mathsf{Setup}(1^\lambda, \mathsf{param}) \to \mathsf{crs}$**:** On input the security parameter 1^λ and some parameter param specifying $\mathcal{F}$, this algorithm outputs a common reference string crs.

$\mathsf{Gen}(\mathsf{crs}, \mathsf{aux}) \to (\mathsf{pk}_i, \mathsf{sk}_i)$**:** On input the crs and a state aux, this algorithm outputs a pair of a public and a secret key $(\mathsf{pk}_i, \mathsf{sk}_i)$.

$\mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, f) \to (\mathsf{mpk}, \mathsf{aux}')$**:** On input the crs, a state aux, a public key pk and a function $f \in \mathcal{F}_\lambda$, this algorithm outputs a master public key mpk and an updated state aux'. We require Reg to be deterministic.

$\mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}) \to \mathsf{ct}$**:** On input the master public key mpk, a public input $x_{\mathsf{pub}} \in \mathcal{X}_{\lambda,\mathsf{pub}}$ and a private input $x_{\mathsf{pri}} \in \mathcal{X}_{\lambda,\mathsf{pri}}$, this algorithm outputs a ciphertext ct.

$\mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}) \to \mathsf{hsk}$**:** On input the crs, a state aux and a public key pk, this algorithm outputs a helper secret key hsk. We require Update to be deterministic.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct}) \to y \vee \perp \vee \mathsf{GetUpdate}$**:** On input a secret key sk, a helper secret key hsk and a ciphertext ct, this algorithm either outputs a value $y \in \mathcal{Y}_\lambda$, or a special symbol $\perp$ indicating decryption failure, or a special message GetUpdate indicating an updated helper secret key is needed to decrypt the ciphertext. We require Dec to be deterministic.

We recall the definitions of correctness, compactness and update efficiency.

Definition 6.2 (Correctness, Compactness and Update Efficiency of RFE)**.** Given a RFE scheme RFE and an (unbounded) adversary $\mathcal{A}$, we define the experiment $\mathbf{Exp}_{\mathsf{RFE},\mathcal{A}}$ as follows:

- *Setup.* Launch $\mathcal{A}(1^\lambda)$ and receive param from it. Run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param})$ and send crs to $\mathcal{A}$. Initialize the auxiliary input $\mathsf{aux} := \perp$, two empty dictionaries $\mathcal{E}, \mathcal{R}$ and counters $i_{\mathsf{reg}}, i^*_{\mathsf{reg}}, i_{\mathsf{enc}} := 1$ to keep track of QRegNT, QRegT and QEnc queries. Let $b = 0$ and $f^*, \mathsf{pk}^*, \mathsf{sk}^*, \mathsf{hsk}^* := \perp$.
- *Query.* Repeat the following for arbitrarily many rounds determined by $\mathcal{A}$. The oracle QRegT can be queried exactly once. In each round, $\mathcal{A}$ has four options.
  - $\mathsf{QRegNT}(\mathsf{pk}, f)$: upon $\mathcal{A}$ submitting a public key pk and a function $f \in \mathcal{F}_\lambda$, run $(\mathsf{mpk}, \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, f)$ and return $(i_{\mathsf{reg}}, \mathsf{mpk}, \mathsf{aux}')$ to $\mathcal{A}$. Update $\mathcal{R}[i_{\mathsf{reg}}] := (\mathsf{mpk}, \mathsf{aux}')$, $\mathsf{aux} := \mathsf{aux}'$ and $i_{\mathsf{reg}} := i_{\mathsf{reg}} + 1$.

– $\mathsf{QRegT}(f)$: upon $\mathcal{A}$ submitting a function $f \in \mathcal{F}_\lambda$, run $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{crs},\mathsf{aux})$, $(\mathsf{mpk},\mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, f)$, $\mathsf{hsk} := \mathsf{Update}(\mathsf{crs},\mathsf{aux}',\mathsf{pk})$ and return $(i_{\mathsf{reg}},\mathsf{mpk},\mathsf{aux}',\mathsf{pk},\mathsf{sk},\mathsf{hsk})$ to $\mathcal{A}$. Update $\mathcal{R}[i_{\mathsf{reg}}] := (\mathsf{mpk},\mathsf{aux}')$, $\mathsf{aux} := \mathsf{aux}'$, $i_{\mathsf{reg}} := i_{\mathsf{reg}}+1$ and $i^*_{\mathsf{reg}} := i_{\mathsf{reg}}$. Furthermore, set $f^* := f$, $\mathsf{pk} := \mathsf{pk}^*$, $\mathsf{sk} := \mathsf{sk}^*$ and $\mathsf{hsk} := \mathsf{hsk}^*$,

– $\mathsf{QEnc}(j, x_{\mathsf{pub}}, x_{\mathsf{pri}})$: upon $\mathcal{A}$ submitting an index $j \in [i^*_{\mathsf{reg}}; i_{\mathsf{reg}}]$ and a message $(x_{\mathsf{pub}}, x_{\mathsf{pri}}) \in \mathcal{X}_{\lambda,\mathsf{pub}}$, retrieve $(\mathsf{mpk},\star) := \mathcal{R}[j]$, run $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}})$ and return $(i_{\mathsf{enc}},\mathsf{ct})$ to $\mathcal{A}$. Set $\mathcal{E}[i_{\mathsf{enc}}] := (x_{\mathsf{pub}}, x_{\mathsf{pri}}, \mathsf{ct})$ and $i_{\mathsf{enc}} := i_{\mathsf{enc}}+1$.

– $\mathsf{QDec}(j)$: upon $\mathcal{A}$ submitting an index $j \in [i_{\mathsf{enc}}]$, if $\mathsf{sk}^* = \bot$ return $\bot$. Otherwise, retrieve the tuple $(x_{\mathsf{pub}}, x_{\mathsf{pri}}, \mathsf{ct}) := \mathcal{E}[j]$ and run $y \leftarrow \mathsf{Dec}(\mathsf{sk}^*, \mathsf{hsk}^*, \mathsf{ct})$. If $y = \mathsf{GetUpdate}$, then run $\mathsf{hsk}^* \leftarrow \mathsf{Update}(\mathsf{crs},\mathsf{aux},\mathsf{pk}^*)$ and recompute $y \leftarrow \mathsf{Dec}(\mathsf{sk}^*, \mathsf{hsk}^*, \mathsf{ct})$. Set $b = 1$ if $y \neq f^*(x_{\mathsf{pub}}, x_{\mathsf{pri}})$.

We say that FE is

- correct if $\Pr[\mathbf{Exp}_{\mathsf{RFE},\mathcal{A}}(1^\lambda) \to 0] = 1$ for all adversaries $\mathcal{A}$,
- compact if $|\mathsf{mpk}| = \mathsf{poly}(\lambda, \log L)$ and $|\mathsf{hsk}| = \mathsf{poly}(\lambda, \log L)$ at any stage during the execution of the experiment $\mathbf{Exp}_{\mathsf{RFE},\mathcal{A}}(1^\lambda)$, and
- update efficient if the oracle QDec invokes Update at most $O(\log|\mathcal{R}|)$ times and each invocation runs in time $\mathsf{poly}(\log|\mathcal{R}|)$.

We define our pseudorandom ciphertext security notion. At a high level, we require ciphertexts to be pseudorandom so long as the output of the functionality itself is pseudorandom.

Definition 6.3 (Sel-prCT Security for RFE)**.** Given a RFE scheme RFE with ciphertext space $\mathcal{CT} = \{\mathcal{CT}_\lambda\}_{\lambda\in\mathbb{N}}$, an interactive PPT algorithm Samp and a PPT adversary $\mathcal{A}$, we define $\mathbf{Exp}^{\mathsf{xxx\text{-}yyy}}_{\mathsf{RFE},\mathsf{Samp},\mathcal{A}}$, for $\mathsf{xxx} \in \{\boxed{\mathsf{pre}}, \boxed{\mathsf{post}}\}$ and $\mathsf{yyy} \in \{\mathsf{real},\mathsf{rand}\}$, as follows.

- *Setup.* Launch $\mathsf{Samp}(1^\lambda)$ and receive from it param and a challenge message $(x^*_{\mathsf{pub}}, x^*_{\mathsf{pri}}) \in \mathcal{X}_{\lambda,\mathsf{pub}} \times \mathcal{X}_{\lambda,\mathsf{pri}}$. Run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param})$ and send crs to Samp. Initialize the auxiliary input $\mathsf{aux} := \bot$, the master public key $\mathsf{mpk} := \bot$, a counter $i_{\mathsf{reg}} := 0$ to keep track of QRegHK queries, an empty set $\mathcal{C} := \varnothing$ and empty dictionaries $\mathcal{D}, \mathcal{R}$, and a transcript $\boxed{\mathsf{rec}_{\mathsf{pre}} := x^*_{\mathsf{pub}}}$ $\boxed{\mathsf{rec}_{\mathsf{post}} := (x^*_{\mathsf{pub}}, \mathsf{crs})}$.
- *Query.* Repeat the following for arbitrarily many rounds determined by Samp. In each round, Samp has three options.
  – $\mathsf{QRegCK}(\mathsf{pk}, f)$: upon Samp submitting a public key pk and a function $f \in \mathcal{F}_\lambda$, run $(\mathsf{mpk}',\mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs},\mathsf{aux},\mathsf{pk},f)$ and return $(\mathsf{mpk}',\mathsf{aux}')$ to Samp. Set $\mathsf{mpk} := \mathsf{mpk}'$, $\mathsf{aux} := \mathsf{aux}'$, $\mathcal{D}[\mathsf{pk}] := \mathcal{D}[\mathsf{pk}] \cup \{f\}$. Further, add pk to $\mathcal{C}$, and set $\boxed{\mathsf{rec}_{\mathsf{pre}} := \mathsf{rec}_{\mathsf{pre}} \,\|\, \mathcal{D}[\mathsf{pk}]}$, $\boxed{\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \,\|\, (\mathsf{pk}, \mathcal{D}[\mathsf{pk}])}$.
  – $\mathsf{QRegHK}(f)$: upon Samp submitting a function $f \in \mathcal{F}_\lambda$, run $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{crs},\mathsf{aux})$, $(\mathsf{mpk}',\mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs},\mathsf{aux},\mathsf{pk},f)$ and return $(i_{\mathsf{reg}},\mathsf{mpk}',\mathsf{aux}',\mathsf{pk})$ to Samp. Set $\mathsf{mpk} := \mathsf{mpk}'$, $\mathsf{aux} := \mathsf{aux}'$, $\mathcal{D}[\mathsf{pk}] := \mathcal{D}[\mathsf{pk}] \cup \{f\}$, $\mathcal{R}[i_{\mathsf{reg}}] := (\mathsf{pk},\mathsf{sk})$, $i_{\mathsf{reg}} := i_{\mathsf{reg}}+1$, $\boxed{\mathsf{rec}_{\mathsf{pre}} := \mathsf{rec}_{\mathsf{pre}} \,\|\, \mathcal{D}[\mathsf{pk}]}$, $\boxed{\mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \,\|\, (\mathsf{pk}, \mathcal{D}[\mathsf{pk}])}$.
  – $\mathsf{QCorHK}(j)$: upon Samp submitting an index $j \in [i_{\mathsf{reg}}]$, retrieve $(\mathsf{pk},\mathsf{sk}) := \mathcal{R}[j]$ and send sk to Samp. Further, add pk to $\mathcal{C}$ $\boxed{\text{and set } \mathsf{rec}_{\mathsf{post}} := \mathsf{rec}_{\mathsf{post}} \,\|\, (\mathsf{pk},\mathsf{sk})}$.
- *Challenge.* Compute

$$\{\delta^*_f := f(x^*_{\mathsf{pub}}, x^*_{\mathsf{pri}})\}_{f\in\bigcup_{\mathsf{pk}\in\mathcal{C}}\mathcal{D}[\mathsf{pk}]}, \qquad \{\delta^{\$}_f \xleftarrow{\$} \mathcal{Y}_\lambda\}_{f\in\bigcup_{\mathsf{pk}\in\mathcal{C}}\mathcal{D}[\mathsf{pk}]},$$
$$\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x^*_{\mathsf{pub}}, x^*_{\mathsf{pri}}), \qquad \mathsf{ct}^{\$} \xleftarrow{\$} \mathcal{CT}_\lambda,$$

$$\begin{aligned}
&\text{in } \mathbf{Exp}^{\text{pre-real}}_{\mathsf{RFE},\mathsf{Samp},\mathcal{A}}\colon && \mathsf{rec}_{\mathsf{pre}} \coloneqq \mathsf{rec}_{\mathsf{pre}} \parallel \left(\mathsf{aux}, \{\delta^{*}_{f}\}_{f\in\bigcup_{\mathsf{pk}\in\mathcal{C}}\mathcal{D}[\mathsf{pk}]}\right),\\
&\text{in } \mathbf{Exp}^{\text{pre-rand}}_{\mathsf{RFE},\mathsf{Samp},\mathcal{A}}\colon && \mathsf{rec}_{\mathsf{pre}} \coloneqq \mathsf{rec}_{\mathsf{pre}} \parallel \left(\mathsf{aux}, \{\delta^{\$}_{f}\}_{f\in\bigcup_{\mathsf{pk}\in\mathcal{C}}\mathcal{D}[\mathsf{pk}]}\right),\\
&\text{in } \mathbf{Exp}^{\text{post-real}}_{\mathsf{RFE},\mathsf{Samp},\mathcal{A}}\colon && \mathsf{rec}_{\mathsf{post}} \coloneqq \mathsf{rec}_{\mathsf{post}} \parallel \left(\mathsf{aux}, \mathsf{ct}^{*}\right),\\
&\text{in } \mathbf{Exp}^{\text{post-rand}}_{\mathsf{RFE},\mathsf{Samp},\mathcal{A}}\colon && \mathsf{rec}_{\mathsf{post}} \coloneqq \mathsf{rec}_{\mathsf{post}} \parallel \left(\mathsf{aux}, \mathsf{ct}^{\$}\right).
\end{aligned}$$

- *Guess.* Run the adversary $\mathcal{A}$ on input $(1^\lambda, \boxed{\mathsf{rec}_{\mathsf{pre}}}\ \boxed{\mathsf{rec}_{\mathsf{post}}})$ whose output $b \in \{0,1\}$ is also the outcome of the experiment.

For $\mathsf{xxx} \in \{\mathsf{pre}, \mathsf{post}\}$, we define the advantage function

$$\mathbf{Adv}^{\mathsf{xxx}}_{\mathsf{RFE},\mathsf{Samp},\mathcal{A}}(\lambda) \coloneqq \left|\Pr\left[\mathbf{Exp}^{\mathsf{xxx\text{-}real}}_{\mathsf{RFE},\mathsf{Samp},\mathcal{A}}(1^\lambda) \to 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{xxx\text{-}rand}}_{\mathsf{RFE},\mathsf{Samp},\mathcal{A}}(1^\lambda) \to 1\right]\right|.$$

The RFE scheme RFE is said to be $\mathsf{Sel\text{-}prCT}$ secure if for every PPT adversary $\mathcal{A}_{\mathsf{post}}$, there exists another PPT adversary $\mathcal{A}_{\mathsf{pre}}$ such that

$$\mathbf{Adv}^{\mathsf{pre}}_{\mathsf{RFE},\mathsf{Samp},\mathcal{A}_{\mathsf{pre}}}(\lambda) \geq \mathbf{Adv}^{\mathsf{post}}_{\mathsf{RFE},\mathsf{Samp},\mathcal{A}_{\mathsf{post}}}(\lambda)/\mathsf{poly}(\lambda) - \mathsf{negl}(\lambda)$$

and $\mathsf{Time}(\mathcal{A}_{\mathsf{pre}}) \leq \mathsf{Time}(\mathcal{A}_{\mathsf{post}}) \cdot \mathsf{poly}(\lambda)$.

We will make use of the following lemma to convert our sRFE schemes to RFE.

Lemma 6.4. *If there exists a* $\mathsf{Sel\text{-}prCT}$*-secure sRFE scheme for a functionality* $\mathcal{F}$*, then there exists a* $\mathsf{Sel\text{-}prCT}$*-secure RFE scheme for the same functionality* $\mathcal{F}$*.*

As recalled in Fact 3.36, Hohenberger et al. [HLWW23] introduced a transformation from slotted registered ABE to registered ABE, which is also being used by many subsequent works [FFM+23, DPY24, ZLZ+24] to lift a slotted RFE to RFE. At a high level, the transformation relies on a simple powers-of-two encoding strategy. Specifically, to support a system with $L = 2^\ell$ users, we employ $(\ell+1)$ instances of sRFE to construct a RFE scheme. We can use the same transformation to upgrade $\mathsf{Sel\text{-}prCT}$-secure sRFE into a full-fledged $\mathsf{Sel\text{-}prCT}$-secure RFE, hence we ignore the formal proof of the above theorem.

Results in the Registered Setting. We now use Fact 3.36 and Lemma 6.4 to upgrade our slotted schemes to full-fledged RFE and RABE. First, combining Theorem 4.17 and Lemma 6.4, we get the following theorem.

Theorem 6.5. *Assuming LWE and prFE, there exist* $\mathsf{Sel\text{-}prCT}$ *secure RFE schemes supporting the function classes* $\mathcal{C}_{\ell_{\mathsf{pub}},\ell_{\mathsf{pri}},\ell_{\mathsf{out}}}$ *and* $\mathcal{T}_{\ell_{\mathsf{out}}}$ *with the following parameters:*

$$\begin{aligned}
|\mathsf{crs}| &= \log L \cdot \mathsf{poly}(\lambda) & |\mathsf{mpk}| &= \log L \cdot \mathsf{poly}(\lambda)\\
|\mathsf{hsk}_i| &= (\log L)^2 \cdot \ell_{\mathsf{out}} \cdot \mathsf{poly}(\lambda) & |\mathsf{ct}| &= (\log L)^2 \cdot \mathsf{poly}(\lambda) + \ell_{\mathsf{pri}},
\end{aligned}$$

where ℓ_{pri} *denotes the length of the private message encrypted in* ct*. (Note that this length is not bounded at the time of setup in the case of Turing machines which is the reason why* ℓ_{pri} *does not appear as an index of the function class* $\mathcal{T}_{\ell_{\mathsf{out}}}$*).*

Furthermore, a combination of Theorem 5.7, Theorem 5.11 and Fact 3.36 yields the following.

Theorem 6.6. *Assuming LWE and prFE, there exist* $\mathsf{Sel\text{-}IND}$ *secure KP-RABE, CP-RABE and RPE schemes supporting the policy classes* $\mathcal{C}_\ell$ *and* $\mathcal{T}$ *with the following parameters:*

$$\begin{aligned}
|\mathsf{crs}| &= \log L \cdot \mathsf{poly}(\lambda) & |\mathsf{mpk}| &= \log L \cdot \mathsf{poly}(\lambda)\\
|\mathsf{hsk}_i| &= (\log L)^2 \cdot \mathsf{poly}(\lambda) & |\mathsf{ct}| &= (\log L)^2 \cdot \mathsf{poly}(\lambda) + \ell_{\mathsf{in}} + \ell_{\mathsf{att}}
\end{aligned}$$

where the message and attributes have lengths ℓ_{in} *and* ℓ_{att} *respectively.*

7 prCT Secure FE for Turing Machines and Applications to ABE

In this section, we construct VerSel-prCT secure FE for Turing machines and demonstrate how it can be used to build KP-ABE and CP-ABE for Turing machines. Our constructions enjoy optimal asymptotic parameters, i.e., master public keys and secret keys are of size $\mathsf{poly}(\lambda)$ and ciphertexts encrypting private inputs of length ℓ_{pri} have size $\mathsf{poly}(\lambda)+\ell_{\mathsf{pri}}$.

7.1 Construction of prCT Secure FE for TMs

In this section, we construct a VerSel-prCT secure FE scheme for the class $\mathcal{T}_{\ell_{\mathsf{pri}}}$ of Turing machines with public input space $\mathcal{X}_{\mathsf{pub}}=\{0,1\}^*$ and private input space $\mathcal{X}_{\mathsf{pri}}=\{0,1\}^{\ell_{\mathsf{pri}}}$.

Construction 7.1 (Reusable VerSel-prCT Secure FE for TMs)**.** The construction uses the following ingredients:

- A blind garbling scheme $\mathsf{bGC}=\mathsf{bGC}.(\mathsf{Garble},\mathsf{Eval},\mathsf{Sim})$ with decomposability (Definition 3.16). We assume that the labels and the random coins used by $\{\mathsf{bGC}.\mathsf{Garble}_k\}_k$ and $\{\mathsf{bGC}.\mathsf{Garble}_{\mathsf{inp},k}\}_k$ are in $\{0,1\}^\lambda$. The former is guaranteed by Definition 3.12 and the latter can be achieved without loss of generality by using a PRF to derive longer (pseudo-)random coins if needed. We can instantiate bGC with the required properties assuming one-way functions (Fact 3.17).
- A pseudorandom function $\mathsf{PRF}\colon \{0,1\}^\lambda\times\{0,1\}^\lambda\to\{0,1\}^\lambda$ with key space, input space and output space being $\{0,1\}^\lambda$. PRF can be instantiated assuming one-way functions.
- A laconic pPRIO scheme $\mathsf{LprIO}=\mathsf{LprIO}.(\mathsf{Digest},\mathsf{ObfOff},\mathsf{ObfOn},\mathsf{Eval})$. Without loss of generality, the state output by $\mathsf{LprIO}.\mathsf{ObfOff}$ and the random coins of $\mathsf{LprIO}.\mathsf{ObfOn}$ are in $\{0,1\}^\lambda$. To achieve the former, we let the state be the random coins used by $\mathsf{LprIO}.\mathsf{ObfOff}$ which can in turn be replaced with a string in $\{0,1\}^\lambda$ using a PRF. For the latter, we can also use a PRF. We note that Definition 3.43 guarantees that the length of digests is bounded by a fixed polynomial $\ell_{\mathsf{dig}}=\ell_{\mathsf{dig}}(\lambda)$ in the security parameter. We can instantiate LprIO with the desired properties assuming LWE and pPRIO (Fact 3.45).
- A FE scheme $\mathsf{FE}=\mathsf{FE}.(\mathsf{Setup},\mathsf{Gen},\mathsf{Agg},\mathsf{Enc},\mathsf{Dec})$ with trivial offline encryption (Definition 3.26) and reusable VerSel-prCT security (Definition 3.27) for the class $\mathcal{C}_{\ell'_{\mathsf{pri}},\ell'_{\mathsf{dep}},\ell'_{\mathsf{out}}}$ consisting of circuits with public input length $\ell'_{\mathsf{pub}}=0$, private input length $\ell'_{\mathsf{pri}}=\ell_{\mathsf{pri}}+\ell_{\mathsf{dig}}+9\lambda$, maximum depth ℓ'_{dep} and output length ℓ'_{out}, where we set ℓ'_{dep} and ℓ'_{out} so that the circuit class contains $C_{\mathsf{kgen}}[\mathbf{r},\mathsf{dig}_M]$ defined in Figure 14. We denote the information specifying the circuit class by $\mathsf{param}'=(1^{\ell'_{\mathsf{pri}}},1^{\ell'_{\mathsf{dep}}},1^{\ell'_{\mathsf{out}}})$. We can construct FE with the desired properties assuming LWE and pPRIO (Fact 3.28).

The details of the VerSel-prCT secure FE scheme for Turing machines are as follows.

$\mathsf{Setup}(1^\lambda,1^{\ell_{\mathsf{pri}}})$**:** Run $\mathsf{FE}.(\mathsf{mpk},\mathsf{msk})\leftarrow\mathsf{FE}.\mathsf{Setup}(1^\lambda,\mathsf{param}')$ and output $(\mathsf{mpk}:=\mathsf{FE}.\mathsf{mpk},\mathsf{msk}:=\mathsf{FE}.\mathsf{msk})$.

$\mathsf{KeyGen}(\mathsf{msk},M)$**:** Parse $\mathsf{msk}=\mathsf{FE}.\mathsf{msk}$, sample $\mathbf{r}\xleftarrow{\$}\{0,1\}^\lambda$ and compute

$$\begin{aligned}\mathsf{dig}_M&\leftarrow\mathsf{LprIO}.\mathsf{Digest}(\{(k,C_{\mathsf{tm}}^{(k)})\}_{k\in[|C_{\mathsf{tm}}|]})\\ \mathsf{FE}.\mathsf{sk}_M&\leftarrow\mathsf{FE}.\mathsf{KeyGen}(\mathsf{FE}.\mathsf{msk},C_{\mathsf{kgen}})\,,\end{aligned}$$

where $C_{\mathsf{tm}}=C_{\mathsf{tm}}[M]$ and $C_{\mathsf{kgen}}=C_{\mathsf{kgen}}[\mathbf{r},\mathsf{dig}_M]$ are defined in Figure 13 and Figure 14, respectively. Output $\mathsf{sk}_M:=(\mathbf{r},\mathsf{dig}_M,\mathsf{FE}.\mathsf{sk}_M)$.

$\mathsf{Enc}(\mathsf{mpk},x)$**:** The encryption algorithm proceeds in two steps.

$\mathsf{EncOff}(\mathsf{mpk})$**:** Parse $\mathsf{mpk}=\mathsf{FE}.\mathsf{mpk}$ and run

$$\begin{aligned}(\widehat{C}_{\mathsf{off}},\mathsf{LprIO}.\mathsf{st})&\leftarrow\mathsf{LprIO}.\mathsf{ObfOff}(1^\lambda,1^S)\\ (\widehat{C}'_{\mathsf{off}},\mathsf{LprIO}.\mathsf{st}')&\leftarrow\mathsf{LprIO}.\mathsf{ObfOff}(1^\lambda,1^S)\,,\end{aligned}$$

Input:
- two (fixed-length) bit strings $\mathsf{bits}(\ell), \mathsf{bits}(t) \in \{0,1\}^\lambda$
- a LprIO state LprIO.st
- random coins $R_{\mathsf{dig}}, R_{\mathsf{bgc}}, R_{\mathsf{cir}} \in \{0,1\}^\lambda$

Output: a LprIO online obfuscation $\widehat{C}_{\mathsf{on,cir}}$

Hardwired Values: a Turing machine M

- Run

$$\begin{aligned}\mathsf{dig}_T &\leftarrow \mathsf{LprIO.Digest}(\{(k, T^{(k)})\}_{k\in[|T|]}; R_{\mathsf{dig}})\\ \widehat{C}_{\mathsf{on,cir}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.st}, \mathsf{dig}_T, E_{\mathsf{cir}}[R_{\mathsf{bgc}}]; R_{\mathsf{cir}})\ ,\end{aligned}$$

  where $T = T_\ell[M, t]$ and E_{cir} are defined in Figure 8 and 5, respectively.

- Output $\widehat{C}_{\mathsf{on,cir}}$.

Figure 13: Definition of the circuit $C_{\mathsf{tm}}[M]$

Input:
- a bit string $\mathsf{bits}(t) \in \{0,1\}^\lambda$
- a LprIO digest $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}$; we implicitly assume that $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}$ contains the length $\ell_{\mathsf{pub}} := |\mathbf{x}_{\mathsf{pub}}|$
- a private input vector $\mathbf{x}_{\mathsf{pri}} \in \{0,1\}^{\ell_{\mathsf{pri}}}$
- two LprIO states LprIO.st and LprIO.st$'$
- PRF seeds $\mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}} \in \{0,1\}^\lambda$

Output:
- two sets of labels $\{\mathsf{lab}'_{k,\mathbf{y}[k]}\}_{k\in[6\lambda]}$ and $\{\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}$
- LprIO online obfuscations $\widehat{C}'_{\mathsf{on,cir}}$ and $\widehat{C}_{\mathsf{on,pub}}$

Hardwired Values:
- a string $\mathbf{r} \in \{0,1\}^\lambda$
- a LprIO digest dig_M

- Compute $R_{\mathsf{str}} = \mathsf{PRF}(\mathsf{sd}_{\mathsf{str}}, \mathbf{r})$ and $R'_{\mathsf{str}'} = \mathsf{PRF}(\mathsf{sd}'_{\mathsf{str}'}, \mathbf{r})$ for $\mathsf{str} \in \{\mathsf{dig}, \mathsf{pub}, \mathsf{cir}, \mathsf{bgc}\}$ and $\mathsf{str}' \in \{\mathsf{cir}, \mathsf{bgc}\}$.

- Define $\mathbf{y} = \big(\mathsf{bits}(\ell) \in \{0,1\}^\lambda, \mathsf{bits}(t), \mathsf{LprIO.st}, R_{\mathsf{dig}}, R_{\mathsf{bgc}}, R_{\mathsf{cir}}\big) \in \{0,1\}^{6\lambda}$, where $\ell = \ell_{\mathsf{pub}} + \ell_{\mathsf{pri}}$.

- Run

$$\begin{aligned}(\mathsf{lab}'_{k,0}, \mathsf{lab}'_{k,1}) &\leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R'_{\mathsf{bgc}}) \qquad \text{for } k \in [6\lambda]\\ (\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) &\leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R_{\mathsf{bgc}}) \qquad \text{for } k \in [\ell_{\mathsf{pub}}+1; \ell]\ .\end{aligned}$$

- Run

$$\begin{aligned}\widehat{C}'_{\mathsf{on,cir}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.st}', \mathsf{dig}_M, E_{\mathsf{cir}}[R'_{\mathsf{bgc}}]; R'_{\mathsf{cir}})\\ \widehat{C}_{\mathsf{on,pub}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.st}, \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, E_{\mathsf{pub}}[R_{\mathsf{bgc}}]; R_{\mathsf{pub}})\ ,\end{aligned}$$

  where E_{cir} and E_{pub} are defined in Figure 5 and 6.

- Output $(\{\mathsf{lab}'_{k,\mathbf{y}[k]}\}_{k\in[6\lambda]}, \{\mathsf{lab}_{k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}, \widehat{C}'_{\mathsf{on,cir}}, \widehat{C}_{\mathsf{on,pub}})$.

Figure 14: Definition of the circuit $C_{\mathsf{kgen}}[\mathbf{r}, \mathsf{dig}_M]$

where S is the maximum size of the circuits $E_{\mathsf{cir}}[R_{\mathsf{bgc}}]$ and $E_{\mathsf{pub}}[R_{\mathsf{bgc}}]$ defined in Figure 5 and 6. Output $\mathsf{ct}_{\mathsf{off}} := (\widehat{C}_{\mathsf{off}}, \widehat{C}'_{\mathsf{off}})$ and $\mathsf{st} = (\mathsf{LprIO.st}, \mathsf{LprIO.st}', \mathsf{FE.mpk})$.

$\mathsf{EncOn}(\mathsf{st}, x_{\mathsf{pub}}, x_{\mathsf{pri}})$**:** Parse $\mathsf{st} = (\mathsf{LprIO.st}, \mathsf{LprIO.st}', \mathsf{FE.mpk})$, $x_{\mathsf{pub}} = (1^t, \mathbf{x}_{\mathsf{pub}}) \in \{0,1\}^*$ and $x_{\mathsf{pri}} = \mathbf{x}_{\mathsf{pri}} \in \{0,1\}^{\ell_{\mathsf{pri}}}$. Sample PRF seeds $\mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}} \xleftarrow{\$} \{0,1\}^\lambda$ and run

$$\begin{aligned} \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} &\leftarrow \mathsf{LprIO.Digest}\big(1^\lambda, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}\big) \\ \mathsf{FE.ct} &\leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, \mathbf{x}'_{\mathsf{pri}}) \,, \end{aligned}$$

where

$$\mathbf{x}'_{\mathsf{pri}} = \big(\mathsf{bits}(t), \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, \mathbf{x}_{\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{LprIO.st}', \mathsf{sd}_{\mathsf{dig}}, \mathsf{sd}_{\mathsf{pub}}, \mathsf{sd}_{\mathsf{cir}}, \mathsf{sd}'_{\mathsf{cir}}, \mathsf{sd}_{\mathsf{bgc}}, \mathsf{sd}'_{\mathsf{bgc}}\big) \in \{0,1\}^{\ell'_{\mathsf{pri}}} \,.$$

Output $\mathsf{ct}_{\mathsf{on}} := \mathsf{FE.ct}$.

The final output of $\mathsf{Enc}(\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}})$ is $\mathsf{ct} = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}})$.

$\mathsf{Dec}(\mathsf{sk}_M, M, \mathsf{ct}, x_{\mathsf{pub}})$**:** Parse $\mathsf{sk}_M = (\mathbf{r}, \mathsf{dig}_M, \mathsf{FE.sk}_M)$, $\mathsf{ct} = (\widehat{C}_{\mathsf{off}}, \widehat{C}'_{\mathsf{off}}, \mathsf{FE.ct})$ and $x_{\mathsf{pub}} = (1^t, \mathbf{x}_{\mathsf{pub}})$. Run

$$\big(\{\mathsf{lab}'_k\}_{k\in[6\lambda]}, \{\mathsf{lab}_k\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}, \widehat{C}'_{\mathsf{on,cir}}, \widehat{C}_{\mathsf{on,pub}}\big) \leftarrow \mathsf{FE.Dec}\big(\mathsf{FE.sk}_M, C_{\mathsf{kgen}}[\mathbf{r}, \mathsf{dig}_M], \mathsf{FE.ct}\big) \,,$$

where $\ell_{\mathsf{pub}} = |\mathbf{x}_{\mathsf{pub}}|$, $\ell = \ell_{\mathsf{pub}} + \ell_{\mathsf{pri}}$ and $C_{\mathsf{kgen}}[\mathbf{r}, \mathsf{dig}_M]$ is defined in Figure 14. Compute

$$\begin{aligned} \widetilde{C}_{\mathsf{tm}} = \{\widetilde{C}^{(k)}_{\mathsf{tm}}\}_{k\in[|C_{\mathsf{tm}}|]} &\leftarrow \mathsf{LprIO.Eval}\big(\{(k, C^{(k)}_{\mathsf{tm}})\}_{k\in[|C_{\mathsf{tm}}|]}, \widehat{C}'_{\mathsf{cir}} = (\widehat{C}'_{\mathsf{off}}, \widehat{C}'_{\mathsf{on,cir}})\big) \\ \widehat{C}_{\mathsf{on,cir}} &\leftarrow \mathsf{bGC.Eval}\big(\widetilde{C}_{\mathsf{tm}}, \{\mathsf{lab}'_k\}_{k\in[6\lambda]}\big) \\ \widetilde{T} = \{\widetilde{T}^{(k)}\}_{k\in[|T|]} &\leftarrow \mathsf{LprIO.Eval}\big(\{(k, T^{(k)})\}_{k\in[|T|]}, \widehat{C}_{\mathsf{cir}} = (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,cir}})\big) \\ \{\mathsf{lab}_k\}_{k\in[\ell_{\mathsf{pub}}]} &\leftarrow \mathsf{LprIO.Eval}\big(\{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}, \widehat{C}_{\mathsf{pub}} = (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,pub}})\big) \\ z &\leftarrow \mathsf{bGC.Eval}(\widetilde{T}, \{\mathsf{lab}_k\}_{k\in[\ell]}) \,, \end{aligned}$$

where the circuits $C_{\mathsf{tm}} = C_{\mathsf{tm}}[M]$ and $T = T_\ell[M, t]$ are defined in Figure 13 and 8. Output z.

Proposition 7.2 (Correctness and Efficiency)**.** *The FE scheme for the class $\mathcal{T}_{\ell_{\mathsf{pri}}}$ in Construction 7.1 is correct and has the following parameters:*

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda, \ell_{\mathsf{pri}}) \,, \qquad |\mathsf{sk}_M| = \mathsf{poly}(\lambda, \ell_{\mathsf{pri}}) \,, \qquad |\mathsf{ct}| = \mathsf{poly}(\lambda, \ell_{\mathsf{pri}}) \,.$$

Proposition 7.3 (Security)**.** *If* bGC *is simulation secure (Definition 3.14) and blind (Definition 3.15),* PRF *is secure,* LprIO *is secure (Definiton 3.44) and* FE *satisfies reusable* VerSel-prCT *security (Definition 3.27), then the FE scheme in Construction 7.1 also satisfies reusable* VerSel-prCT *security.*

The proofs of Propositions 7.2 and 7.3 can be found in Sections 7.2 and 7.3, respectively. In Construction 4.14, we describe a generic conversion that upgrades a Sel-prCT secure sRFE scheme for the class $\mathcal{T}_{\ell_{\mathsf{pri}}}$ (i.e., Turing machines with bounded-length private inputs and one-bit outputs) to the class $\mathcal{T}_{\ell_{\mathsf{out}}}$ (i.e., unbounded-length private inputs and ℓ_{out}-bits outputs). In addition, this transformation improves the efficiency parameters by reducing the dependency on ℓ_{pri} to an information-theoretic minimum. The very same idea also works in the (plain) FE setting and leads us to the following theorem.

Theorem 7.4. *Assuming LWE and prFE, there exists a FE scheme for $\mathcal{T}_{\ell_{\mathsf{out}}}$ satisfying reusable* VerSel-prCT *security with the following efficiency parameters:*

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda) \,, \qquad |\mathsf{sk}_M| = \ell_{\mathsf{out}} \cdot \mathsf{poly}(\lambda) \,, \qquad |\mathsf{ct}| = \mathsf{poly}(\lambda) + \ell_{\mathsf{pri}} \,,$$

where ℓ_{pri} denotes the length of the private message encrypted in ct*. (But this length is not bounded at the time of setup which is the reason why ℓ_{pri} does not appear as an index of the function class $\mathcal{T}_{\ell_{\mathsf{out}}}$). In particular, if ℓ_{out} is a fixed polynomial in λ, then these parameters are asymptotically optimal.*

7.2 Proof of Correctness and Efficiency

Proof of Proposition 7.2. We argue that Construction 7.1 is correct and has parameters as stated.

Correctness. Let $\lambda \in \mathbb{N}$, $M \in \mathcal{T}_{\ell_{\mathsf{pri}}}$ and $(x_{\mathsf{pub}}, x_{\mathsf{pri}}) \in \mathcal{X}_{\mathsf{pub}} \times \mathcal{X}_{\mathsf{pri}}$, where $x_{\mathsf{pub}} = (1^t, \mathbf{x}_{\mathsf{pub}}) \in \{0,1\}^*$ and $x_{\mathsf{pri}} = \mathbf{x}_{\mathsf{pri}} \in \{0,1\}^{\ell_{\mathsf{pri}}}$. Define $\ell_{\mathsf{pub}} := |\mathbf{x}_{\mathsf{pub}}|$ and $\ell := \ell_{\mathsf{pub}} + \ell_{\mathsf{pri}}$. Furthermore, let

$$\begin{aligned}(\mathsf{mpk}, \mathsf{msk}) &\leftarrow \mathsf{Setup}(1^\lambda, \mathsf{param})\\ \mathsf{sk}_M &\leftarrow \mathsf{KeyGen}(\mathsf{msk}, M)\\ (\mathsf{ct}_{\mathsf{off}}, \mathsf{st}) &\leftarrow \mathsf{EncOff}(\mathsf{mpk})\\ \mathsf{ct}_{\mathsf{on}} &\leftarrow \mathsf{EncOn}(\mathsf{st}, x_{\mathsf{pub}}, x_{\mathsf{pri}})\ .\end{aligned}$$

$\mathsf{Dec}(\mathsf{sk}_M, M, \mathsf{ct}, x_{\mathsf{pub}})$ parses $\mathsf{sk}_M = (\mathbf{r}, \mathsf{dig}_M, \mathsf{FE.sk}_M)$, $\mathsf{ct} = (\widehat{C}_{\mathsf{off}}, \widehat{C}'_{\mathsf{off}}, \mathsf{FE.ct})$ and $x_{\mathsf{pub}} = (1^t, \mathbf{x}_{\mathsf{pub}})$. Then it starts by running FE decryption

$$\big(\{\mathsf{lab}'_k\}_{k\in[6\lambda]}, \{\mathsf{lab}_k\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}, \widehat{C}'_{\mathsf{on,cir}}, \widehat{C}_{\mathsf{on,pub}}\big) \leftarrow \mathsf{FE.Dec}\big(\mathsf{FE.sk}_M, C_{\mathsf{kgen}}[\mathbf{r}, \mathsf{dig}_M], \mathsf{FE.ct}\big)\ ,$$

where by the definition of $C_{\mathsf{kgen}}[\mathbf{r}, \mathsf{dig}_M]$, we have

$$\begin{aligned}\big\{(\mathsf{lab}'_{k,0}, \mathsf{lab}'_{k,1}) &\leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R'_{\mathsf{bgc}})\big\}_{k\in[6\lambda]}\\ \big\{(\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) &\leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R_{\mathsf{bgc}} = \mathsf{PRF}(\mathsf{sd}_{\mathsf{bgc}}, \mathbf{r}))\big\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}\\ \widehat{C}'_{\mathsf{on,cir}} &\leftarrow \mathsf{LprIO.ObfOn}\big(\mathsf{LprIO.st}, \mathsf{dig}_M, E_{\mathsf{cir}}[R'_{\mathsf{bgc}}]; R'_{\mathsf{cir}}\big)\\ \widehat{C}_{\mathsf{on,pub}} &\leftarrow \mathsf{LprIO.ObfOn}\big(\mathsf{LprIO.st}, \mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}}, E_{\mathsf{pub}}[R_{\mathsf{bgc}}]; R_{\mathsf{pub}}\big)\end{aligned}$$

and obtain the labels $\{\mathsf{lab}'_k := \mathsf{lab}'_{k,\mathbf{y}[k]}\}_{k\in[6\lambda]}$ and $\{\mathsf{lab}_k := \mathsf{lab}'_{k,\mathbf{x}_{\mathsf{pri}}[k-\ell_{\mathsf{pub}}]}\}_{k\in[\ell_{\mathsf{pub}}+1;\ell]}$ for

$$\mathbf{y} = \big(\mathsf{bits}(\ell), \mathsf{bits}(t), \mathsf{LprIO.st}, R_{\mathsf{dig}}, R_{\mathsf{bgc}}, R_{\mathsf{cir}}\big) \in \{0,1\}^{6\lambda}\ .$$

Next, the decryption algorithm evaluates the LprIO obfuscations. In the first step, we have

$$\widetilde{C}_{\mathsf{tm}} = \{\widetilde{C}_{\mathsf{tm}}{}^{,(k)}\}_{k\in[|C_{\mathsf{tm}}|]} \leftarrow \mathsf{LprIO.Eval}\big(\{(k, C^{(k)}_{\mathsf{tm}})\}_{k\in[|C_{\mathsf{tm}}|]}, \widehat{C}'_{\mathsf{cir}} = (\widehat{C}'_{\mathsf{off}}, \widehat{C}'_{\mathsf{on,cir}})\big)\ .$$

From the correctness of LprIO, it follows that

$$\widetilde{C}^{(k)}_{\mathsf{tm}} = E_{\mathsf{cir}}[R'_{\mathsf{bgc}}](k, C^{(k)}_{\mathsf{tm}}) \leftarrow \mathsf{bGC.Garble}_k(1^\lambda, C^{(k)}_{\mathsf{tm}}; R'_{\mathsf{bgc}})\ ,$$

where $C_{\mathsf{tm}} = C_{\mathsf{tm}}[M]$. A joint evaluation of $\widetilde{C}_{\mathsf{tm}}$ with $\{\mathsf{lab}'_k\}_{k\in[6\lambda]}$ obtained from the FE decryption gives

$$\widehat{C}_{\mathsf{on,cir}} = C_{\mathsf{tm}}(\mathbf{y}) \leftarrow \mathsf{bGC.Eval}\big(\widetilde{C}_{\mathsf{tm}}, \{\mathsf{lab}'_k\}_{k\in[6\lambda]}\big)$$

By the definition of C_{tm}, we have

$$\begin{aligned}\mathsf{dig}_T &\leftarrow \mathsf{LprIO.Digest}(\{(k, T^{(k)})\}_{k\in[|T|]}; R_{\mathsf{dig}}\mathsf{PRF}(\mathsf{sd}_{\mathsf{dig}}, \mathbf{r}))\\ \widehat{C}_{\mathsf{on,cir}} &\leftarrow \mathsf{LprIO.ObfOn}(\mathsf{LprIO.st}, \mathsf{dig}_T, E_{\mathsf{cir}}[R_{\mathsf{bgc}}]; R_{\mathsf{cir}})\ ,\end{aligned}$$

where $T = T_\ell[M, t]$ is a circuit that evaluates the Turing machine M for t steps on input a vector of length $\ell = \ell_{\mathsf{pub}} + \ell_{\mathsf{pri}}$. At this point, we have $(\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,cir}}, \widehat{C}_{\mathsf{on,pub}})$. Their evaluation gives

$$\begin{aligned}\widetilde{T} = \{\widetilde{T}^{(k)}\}_{k\in[|T|]} &\leftarrow \mathsf{LprIO.Eval}\big(\{(k, T^{(k)})\}_{k\in[|T|]}, \widehat{C}_{\mathsf{cir}} = (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,cir}})\big)\\ \{\mathsf{lab}_k\}_{k\in[\ell_{\mathsf{pub}}]} &\leftarrow \mathsf{LprIO.Eval}\big(\{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k\in[\ell_{\mathsf{pub}}]}, \widehat{C}_{\mathsf{pub}} = (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on,pub}})\big)\end{aligned}$$

and, again by the correctness of LprIO and the definitions of $E_{\mathsf{cir}}[R_{\mathsf{bgc}}]$ and $E_{\mathsf{pub}}[R_{\mathsf{bgc}}]$, we conclude that

$$\begin{aligned}\widetilde{T}^{(k)} &\leftarrow \mathsf{bGC.Garble}_k(1^\lambda, T^{(k)}; R_{\mathsf{bgc}}) = E_{\mathsf{cir}}[R_{\mathsf{bgc}}](k, T^{(k)})\\ (\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}) &\leftarrow \mathsf{bGC.Garble}_{\mathsf{inp},k}(1^\lambda; R_{\mathsf{bgc}})\end{aligned}$$

such that $\mathsf{lab}_k = \mathsf{lab}_{\mathbf{x}_{\mathsf{pub}}[k]}$. Putting everything together, bGC evaluation yields $z \leftarrow \mathsf{bGC.Eval}(\widetilde{T}, \{\mathsf{lab}_k\}_{k\in[\ell]})$ satisfying $z = T(x_{\mathsf{pub}}, x_{\mathsf{pri}}) = M(1^t, x_{\mathsf{pub}}, x_{\mathsf{pri}})$.

Efficiency. We start by analyzing the size of the circuit $C_{\mathsf{kgen}}[\mathbf{r}, \mathsf{dig}_M]$.

- The evaluations of PRF can be performed by circuits of size $\mathsf{poly}(\lambda)$.
- The $(6\lambda + \ell_{\mathsf{pri}})$ computations of $\mathsf{bGC.Garble}_{\mathsf{inp},k}$ can each be implemented by a circuit of size $\mathsf{poly}(\lambda)$, so the overall computation can be performed in size $\mathsf{poly}(\lambda, \ell_{\mathsf{pri}})$.
- To bound the size of the first execution of $\mathsf{LprIO.ObfOn}$, we observe that the overall input length is a fixed polynomial in λ since $|\mathsf{LprIO.st}| = \lambda$ and $|\mathsf{dig}_M| = \mathsf{poly}(\lambda)$ (guaranteed by Definition 3.43 and Theorem B.4) as well as $|E_{\mathsf{cir}}[R'_{\mathsf{bgc}}]| = \mathsf{poly}(\lambda)$ (argued in the proof of Proposition 4.9). Since $\mathsf{LprIO.ObfOn}$ is a poly-time algorithm, this implies that the overall size can be bounded by $\mathsf{poly}(\lambda)$.
- The size of a circuit performing the second execution of $\mathsf{LprIO.ObfOn}$ can be bounded similarly. First, we observe that the overall input length of $\mathsf{LprIO.ObfOn}$ is a fixed polynomial in λ. In particular, we recall from the proof of Proposition 4.9 that $|E_{\mathsf{pub}}[R_{\mathsf{bgc}}]| = \mathsf{poly}(\lambda)$. Thus, the size of the circuit computing $\mathsf{LprIO.ObfOn}$ is also $\mathsf{poly}(\lambda)$.

Thus, we can initialize FE with the parameters $\ell'_{\mathsf{pri}} = \ell'_{\mathsf{dep}} = \ell'_{\mathsf{out}} = \mathsf{poly}(\lambda, \ell_{\mathsf{pri}})$. Plugging this choice into the parameters of our instantiation of FE (see Fact 3.28) gives the parameters as stated. □

7.3 Proof of Security

Proof of Proposition 7.3. To avoid a clash of variables, we will use the following convention throughout this proof. The FE scheme for Turing machines which is built in Construction 7.1 is denoted by FE. On the other hand, the FE scheme for bounded depth circuits which serves as a building block and which was previously denoted by FE is now denoted by FE'. All variables (e.g., keys, inputs, etc.) belonging either to FE or FE' will be distinguished in the same way.

We consider a PPT sampler Samp that on input 1^λ outputs

$$\left(\mathsf{aux} \in \{0,1\}^*, \{M_i\}_{i\in[Q_{\mathsf{key}}]} \subseteq \mathcal{T}_{\ell_{\mathsf{pri}}}, \{x_j = (x_{j,\mathsf{pub}}, x_{j,\mathsf{pri}})\}_{j\in[Q_{\mathsf{msg}}]} \subseteq \{0,1\}^* \times \{0,1\}^{\ell_{\mathsf{pri}}}\right).$$

To prove reusable VerSel-prCT security as per Definition 3.27, we need to show that

$$\left(\begin{array}{c} \mathsf{aux}, \mathsf{mpk} := \mathsf{FE}'.\mathsf{mpk}, \mathsf{ct}_{\mathsf{off}} := (\widehat{C}_{\mathsf{off}}, \widehat{C}'_{\mathsf{off}}), \\ \{M_i, \mathsf{sk}_{M_i} := (\mathbf{r}_i, \mathsf{dig}_{M_i}, \mathsf{FE}'.\mathsf{sk}_{M_i})\}_{i\in[Q_{\mathsf{key}}]}, \\ \{x_{j,\mathsf{pub}} := (1^{t_j}, \mathbf{x}_{j,\mathsf{pub}}), \Delta^*_j := \mathsf{FE}'.\mathsf{ct}_j\}_{j\in[Q_{\mathsf{msg}}]} \end{array}\right) \approx_c \left(\begin{array}{c} \mathsf{aux}, \mathsf{mpk} := \mathsf{FE}'.\mathsf{mpk}, \mathsf{ct}_{\mathsf{off}} := (\widehat{C}_{\mathsf{off}}, \widehat{C}'_{\mathsf{off}}), \\ \{M_i, \mathsf{sk}_{M_i} := (\mathbf{r}_i, \mathsf{dig}_{M_i}, \mathsf{FE}'.\mathsf{sk}_{M_i})\}_{i\in[Q_{\mathsf{key}}]}, \\ \{x_{j,\mathsf{pub}} := (1^{t_j}, \mathbf{x}_{j,\mathsf{pub}}), \Delta^{\$}_j \xleftarrow{\$} \mathsf{FE}'.\mathcal{CT}\}_{j\in[Q_{\mathsf{msg}}]} \end{array}\right) \tag{30}$$

assuming we have

$$\left(\begin{array}{c} \mathsf{aux}, \{M_i\}_{i\in[Q_{\mathsf{key}}]}, \{x_{j,\mathsf{pub}} := (1^{t_j}, \mathbf{x}_{j,\mathsf{pub}})\}_{j\in[Q_{\mathsf{msg}}]}, \\ \{\delta^*_{i,j} := M_i(x_j)\}_{(i,j)\in[Q_{\mathsf{key}}]\times[Q_{\mathsf{msg}}]} \end{array}\right) \approx_c \left(\begin{array}{c} \mathsf{aux}, \{M_i\}_{i\in[Q_{\mathsf{key}}]}, \{x_{j,\mathsf{pub}} := (1^{t_j}, \mathbf{x}_{j,\mathsf{pub}})\}_{j\in[Q_{\mathsf{msg}}]}, \\ \{\delta^{\$}_{i,j} \xleftarrow{\$} \mathcal{Y}_\lambda\}_{(i,j)\in[Q_{\mathsf{key}}]\times[Q_{\mathsf{msg}}]} \end{array}\right) \tag{31}$$

where

- $(\mathsf{aux}, \{M_i\}_{i\in[Q_{\mathsf{key}}]}, \{x_j = (x_{j,\mathsf{pub}}, x_{j,\mathsf{pri}})\}_{j\in[Q_{\mathsf{msg}}]}) \leftarrow \mathsf{Samp}(1^\lambda)$
- $\mathsf{FE}'.(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE}'.\mathsf{Setup}(1^\lambda, \mathsf{param}')$
- $(\widehat{C}_{\mathsf{off}}, \mathsf{LprIO.st}) \leftarrow \mathsf{LprIO.ObfOff}(1^\lambda, 1^S)$ and $(\widehat{C}'_{\mathsf{off}}, \mathsf{LprIO.st}') \leftarrow \mathsf{LprIO.ObfOff}(1^\lambda, 1^S)$
- for all $i \in [Q_{\mathsf{key}}]$:
  - $\mathbf{r}_i \xleftarrow{\$} \{0,1\}^\lambda$
  - $\mathsf{dig}_{M_i} \leftarrow \mathsf{LprIO.Digest}(\{(k, C^{(k)}_{\mathsf{tm}})\}_{k\in[|C_{\mathsf{tm}}|]})$ for $C_{i,\mathsf{tm}} = C[M_i]$ defined in Figure 13
  - $\mathsf{FE.sk}_{M_i} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, C_{i,\mathsf{kgen}})$ for $C_{\mathsf{kgen}}[\mathbf{r}_i, \mathsf{dig}_{M_i}]$ defined in Figure 14

- for all $j \in [Q_{\mathsf{msg}}]$:
  - $\mathsf{sd}_{j,\mathsf{dig}}, \mathsf{sd}_{j,\mathsf{pub}}, \mathsf{sd}_{j,\mathsf{cir}}, \mathsf{sd}'_{j,\mathsf{cir}}, \mathsf{sd}_{j,\mathsf{bgc}}, \mathsf{sd}'_{j,\mathsf{bgc}} \xleftarrow{\$} \{0,1\}^\lambda$
  - $\mathsf{dig}_{\mathbf{x}_{\mathsf{pub}}} \leftarrow \mathsf{LprIO.Digest}(1^\lambda, \{(k, \mathbf{x}_{\mathsf{pub}}[k])\}_{k \in [\ell_{\mathsf{pub}}]})$
  - $\mathbf{x}'_{j,\mathsf{pri}} = (\mathsf{bits}(t_j), \mathsf{dig}_{\mathbf{x}_{j,\mathsf{pub}}}, \mathbf{x}_{j,\mathsf{pri}}, \mathsf{LprIO.st}, \mathsf{LprIO.st}', \mathsf{sd}_{j,\mathsf{dig}}, \mathsf{sd}_{j,\mathsf{pub}}, \mathsf{sd}_{j,\mathsf{cir}}, \mathsf{sd}'_{j,\mathsf{cir}}, \mathsf{sd}_{j,\mathsf{bgc}}, \mathsf{sd}'_{j,\mathsf{bgc}})$
  - $\mathsf{FE}'.\mathsf{ct} \leftarrow \mathsf{FE}'.\mathsf{Enc}(\mathsf{FE}'.\mathsf{mpk}, \mathbf{x}'_{j,\mathsf{pri}})$
- $\mathsf{FE}'.\mathcal{CT}$ denotes the ciphertext space of FE'.

We note that $\mathsf{FE}'.\mathsf{ct}_{\mathsf{off}}$ does not appear in the above distributions because FE' has *trivial offline encryption* by assumption. We invoke the VerSel-prCT security of FE' with respect to a sampler $\mathsf{Samp}_{\mathsf{FE}'}(1^\lambda)$ which outputs the following

$$\left(\begin{array}{c} \mathsf{aux}_{\mathsf{FE}'} := \left(\mathsf{aux}, \widetilde{C}_{\mathsf{off}}, \widetilde{C}'_{\mathsf{off}}, \{M_i, \mathbf{r}_i\}_{i \in [Q_{\mathsf{key}}]}, \{x_{j,\mathsf{pub}}\}_{j \in [Q_{\mathsf{msg}}]}\right), \\ \{C_{\mathsf{kgen}}[\mathbf{r}_i, \mathsf{dig}_{M_i}]\}_{i \in [Q_{\mathsf{key}}]}, \{x'_j := (x'_{j,\mathsf{pub}} = \varepsilon, x'_{j,\mathsf{pri}} = \mathbf{x}'_{j,\mathsf{pri}})\}_{j \in [Q_{\mathsf{msg}}]} \end{array}\right).$$

By the security of FE' with respect to sampler $\mathsf{Samp}_{\mathsf{FE}'}$, Equation (30) holds if

$$\left(\begin{array}{c} \mathsf{aux}_{\mathsf{FE}'}, \{C_{\mathsf{kgen}}[\mathbf{r}_i, \mathsf{dig}_{M_i}]\}_{i \in [Q_{\mathsf{key}}]}, \\ \{\Gamma^*_{i,j} := C_{\mathsf{kgen}}[\mathbf{r}_i, \mathsf{dig}_{M_i}](x'_j)\}_{(i,j) \in [Q_{\mathsf{key}}] \times [Q_{\mathsf{msg}}]} \end{array}\right) \approx_c \left(\begin{array}{c} \mathsf{aux}_{\mathsf{FE}'}, \{C_{\mathsf{kgen}}[\mathbf{r}_i, \mathsf{dig}_{M_i}]\}_{i \in [Q_{\mathsf{key}}]}, \\ \{\Gamma^{\$}_{i,j} \xleftarrow{\$} \{0,1\}^{\ell'_{\mathsf{out}}}\}_{(i,j) \in [Q_{\mathsf{key}}] \times [Q_{\mathsf{msg}}]} \end{array}\right) \tag{32}$$

The rest of the proof consists in proving (31) $\implies$ (32). Note that neither of the involved distributions contains anything related to FE' anymore. Therefore, the proof of this implication is conceptually the same as proving the analogous implication in the case of sRFE: (20) $\implies$ (21) (see Construction 4.11, proof of Proposition 4.10). We briefly highlight the modifications:

- LprIO in Construction 4.11 is a laconic pPRIO scheme *with global setup* whereas in Construction 7.1 it is a *plain* laconic pPRIO scheme. This simplifies the current proof as we can drop LprIO.crs from all distributions.
- Besides the difference with LprIO.crs, the circuit $C_{\mathsf{reg}}[i, \mathsf{LprIO.crs}, \mathsf{dig}_{M_i}]$ hardwires a slot index $i \in [L]$ whereas $C_{\mathsf{kgen}}[\mathbf{r}_i, \mathsf{dig}_{M_i}]$ hardwires a random vector $\mathbf{r}_i \xleftarrow{\$} \{0,1\}^\lambda$. Therefore, the current proof requires an additional step to argue that collisions between different $\mathbf{r}_i$'s happen only with negligible probability.
- (32) considers Q_{msg} challenge messages whereas (21) considers only a single one. This is related to the stronger *reusable* security notion that we attempt to show in the current proof, but it does not affect the argument since the security of the underlying LprIO scheme allows to provide many online obfuscations for a single offline obfuscation anyway (see Definition 3.44).
- Functions in (32) are indexed by $i \in [Q_{\mathsf{key}}]$ whereas the index set in (21) is $\mathcal{I}_{\mathsf{cor}} \cup \mathcal{I}_{\mathsf{mal}}$ (the set of corrupted or malicious slots). This is only a syntactical difference and does not affect the proof.

□

7.4 Application to KP-ABE and PE for TMs with Optimal Asymptotic Parameters

We build KP-ABE and PE schemes for the class $\mathcal{T}$ of all Turing machines. As both constructions are very similar, we present them at once. We use gray (resp. blue) to indicate that a component is only used in the case of KP-ABE (resp. PE). For notational simplicity, we describe the construction for the fixed message space $\mathcal{M} = \{0,1\}^\lambda$. We note that this restriction is without loss of generality. Indeed, using the hybrid encryption framework we can upgrade these schemes to support arbitrary message spaces while preserving our asymptotic efficiency parameters.

Construction 7.5 (KP-ABE and PE for $\mathcal{T}$)**.** The construction uses the following building blocks:

- A FE scheme $\mathsf{FE} = \mathsf{FE}.(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Agg}, \mathsf{EncOff}, \mathsf{EncOn}, \mathsf{Dec})$ with reusable VerSel-prCT security for the following function class $\mathcal{F}$, where

KP-ABE for $\mathcal{T}$:	$\mathcal{F} = \mathcal{T}_{\ell_{\mathsf{pri}}, \ell_{\mathsf{out}}}$	with $\ell_{\mathsf{pri}} = 2\lambda$ and $\ell_{\mathsf{out}} = \lambda$,
PE for $\mathcal{T}$:	$\mathcal{F} = \mathcal{T}_{\ell_{\mathsf{out}}}$	with $\ell_{\mathsf{out}} = \lambda$.

  Such FE schemes exist assuming LWE, pPRIO and VerSel-prCT secure FE for bounded depth circuits (Theorem 7.4).

- A pseudorandom function $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ with key space, input space and output space being $\{0,1\}^\lambda$. PRF can be instantiated assuming one-way functions.

The details of the construction are as follows.

$\mathsf{Setup}(1^\lambda)$**.** Run $\mathsf{FE}.(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE}.\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{pri}}}, 1^{\ell_{\mathsf{out}}})$ and output $(\mathsf{mpk} := \mathsf{FE}.\mathsf{mpk}, \mathsf{msk} := \mathsf{FE}.\mathsf{msk})$.

$\mathsf{KeyGen}(\mathsf{msk}, M)$**:** Parse $\mathsf{msk} = \mathsf{FE}.\mathsf{msk}$, sample $\mathbf{r} \xleftarrow{\$} \{0,1\}^\lambda$ and compute

$$\mathsf{FE}.\mathsf{sk}_M \leftarrow \mathsf{FE}.\mathsf{KeyGen}(\mathsf{FE}.\mathsf{msk}, M_{\mathsf{kgen}}[\mathbf{r}, M]) ,$$

where $M_{\mathsf{kgen}}[\mathbf{r}, M]$is defined in Figure 15. Output $\mathsf{sk}_M := (\mathbf{r}, \mathsf{FE}.\mathsf{sk}_M)$.

Public Input: an attribute $x \in \{0,1\}^*$

Private Input:
- an attribute $x \in \{0,1\}^*$
- a message $\mu \in \{0,1\}^\lambda$
- a PRF seed sd

Output: a bit string in $\{0,1\}^\lambda$

Hardwired Values:
- a vector $\mathbf{r} \in \{0,1\}^\lambda$
- a Turing machine M

- Compute $b = M(x)$.
- Output μ if $b = 0$ and $\mathsf{PRF}(\mathsf{sd}, i)$ if $b = 1$.

Figure 15: Definition of the Turing machine $M_{\mathsf{kgen}}[\mathbf{r}, M]$

$\mathsf{Enc}(\mathsf{mpk}, x, \mu)$**.** The encryption algorithm proceeds in two steps.

$\mathsf{EncOff}(\mathsf{mpk})$**.** Parse $\mathsf{mpk} = \mathsf{FE}.\mathsf{mpk}$, run

$$\mathsf{FE}.(\mathsf{ct}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{FE}.\mathsf{EncOff}(\mathsf{FE}.\mathsf{mpk})$$

and return $(\mathsf{ct}_{\mathsf{off}} := \mathsf{FE}.\mathsf{ct}_{\mathsf{off}}, \mathsf{st} = \mathsf{FE}.\mathsf{st})$

$\mathsf{EncOn}(\mathsf{st}, x, \mu)$**.** Parse $\mathsf{st} = \mathsf{FE}.\mathsf{st}$, sample a PRF seed $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$ and set $x_{\mathsf{pub}} := x,\ x_{\mathsf{pri}} = (\mu, \mathsf{sd})$ (resp. $x_{\mathsf{pub}} := \varepsilon,\ x_{\mathsf{pri}} = (x, \mu, \mathsf{sd})$). Then run

$$\mathsf{FE}.\mathsf{ct} \leftarrow \mathsf{FE}.\mathsf{Enc}(\mathsf{FE}.\mathsf{mpk}, x_{\mathsf{pub}}, x_{\mathsf{pri}}) ,$$

and output $\mathsf{ct}_{\mathsf{on}} = \mathsf{FE}.\mathsf{ct}_{\mathsf{on}}$.

The final output of $\mathsf{Enc}(\mathsf{mpk}, x, \mu)$ is $\mathsf{ct} = (\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}})$.

$\mathsf{Dec}(\mathsf{sk}_M, M, \mathsf{ct}, x)$**.** Parse $\mathsf{sk}_M = (\mathbf{r}, \mathsf{FE}.\mathsf{sk}_M)$ and $\mathsf{ct} = \mathsf{FE}.\mathsf{ct}$. Set $x_{\mathsf{pub}} := x$ (resp. $x_{\mathsf{pub}} := \varepsilon$), run

$$z \leftarrow \mathsf{FE}.\mathsf{Dec}(\mathsf{FE}.\mathsf{sk}_M, M_{\mathsf{kgen}}[\mathbf{r}, M], \mathsf{FE}.\mathsf{ct}, x_{\mathsf{pub}}) ,$$

and output z, where $M_{\mathsf{kgen}}[\mathbf{r}, M]$ is defined in Figure 15.

Proposition 7.6 (Correctness and Efficiency)**.** *The KP-ABE and PE schemes described in Construction 7.5 are correct and have the following parameters:*

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda)\,, \qquad |\mathsf{sk}_M| = \mathsf{poly}(\lambda)\,, \qquad |\mathsf{ct}| = \mathsf{poly}(\lambda) + |x|\,.$$

Proof. The correctness readily follows from the correctness of FE and the definition of $M_{\mathsf{kgen}}[\mathbf{r}, M]$. Specifically, for $\mathsf{sk}_M = (\mathbf{r}, \mathsf{FE.sk}_M)$, $\mathsf{ct} = \mathsf{FE.ct}$ and $x_{\mathsf{pub}} := x$ (resp. $x_{\mathsf{pub}} := \varepsilon$), we have

$$z := \mathsf{FE.Dec}\big(\mathsf{FE.sk}_M, M_{\mathsf{kgen}}[\mathbf{r}, M], \mathsf{FE.ct}, x_{\mathsf{pub}}\big) = M_{\mathsf{kgen}}[\mathbf{r}, M](x, \mu, \mathsf{sd}) = \begin{cases} \mu & \text{if } M(x) = 0 \\ \mathsf{PRF}(\mathsf{sd}, \mathbf{r}) & \text{if } M(x) = 1\,. \end{cases}$$

To obtain the parameters, we plug the values of ℓ_{pri} and ℓ_{out} into the parameters of FE (Theorem 7.4). □

Proposition 7.7 (Security)**.** *If* FE *satisfies reusable* VerSel-prCT *security and* PRF *is secure, then the Construction 7.5 satisfies reusable* VerSel-INDr *security.*

Our construction is very similar to the KP-ABE scheme for unbounded depth circuits in [AKY24b, Section 6]. We therefore omit the proof.

Summarizing our results we obtain the following theorem. Specifically, we consider a ciphertext ct encrypting a message μ of length ℓ_{in} (using the hybrid encryption framework) with respect to an attribute x of length $\ell_{\mathsf{att}} = \ell$.

Theorem 7.8. *Assuming LWE, and prFE, there exist KP-ABE and PE for Turing machines satisfying reusable* VerSel-INDr *security with the following parameters:*

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda)\,, \qquad |\mathsf{sk}_M| = \mathsf{poly}(\lambda)\,, \qquad |\mathsf{ct}| = \mathsf{poly}(\lambda) + \ell_{\mathsf{in}} + \ell_{\mathsf{att}}\,.$$

These parameters are asymptotically optimal.

7.5 Application to CP-ABE for TMs with Optimal Asymptotic Parameters

We build CP-ABE schemes for the class $\mathcal{T}$ of all Turing machines. As in the case of KP-ABE, we describe the construction for the fixed message space $\mathcal{M} = \{0,1\}^\lambda$ which can be generically upgraded to support arbitrary message spaces using the hybrid encryption framework.

Construction 7.9 (CP-ABE for $\mathcal{T}$)**.** The construction uses the following building blocks:

- A KP-ABE scheme $\mathsf{kpABE} = \mathsf{kpABE.}(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc} = (\mathsf{EncOff}, \mathsf{EncOn}), \mathsf{Dec})$ for the policy class $\mathcal{T}$ satisfying reusable VerSel-INDr security. We denote the output space of $\mathsf{kpABE.EncOn}$ by $\mathsf{kpABE.}\mathcal{CT}_{\mathsf{on}} = \{0,1\}^{\ell_{\mathsf{on}}}$ and require its size to be some fixed polynomial $\ell_{\mathsf{on}} = \ell_{\mathsf{on}}(\lambda)$ in λ. Without loss of generality, we assume that the state output by $\mathsf{kpABE.EncOff}$ and the random coins of $\mathsf{kpABE.EncOn}$ be in $\{0,1\}^\lambda$. To achieve the former, we let the state be the random coins used by $\mathsf{kpABE.EncOff}$ which can in turn be replaced with a string in $\{0,1\}^\lambda$ using a PRF. For the latter, we can also use a PRF to derive longer (pseudo-)random coins if needed. A KP-ABE scheme with the required properties exist assuming LWE, pPRIO and VerSel-prCT secure FE for bounded depth circuits (Theorem 7.8).
- A FE scheme $\mathsf{FE} = \mathsf{FE.}(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Agg}, \mathsf{EncOff}, \mathsf{EncOn}, \mathsf{Dec})$ with VerSel-prCT security (Definition 3.27)[10] for the circuit class $\mathcal{C}_{\ell_{\mathsf{pub}}, \ell_{\mathsf{pri}}, \ell_{\mathsf{out}}}$, where $\ell_{\mathsf{pub}} = 0$, $\ell_{\mathsf{pri}} = 3\lambda$ and $\ell_{\mathsf{out}} = \ell_{\mathsf{on}}$. We can construct such an FE scheme assuming prFE and LWE (Fact 3.28).
- A pseudorandom function $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ with key space, input space and output space being $\{0,1\}^\lambda$. PRF can be instantiated assuming one-way functions.

The details of the CP-ABE scheme are as follows.

[10] Here, we do not need reusability, i.e., $Q_{\mathsf{msg}} = 1$ suffices.

$\mathsf{Setup}(1^\lambda)$**.** Run $\mathsf{FE}.(\mathsf{mpk},\mathsf{msk}) \leftarrow \mathsf{FE}.\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{pub}}}, 1^{\ell_{\mathsf{pri}}}, 1^{\ell_{\mathsf{out}}})$ and output $(\mathsf{mpk} := \mathsf{FE}.\mathsf{mpk}, \mathsf{msk} := \mathsf{FE}.\mathsf{msk})$.

$\mathsf{KeyGen}(\mathsf{msk}, x)$**:** Parse $\mathsf{msk} = \mathsf{FE}.\mathsf{msk}$, sample $\mathbf{r} \xleftarrow{\$} \{0,1\}^\lambda$ and compute

$$\mathsf{FE}.\mathsf{sk}_x \leftarrow \mathsf{FE}.\mathsf{KeyGen}(\mathsf{FE}.\mathsf{msk}, C_{\mathsf{kgen}}[\mathbf{r}, x]) ,$$

where $C_{\mathsf{kgen}}[\mathbf{r}, x]$is defined in Figure 16. Output $\mathsf{sk}_x := (\mathbf{r}, \mathsf{FE}.\mathsf{sk}_x)$.

(Private) Input:
- a message $\mu \in \{0,1\}^\lambda$
- a kpABE state $\mathsf{kpABE}.\mathsf{st} \in \{0,1\}^\lambda$
- a PRF seed $\mathsf{sd} \in \{0,1\}^\lambda$

Output: a kpABE online ciphertext $\mathsf{kpABE}.\mathsf{ct}_{\mathsf{on}} \in \mathsf{kpABE}.\mathcal{CT}_{\mathsf{on}}$

Hardwired Values:
- a vector $\mathbf{r} \in \{0,1\}^\lambda$
- an attribute $x \in \{0,1\}^*$

- Compute $R = \mathsf{PRF}(\mathsf{sd}, \mathbf{r})$.
- Output $\mathsf{kpABE}.\mathsf{ct}_{\mathsf{on}} \leftarrow \mathsf{kpABE}.\mathsf{EncOn}(\mathsf{kpABE}.\mathsf{st}, x, \mu; R)$.

Figure 16: Definition of the circuit $C_{\mathsf{kgen}}[\mathbf{r}, x]$

$\mathsf{Enc}(\mathsf{mpk}, M, \mu)$**:** Parse $\mathsf{mpk} = \mathsf{FE}.\mathsf{mpk}$, sample $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$ and run

$$\begin{aligned}
\mathsf{kpABE}.(\mathsf{mpk},\mathsf{msk}) &\leftarrow \mathsf{kpABE}.\mathsf{Setup}(1^\lambda)\\
\mathsf{kpABE}.(\mathsf{ct}_{\mathsf{off}},\mathsf{st}) &\leftarrow \mathsf{kpABE}.\mathsf{EncOff}(\mathsf{kpABE}.\mathsf{mpk})\\
\mathsf{kpABE}.\mathsf{sk}_M &\leftarrow \mathsf{kpABE}.\mathsf{KeyGen}(\mathsf{kpABE}.\mathsf{msk}, M)\\
\mathsf{FE}.\mathsf{ct} &\leftarrow \mathsf{FE}.\mathsf{Enc}\big(\mathsf{FE}.\mathsf{mpk}, x_{\mathsf{pub}} = \varepsilon, x_{\mathsf{pri}} = (\mu, \mathsf{kpABE}.\mathsf{st}, \mathsf{sd})\big) .
\end{aligned}$$

Output $\mathsf{ct} := (\mathsf{kpABE}.\mathsf{ct}_{\mathsf{off}}, \mathsf{kpABE}.\mathsf{sk}_M, \mathsf{FE}.\mathsf{ct})$.

$\mathsf{Dec}(\mathsf{sk}_x, x, \mathsf{ct}, M)$**:** Parse $\mathsf{sk}_x = (\mathbf{r}, \mathsf{FE}.\mathsf{sk}_x)$ and $\mathsf{ct} = (\mathsf{kpABE}.\mathsf{ct}_{\mathsf{off}}, \mathsf{kpABE}.\mathsf{sk}_M, \mathsf{FE}.\mathsf{ct})$. Run

$$\begin{aligned}
\mathsf{kpABE}.\mathsf{ct}_{\mathsf{on}} &\leftarrow \mathsf{FE}.\mathsf{Dec}\big(\mathsf{FE}.\mathsf{sk}_x, C_{\mathsf{kgen}}[\mathbf{r}, x], \mathsf{FE}.\mathsf{ct}, \varepsilon\big)\\
z &\leftarrow \mathsf{kpABE}.\mathsf{Dec}\big(\mathsf{kpABE}.\mathsf{msk}, \mathsf{kpABE}.\mathsf{sk}_M, M, \mathsf{kpABE}.(\mathsf{ct}_{\mathsf{off}}, \mathsf{ct}_{\mathsf{on}}), x\big) ,
\end{aligned}$$

and output z, where $C_{\mathsf{kgen}}[\mathbf{r}, x]$ is defined in Figure 16.

Proposition 7.10 (Correctness and Efficiency)**.** *The CP-ABE scheme in Construction 7.9 is correct and has the following parameters:*

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda) , \qquad |\mathsf{sk}_x| = \mathsf{poly}(\lambda) , \qquad |\mathsf{ct}| = \mathsf{poly}(\lambda) .$$

Proof. Correctness readily follows from the correctness of the building blocks and the definition of $C_{\mathsf{kgen}}[\mathbf{r}, x]$. Specifically, for $\mathsf{sk}_x = (\mathbf{r}, \mathsf{FE}.\mathsf{sk}_x)$ and $\mathsf{ct} = (\mathsf{kpABE}.\mathsf{ct}_{\mathsf{off}}, \mathsf{kpABE}.\mathsf{sk}_M, \mathsf{FE}.\mathsf{ct})$, we have

$$\begin{aligned}
&\mathsf{FE}.\mathsf{Dec}\big(\mathsf{FE}.\mathsf{sk}_x, C_{\mathsf{kgen}}[\mathbf{r}, x], \mathsf{FE}.\mathsf{ct}, \varepsilon\big)\\
&= C_{\mathsf{kgen}}[\mathbf{r}, x](\mu, \mathsf{kpABE}.\mathsf{st}, \mathsf{sd})\\
&= \mathsf{kpABE}.\mathsf{EncOn}(\mu, \mathsf{kpABE}.\mathsf{st}, x; \mathsf{PRF}(\mathsf{sd}, \mathbf{r})) =: \mathsf{kpABE}.\mathsf{ct}_{\mathsf{on}}
\end{aligned}$$

Plugging this online ciphertext into the kpABE decryption algorithm yields

$$\mathsf{kpABE}.\mathsf{Dec}\big(\mathsf{kpABE}.\mathsf{msk}, \mathsf{kpABE}.\mathsf{sk}_M, M, \mathsf{kpABE}.(\mathsf{ct}_{\mathsf{off}}.\mathsf{ct}_{\mathsf{on}}), x\big) = \mu \qquad \text{if } M(x) = 0 .$$

For efficiency, when instating kpABE as proposed (Theorem 7.8), we have

$$|\mathsf{kpABE.mpk}| = \mathsf{poly}(\lambda)\,, \quad |\mathsf{kpABE.ct}_{\mathsf{off}}| = \mathsf{poly}(\lambda)\,, \quad |\mathsf{kpABE.ct}_{\mathsf{on}}| = \mathsf{poly}(\lambda)\,, \quad |\mathsf{kpABE.sk}_M| = \mathsf{poly}(\lambda)\,,$$

Then, plugging the values of $\ell_{\mathsf{pub}} = 0$, $\ell_{\mathsf{pri}} = 3\lambda$ and $\ell_{\mathsf{out}} = \ell_{\mathsf{on}}$ into the parameters of FE (Fact 3.28) gives

$$\begin{aligned}
|\mathsf{mpk}| &= |\mathsf{FE.mpk}| = \mathsf{poly}(\lambda) \\
|\mathsf{sk}_x| &= |\mathbf{r}| + \underbrace{|\mathsf{FE.sk}_x|}_{\ell_{\mathsf{out}} \cdot \mathsf{poly}(\lambda)} = \mathsf{poly}(\lambda) \\
|\mathsf{ct}| &= |\mathsf{kpABE.ct}_{\mathsf{off}}| + |\mathsf{kpABE.sk}_M| + \underbrace{|\mathsf{FE.ct}|}_{\mathsf{poly}(\lambda)+\ell_{\mathsf{pri}}} = \mathsf{poly}(\lambda)\,,
\end{aligned}$$

where we recall that $\ell_{\mathsf{on}}(\lambda) = \mathsf{poly}(\lambda)$. □

Proposition 7.11 (Security)**.** *If* kpABE *satisfies reusable* VerSel-INDr *security,* FE *is* VerSel-prCT *secure and* PRF *is secure, then the CP-ABE scheme in Construction 7.9 is* Sel-IND *secure.*

Our construction is very similar to the CP-ABE scheme for unbounded depth circuits in [AKY24b, Section 7]. We therefore omit the proof.

Summarizing our results we obtain the following theorem. Specifically, we consider a ciphertext ct encrypting a message μ of length ℓ_{in} (using the hybrid encryption framework).

Theorem 7.12. *Assuming LWE and prFE, there exists a* VerSel-INDr *secure CP-ABE for Turing machines with the following parameters:*

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda)\,, \qquad |\mathsf{sk}_x| = \mathsf{poly}(\lambda)\,, \qquad |\mathsf{ct}| = \mathsf{poly}(\lambda) + \ell_{\mathsf{in}}\,.$$

These parameters are asymptotically optimal.

Acknowledgments

This work was supported in part by the France 2030 ANR-22-PECY-003 SecureCompute Project.

References

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996.

[AKM+22] Shweta Agrawal, Fuyuki Kitagawa, Anuja Modi, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Bounded functional encryption for Turing machines: Adaptive security from general assumptions. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 618–647. Springer, Cham, November 2022.

[AKY24a] Shweta Agrawal, Simran Kumari, and Shota Yamada. Attribute based encryption for turing machines from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 352–386. Springer, Cham, August 2024.

[AKY24b] Shweta Agrawal, Simran Kumari, and Shota Yamada. Compact pseudorandom functional encryption from evasive LWE. Cryptology ePrint Archive, Report 2024/1719, 2024.

[AKY24c] Shweta Agrawal, Simran Kumari, and Shota Yamada. Pseudorandom multi-input functional encryption and applications. Cryptology ePrint Archive, Report 2024/1720, 2024.

[AM18] Shweta Agrawal and Monosij Maitra. FE and iO for Turing machines from minimal assumptions. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 473–512. Springer, Cham, November 2018.

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

[AMY19a] Shweta Agrawal, Monosij Maitra, and Shota Yamada. Attribute based encryption (and more) for nondeterministic finite automata from LWE. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 765–797. Springer, Cham, August 2019.

[AMY19b] Shweta Agrawal, Monosij Maitra, and Shota Yamada. Attribute based encryption for deterministic finite automata from DLIN. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 91–117. Springer, Cham, December 2019.

[AMYY25] Shweta Agrawal, Anuja Modi, Anshu Yadav, and Shota Yamada. Evasive LWE: Attacks, variants & obfustopia. Cryptology ePrint Archive, Report 2025/375, 2025.

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for Turing machines. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 125–153. Springer, Berlin, Heidelberg, January 2016.

[AS17] Shweta Agrawal and Ishaan Preet Singh. Reusable garbled deterministic finite automata from learning with errors. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *ICALP 2017*, volume 80 of *LIPIcs*, pages 36:1–36:13. Schloss Dagstuhl, July 2017.

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

[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, Cham, 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 Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *ICALP 2022*, volume 229 of *LIPIcs*, pages 28:1–28:20. Schloss Dagstuhl, July 2022.

[BDJ+24] Pedro Branco, Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, Spencer Peters, and Vinod Vaikuntanathan. Pseudorandom obfuscation and applications. Cryptology ePrint Archive, Report 2024/1742, 2024.

[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, Berlin, Heidelberg, May 2014.

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

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013.

[BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 535–564. Springer, Cham, April / May 2018.

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

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 264–302. Springer, Cham, November 2017.

[BÜW24] Chris Brzuska, Akin Ünal, and Ivy K. Y. Woo. Evasive LWE assumptions: Definitions, classes, and counterexamples. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 418–449. Springer, Singapore, December 2024.

[CCH+19] Jung Hee Cheon, Wonhee Cho, Minki Hhan, Jiseung Kim, and Changmin Lee. Statistical zeroizing attack: Cryptanalysis of candidates of BP obfuscation over GGH15 multilinear map. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 253–283. Springer, Cham, August 2019.

[CGH17] Yilei Chen, Craig Gentry, and Shai Halevi. Cryptanalyses of candidate branching program obfuscators. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 278–307. Springer, Cham, April / May 2017.

[CHKL18] Jung Hee Cheon, Minki Hhan, Jiseung Kim, and Changmin Lee. Cryptanalyses of branching program obfuscations over GGH13 multilinear map from the NTRU problem. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 184–210. Springer, Cham, August 2018.

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. Cryptology ePrint Archive, Report 2025/044, 2025.

[CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 577–607. Springer, Cham, August 2018.

[CW25] Valerio Cini and Hoeteck Wee. Faster ABE for turing machines from circular evasive LWE. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 94–125. Springer, Cham, May 2025.

[DJM+25] Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, and Vinod Vaikuntanathan. Simple and general counterexamples for private-coin evasive LWE. Cryptology ePrint Archive, Report 2025/374, 2025.

[DKL+23] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 417–446. Springer, Cham, April 2023.

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

[DQV+21] Lalita Devadas, Willy Quach, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Succinct LWE sampling, random polynomials, and obfuscation. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 256–287. Springer, Cham, November 2021.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, Berlin, Heidelberg, May 2004.

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

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

[GGH96] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Collision-free hashing from lattice problems. Cryptology ePrint Archive, Report 1996/009, 1996.

[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.

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

[GKMR23] Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 1065–1079. ACM Press, November 2023.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run Turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Berlin, Heidelberg, August 2013.

[GKW16] Rishab Goyal, Venkata Koppula, and Brent Waters. Semi-adaptive security and bundling functionalities made generic and easy. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 361–388. Springer, Berlin, Heidelberg, October / November 2016.

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

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 736–749. ACM Press, June 2021.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013.

[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, Cham, August 2020.

[GW20] Junqing Gong and Hoeteck Wee. Adaptively secure ABE for DFA from k-Lin and more. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 278–308. Springer, Cham, May 2020.

[GWW19] Junqing Gong, Brent Waters, and Hoeteck Wee. ABE for DFA from k-Lin. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 732–764. Springer, Cham, August 2019.

[HJL21] Samuel B. Hopkins, Aayush Jain, and Huijia Lin. Counterexamples to new circular security assumptions underlying iO. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 673–700, Virtual Event, August 2021. Springer, Cham.

[HJL25] Yao-Ching Hsieh, Aayush Jain, and Huijia Lin. Lattice-based post-quantum iO from circular security with random opening assumption (part II: zeroizing attacks against private-coin evasive LWE assumptions). Cryptology ePrint Archive, Report 2025/390, 2025.

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th FOCS*, pages 415–434. IEEE Computer Society Press, November 2023.

[HLL24] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. A general framework for lattice-based ABE using evasive inner-product functional encryption. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 433–464. Springer, Cham, May 2024.

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

[JLL23] Aayush Jain, Huijia Lin, and Ji Luo. On the optimal succinctness and efficiency of functional encryption and attribute-based encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 479–510. Springer, Cham, April 2023.

[JLLS23] Aayush Jain, Huijia Lin, Paul Lou, and Amit Sahai. Polynomial-time cryptanalysis of the subspace flooding assumption for post-quantum $i\mathcal{O}$. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 205–235. Springer, Cham, April 2023.

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

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

[KNTY19] Fuyuki Kitagawa, Ryo Nishimaki, Keisuke Tanaka, and Takashi Yamakawa. Adaptively secure and succinct functional encryption: Improving security and efficiency, simultaneously. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 521–551. Springer, Cham, August 2019.

[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, Cham, August 2017.

[LL20] Huijia Lin and Ji Luo. Compact adaptively secure ABE from k-Lin: Beyond NC^1 and towards NL. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 247–277. Springer, Cham, May 2020.

[LT17] Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 630–660. Springer, Cham, 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.

[MPV24] Surya Mathialagan, Spencer Peters, and Vinod Vaikuntanathan. Adaptively sound zero-knowledge SNARKs for UP. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 38–71. Springer, Cham, August 2024.

[MQR22] Mohammad Mahmoody, Wei Qi, and Ahmadreza Rahimi. Lower bounds for the number of decryption updates in registration-based encryption. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 559–587. Springer, Cham, November 2022.

[Pel18] Alice Pellet-Mary. Quantum attacks against indistinguishablility obfuscators proved secure in the weak multilinear map model. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 153–183. Springer, Cham, August 2018.

[PS25] Tapas Pal and Robert Schädlich. Registered functional encryption for attribute-weighted sums with access control. Cryptology ePrint Archive, Paper 2025/836, 2025.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

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

[Wat12] Brent Waters. Functional encryption for regular languages. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 218–235. Springer, Berlin, Heidelberg, August 2012.

[Wee22] Hoeteck Wee. Optimal broadcast encryption and CP-ABE from evasive lattice assumptions. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 217–241. Springer, Cham, May / June 2022.

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

[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, Cham, October 2021.

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

[YZGC25] Xinrui Yang, Yijian Zhang, Ying Gao, and Jie Chen. Registered ABE for circuits from evasive lattice assumptions. Cryptology ePrint Archive, Paper 2025/807, 2025.

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

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

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

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

A Unbounded Blind Batch Encryption

A.1 Overview

Existing blind batch encryption (BBE, Section 3.8) schemes consider a Setup algorithm takes as input some value N, which bounds the length of the secret keys the scheme can support. We call such BBE schemes as *bounded*. However, as we detail later in Section B, our laconic pPRIO with global setup construction requires an *unbounded* BBE scheme, i.e., the Setup algorithm does not fix the length of the secret keys. Therefore, our goal in this section is to construct such an unbounded BBE scheme.

In order to do so, we revisit the transformation from [BLSV18], that given a BBE scheme BBE with certain succinctness properties produces a new BBE scheme BBE' whose CRS and public key sizes are independent of the secret key length. The new BBE scheme BBE' is defined as follows. The setup algorithm generates $d = \log(N/\lambda)$ many CRSes, where N is the length of the secret key, by running the $\mathsf{BBE.Setup}(1^\lambda, 1^{2\lambda})$ algorithm d times. Hence, the crs' of the BBE' scheme is defined as $\mathsf{crs}' := (\mathsf{crs}_0, \ldots, \mathsf{crs}_{d-1})$. The $\mathsf{BBE}'.\mathsf{Gen}$ algorithm hashes the input $X \in \{0,1\}^N$ by considering a Merkle tree of depth d, where the leaves of the tree consist of N/λ nodes and X is evenly split and assigned to the corresponding node. Then, starting from the level of the tree right above the leaves until to the root, we assign a hash value h_v to an internal node v of the tree. For each level i of the tree, we use the corresponding crs_i for hashing, and the final output of the $\mathsf{BBE}'.\mathsf{Gen}$ algorithm is the hash value assigned to the root, i.e., h_ϵ. The transformation from [BLSV18] considered batch encryption setting, that is we do not encrypt for one specific index $i \in [N]$, but instead we encrypt for *all* $i \in [N]$ at once. To this end, the encryption algorithm for each node v generates a garbled circuit $\widetilde{C}_v$, where the garbled circuits associated with the internal nodes are obtained by garbling a circuit that takes as input h_v and outputs a BBE ciphertext that encrypts the labels of the garbled circuits corresponding to its children under the public key h_v. For the leaf nodes, the garbled circuits encrypt the messages instead of the labels. The final ciphertext is composed of all the garbled circuits, along with the labels $\mathsf{lab}_{h_\epsilon}$ that correspond to h_ϵ associated with the root. To decrypt a ciphertext, for each leaf node v, we traverse the hash tree from the root to v and obtain the message corresponding to X_v. Concretely, we first evaluate the garbled circuit $\widetilde{C}_\epsilon$ on labels $\mathsf{lab}_{h_\epsilon}$ to recover the BBE ciphertext ct_ϵ corresponding to the root node, which encrypts the labels of the garbled circuit of the next level. This BBE ciphertext can be decrypted using the string concatenation of h_0 and h_1, which in turn gives the labels $\mathsf{lab}_{h_{v_1}}$ corresponding to h_{v_1}, where v_1 is the first bit of v. Hence, we traverse down the tree in this way until we reach the leaf nodes, where we recover the message itself.

We make two important observations about this transformation. The first observation is that since we are interested in constructing $\mathsf{BBE}'.\mathsf{SingleEnc}$ and $\mathsf{BBE}'.\mathsf{SingleDec}$ algorithm to encrypt a message for an index $i \in [N]$, we can just consider the leaf node v corresponding to the index i. Therefore, during the aforedescribed encryption algorithm we can remove the ciphertext components that are not necessary for traversing down the tree to the leaf node v. More precisely, we include only $\mathsf{lab}_{h_\epsilon}$ and $\widetilde{C}_{\mathsf{bits}(i)}$, i.e., the garbled circuits corresponding to the ancestors of v in the tree, in the ciphertext. These are sufficient to decrypt for some specific leaf node v using the decryption procedure explained above.

The second observation is that instead of generating $d = \log(N/\lambda)$ many CRSes, we can simply generate λ many CRSes during $\mathsf{BBE}'.\mathsf{Setup}$. This in turn ensures that our $|\mathsf{crs}'|$ remains $\mathsf{poly}(\lambda)$, while at the same time we can support a tree of exponential size, i.e., of size 2^λ, which is sufficient to handle any polynomial length input X. This implies that during $\mathsf{BBE}'.\mathsf{Setup}$ we no longer need to pass the secret key length as an additional parameter, and hence, we obtain an *unbounded* BBE construction.

A.2 Construction

In this section we construct an unbounded BBE scheme by relying on the transformation from α-succinctness to full succinctness for BBE given in [BLSV18]. The construction makes use of the following building blocks:

- A (bounded) blind batch encryption scheme $\mathsf{BBE} = \mathsf{BBE.}(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with $\frac{1}{2}$-succinctness,

Input: a BBE public key $h \in \{0,1\}^\beta$

Output: a BBE ciphertexts ct_v

Hardwired Values:
- a BBE common reference string crs_j for index $j = |v|$
- a matrix $\mathbf{A} \in \Sigma^{\lambda \times 2}$ (for some alphabet $\Sigma = \{0,1\}^{\mathsf{poly}(\lambda)}$)
- a BBE randomness $r^{(v)}$

- Compute $\mathsf{ct}_v \leftarrow \mathsf{BBE.Enc}(\mathsf{crs}_j, h, \mathbf{A}; r^{(v)})$.
- Output ct_v

Figure 17: Definition of the circuit $C_v[\mathsf{crs}_j, \mathbf{A}, r^{(v)}]$.

message space $\mathcal{M} = \{0,1\}^\lambda$ and ciphertext space $\mathsf{BBE}.\mathcal{CT} = \{0,1\}^{\ell_{\mathsf{ct}}}$.

- A blind garbling scheme $\mathsf{bGC} = \mathsf{bGC}.(\mathsf{Garble}, \mathsf{Eval}, \mathsf{Sim})$ for any circuit. We can instantiate bGC from any one-way function (Fact 3.17).

We construct an unbounded blind batch encryption scheme $\mathsf{BBE}' = \mathsf{BBE}'.(\mathsf{Setup}', \mathsf{Gen}', \mathsf{SingleEnc}', \mathsf{SingleDec}')$, where the message space is $\mathcal{M} = \{0,1\}^\lambda$:

$\mathsf{Setup}'(1^\lambda)$**:** Set $\beta = \frac{\lambda}{2}$, run $\mathsf{crs}_j \leftarrow \mathsf{BBE.Setup}(1^\lambda, 1^\lambda)$ for all $j \in [0, \lambda-1]$, set $\mathsf{crs}' := (\beta, \mathsf{crs}_0, \ldots, \mathsf{crs}_{\lambda-1})$, and output crs'. (Note that each crs_i supports key generation from $\{0,1\}^\lambda \to \{0,1\}^\beta$.)

$\mathsf{Gen}'(\mathsf{crs}', x \in \{0,1\}^*)$**:** Parse $\mathsf{crs}' = (\beta, \mathsf{crs}_0, \ldots, \mathsf{crs}_{\lambda-1})$, define $N := |x|$ and break x into blocks of size β as $y := (y_0, \ldots, y_{D-1}) \in (\{0,1\}^\beta)^D$, where $D = \frac{N}{\beta}$. Then, set $d = \log(D)$ and define a hash tree of depth d in the following way: each node $v \in \{0,1\}^{\leq d}$ is labeled with a string $h_v \in \{0,1\}^\beta$, such that $h_v = y_v$ for all $v \in \{0,1\}^d$ and for all $v \in \{0,1\}^j$ such that $j < d$, we set

$$h_v := \mathsf{BBE.Gen}(\mathsf{crs}_j, h_{v||0} || h_{v||1}).$$

Output $h := (h_\epsilon, d)$, where h_ϵ denotes the hash value assigned to the root node ϵ of the tree.

$\mathsf{SingleEnc}'(\mathsf{crs}', h, i, \mathbf{m} := (\mathbf{m}_0, \mathbf{m}_1) \in (\{0,1\}^\lambda)^2)$**:** Parse $\mathsf{crs}' := (\beta, \mathsf{crs}_0, \ldots, \mathsf{crs}_{\lambda-1})$ and $h := (h_\epsilon, d)$. If $i \notin [2^{d-1}]$, then return $\perp$. Otherwise, for each $v \in \{0,1\}^{\leq d-1}$ define the circuit $C_v[\mathsf{crs}_j, \mathbf{A}, r^{(v)}]$ shown in Figure 17.

Then, we compute the following:

- For $v \in \{0,1\}^{d-1}$ such that $v = i$, compute $(\widetilde{C}_v, \mathsf{lab}^{(v)}) \leftarrow \mathsf{bGC.Garble}(1^\lambda, C_v[\mathsf{crs}_{d-1}, \mathbf{M}, r^{(v)}])$, for fresh encryption randomness $r^{(v)}$, where $\mathbf{M} := \begin{bmatrix} \mathbf{m}_0^\top & \mathbf{m}_1^\top \end{bmatrix} \in \{0,1\}^{\lambda \times 2}$ is concatenation of the input messages $\mathbf{m}_0, \mathbf{m}_1$.
- For every $v \in \{0,1\}^j$ and $j < d-1$, compute $(\widetilde{C}_v, \mathsf{lab}^{(v)}) \leftarrow \mathsf{bGC.Garble}(1^\lambda, C_v[\mathsf{crs}_j, \mathbf{A}^{(v)}, r^{(v)}])$, where $\mathbf{A}^{(v)} := \begin{bmatrix} \mathsf{lab}^{(v||0)} \\ \mathsf{lab}^{(v||1)} \end{bmatrix}$ denotes the $\lambda \times 2$ matrix written as concatenation of the labels $\mathsf{lab}^{(v||0)}, \mathsf{lab}^{(v||1)}$.

Finally, the $\mathsf{SingleEnc}'$ algorithm outputs

$$\mathsf{ct} := \left(\widetilde{C}_i, \left(\widetilde{C}_v\right)_{v:\, v \in \mathsf{bits}(i)[0:d-2]}, \left(\mathsf{lab}^{(\epsilon)}_{k, h_\epsilon[k]}\right)_{k \in [\beta]}\right),$$

where $h_\epsilon[k]$ denotes the k-th bit of h_ϵ and $\widetilde{C}_v$ denotes the garbled circuits corresponding to the ancestors (up to the root) of the i-th node in level $d-1$, i.e., $i \in \{0,1\}^{d-1}$ and $\mathsf{bits}(i)[0] = \epsilon$.

$\mathsf{SingleDec}'(\mathsf{crs}', x, i, \mathsf{ct})$**:** Parse $\mathsf{crs}' := (\beta, \mathsf{crs}_0, \ldots, \mathsf{crs}_{\lambda-1})$ and $\mathsf{ct} := \left(\widetilde{C}_i, \left(\widetilde{C}_v\right)_{v:\, v \in \mathsf{bits}(i)[0:d-2]}, \mathsf{lab}^{(\epsilon)}\right)$, and compute the following:

- For each $v \in \{0,1\}^{<d-1}$ (in order from the shortest to the longest string) such that $v \in \mathsf{bits}(i)[0 : d-2]$, i.e., v is an ancestor of the i-th node in level $d-1$, compute $\mathsf{ct}_v := \mathsf{bGC.Eval}(\tilde{C}_v, \mathsf{lab}^{(v)})$, and $(\mathsf{lab}^{(v\|0)}, \mathsf{lab}^{(v\|1)}) := \mathsf{BBE.Dec}(\mathsf{crs}_j, h_{v\|0} \| h_{v\|1}, \mathsf{ct}_v)$.
- For $v \in \{0,1\}^{d-1}$ such that $v = i$, compute $\mathsf{ct}_v := \mathsf{bGC.Eval}(\tilde{C}_v, \mathsf{lab}^{(v)})$ and recover the message $m := \mathsf{BBE.Dec}(\mathsf{crs}_{d-1}, h_{v\|0} \| h_{v\|1}, \mathsf{ct}_v)$.
- Output $\mathbf{m}$.

Remark A.1. We remark that if the crs_i of the underlying bounded BBE scheme is a uniformly random bit string, then crs' from our construction is also a uniformly random bit string. We note that β is just a fixed system parameter that is independent of the common reference string.

Proposition A.2 (Correctness, Efficiency and Succinctness)**.** *The unbounded BBE scheme in Construction A.2 is correct, efficient and succinct. Specifically, it has the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda), \quad |h| = O(\lambda), \quad |\mathsf{ct}| = \mathsf{poly}(\lambda, \log N),$$

where N denotes the length of the underlying secret key.

Proposition A.3 (Security)**.** *Assume* BBE *is adaptively secure and adaptively blind (Definitions 3.21 and 3.22), and* bGC *is simulation secure and blind (Definitions 3.14 and 3.15). Then, Construction A.2 is adaptively secure and adaptively blind.*

The proofs of Propositions A.2 and A.3 are provided in Sections A.3 and A.4. Instantiating the underlying BBE scheme with the LWE-based construction given in [AKY24b][11], we obtain the following theorem.

Theorem A.4. *Assuming LWE, there exists an adaptively secure and fully succinct unbounded BBE scheme satisfying*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda), \quad |h| = O(\lambda), \quad |\mathsf{ct}| = \mathsf{poly}(\lambda, \log N),$$

where $\mathsf{crs} \leftarrow \mathsf{Setup}'(1^\lambda)$, $h := (h_\epsilon, d) := \mathsf{Gen}'(\mathsf{crs}, x)$, *for some* $x \in \{0,1\}^N$, *and* $\mathsf{ct} \leftarrow \mathsf{SingleEnc}'(\mathsf{crs}, h, i, \mathbf{m})$ *for some* $i \in [2^{d-1}]$ *and* $\mathbf{m} \in \mathcal{M}^2$.

A.3 Proof of Correctness, Efficiency and Succinctness

Proof of Proposition A.2. We argue that Construction A.2 is correct, efficient and succinct.

Correctness. We prove correctness by an inductive argument as in [BLSV18]. Let $\mathsf{ct} \leftarrow \mathsf{SingleEnc}'(\mathsf{crs}', h, i, \mathbf{M})$, for some public key $h := (h_\epsilon, d)$, index $i \in [2^{d-1}]$ and message $\mathbf{M} := \begin{bmatrix} \mathbf{m}_0^\top & \mathbf{m}_1^\top \end{bmatrix} \in \{0,1\}^{\lambda \times 2}$. For each step of the decryption process corresponding to $v \in \{0,1\}^{<d-1}$ such that $v \in \mathsf{bits}(i)[0 : d-2]$, i.e., v is an ancestor of the i-th node in level $d-1$, by the correctness of the blind garbling scheme bGC we obtain

$$\mathsf{ct}_v = \mathsf{BBE.Enc}\left(\mathsf{crs}_{|v|}, h_v, \begin{bmatrix} \mathsf{lab}^{(v\|0)} \\ \mathsf{lab}^{(v\|1)} \end{bmatrix}; r^{(v)}\right).$$

Then, by correctness of the underlying blind batch encryption scheme BBE, decrypting ct_v gives us

$$\mathsf{lab}^{(v\|0)} = \left(\mathsf{lab}^{(v\|0)}_{k,(h_v\|0)[k]}\right)_{k\in[\beta]} \quad \text{and} \quad \mathsf{lab}^{(v\|0)} = \left(\mathsf{lab}^{(v\|1)}_{k,(h_v\|1)[k]}\right)_{k\in[\beta]}.$$

[11] Looking ahead, in Section A.4 we prove that our unbounded BBE construction achieves adaptive security and blindness (as in Definitions 3.21 and 3.22) by assuming that the underlying bounded BBE scheme is also adaptive. While [AKY24b] proved that their bounded BBE achieves selective security and blindness, we observe that their scheme also achieves our adaptive definitions. This is because throughout the security proof their CRS remains a uniformly random bit string and is never programmed depending on the challenge secret key or index.

Hence, we get that for $v \in \{0,1\}^{d-1}$ such that $v = i$,

$$\mathsf{ct}_v = \mathsf{BBE.Enc}\left(\mathsf{crs}_{d-1}, h_v, \mathbf{M}; r^{(v)}\right),$$

and therefore, we obtain $\mathbf{m}[k] = \mathbf{M}[k, y_v[k]]$ for every $k \in [\lambda]$ (note that $y_v = y_{v\|0} || y_{v\|1} \in \{0,1\}^\lambda$).

Efficiency. Encryption efficiency follows from the fact that we execute at most $D = \frac{N}{\beta} = 2^d$ operation on inputs of length $\mathsf{poly}(\lambda)$. This means that the overall encryption operation takes $\mathsf{poly}(\lambda, D) = \mathsf{poly}(\lambda, N)$ time, which is bounded by $\mathsf{poly}(\lambda)$ for polynomially bounded secret keys i.e., when $|x| = \mathsf{poly}(\lambda)$. Moreover, the ciphertext size is bounded by $\mathsf{poly}(\lambda, d) = \mathsf{poly}(\lambda, \log N)$, because it includes $\mathsf{poly}(d)$ number of garbled circuits, where the circuits have size $\mathsf{poly}(\lambda)$.

Succinctness. Full succinctness follows from the fact that the public key $h := (h_\epsilon, d)$ assigned to the secret key x under crs' is composed of a single BBE public key h_ϵ, which has length λ, and d, which can be represented with at most λ bits, and hence, the overall size is $|h| = O(\lambda)$. Additionally, we have that the crs size is bounded by $\mathsf{poly}(\lambda)$. □

A.4 Proof of Security

Proof of Proposition A.3. Our proof is an adaptation of [BLSV18, Lemma 3.2]. We first show adaptive security and then adaptive blindness.

Adaptive Security. Let $\mathsf{G}_0 = \mathsf{G}_{\epsilon,0}$ denote the original adaptive security game (Definition 3.21). To prove the proposition, we define the hybrids $\mathsf{G}_{v,1}$ and $\mathsf{G}_{v_{\mathsf{next}},0}$ for all $v \in \{0,1\}^{\leq d-1}$, where $v_{\mathsf{next}} \in \{0,1\}^{\leq d}$ is defined to be $v_{\mathsf{next}} := v + 1$ if $v \neq 1^i$ for some i, and $v_{\mathsf{next}} := 0^{i+1}$ if $v = 1^i$. In each of these hybrids we modify the challenge ciphertext as follows.

Game $\mathsf{G}_{v,1}$ for all $v \in \{0,1\}^{<d-1}$: This is the same as $\mathsf{G}_{v,0}$ except that we compute $\widetilde{C}_v$ and $\mathsf{lab}^{(v)}$ using the simulator of bGC as follows

$$\mathsf{ct}'_v \leftarrow \mathsf{BBE.Enc}(\mathsf{crs}_j, h_v, \mathbf{A}^{(v)}; r^{(v)})$$
$$(\widetilde{C}_v, \mathsf{lab}^{(v)}) \leftarrow \mathsf{bGC.Sim}(1^\lambda, 1^{|C_v[\mathsf{crs}_j, h_v, \mathbf{A}^{(v)}, r^{(v)}]|}, 1^\beta, \mathsf{ct}'_v),$$

where $j = |v|$. We have that for every $v \in \{0,1\}^{\leq d-1}$, $\mathsf{G}_{v,0} \approx_c \mathsf{G}_{v,1}$ due to the simulation security of bGC (Definition 3.14), because by induction we maintain the invariant that only the collection of labels $\mathsf{lab}^{(v)} := \left(\mathsf{lab}^{(v)}_{k,h_v[k]}\right)_{k\in[\beta]}$ are used throughout the computations.

Game $\mathsf{G}_{v_{\mathsf{next}},0}$ for all $v \in \{0,1\}^{<d-1}$: This is the same as $\mathsf{G}_{v,1}$ except that we modify the matrix $\mathbf{A}^{(v)}$. Concretely, instead of defining $\mathbf{A}^{(v)} := \begin{bmatrix} \mathsf{lab}^{(v\|0)} \\ \mathsf{lab}^{(v\|1)} \end{bmatrix}$ as in $\mathsf{G}_{v,1}$, we set it as $\mathbf{A}^{(v)} := \begin{bmatrix} \widetilde{\mathbf{A}}^{(v\|0)} \\ \widetilde{\mathbf{A}}^{(v\|1)} \end{bmatrix}$, where $\widetilde{\mathbf{A}}^{(w)}[k, h_w[k]] := \mathsf{lab}^{(w)}_{k,h_w[k]}$ and $\widetilde{\mathbf{A}}^{(v)}[k, 1-h_w[k]] \xleftarrow{\$} \{0,1\}^\lambda$, for all $k \in [\beta], b \in \{0,1\}, a \in \{0,1\}$ such that $w = v \| a$. We have that for every $v \in \{0,1\}^{<d-1}$, $\mathsf{G}_{v,1} \approx_c \mathsf{G}_{v_{\mathsf{next}},0}$ due to the adaptive security of BBE (Definition 3.21), because the decryption only outputs $\mathbf{A}^{(v)}[k, h_{v\|a}[k]] = \widetilde{\mathbf{A}}^{(v\|a)}[k, h_{v\|a}[k]]$ and we have only modified $\mathbf{A}^{(v)}[k, 1 - h_{v\|a}[k]]$, for each $k \in [\beta], a \in \{0,1\}$.

Game $\mathsf{G}_{v,1}$ for $v \in \{0,1\}^{d-1}$ such that $v = i$: This is the same as $\mathsf{G}_{v,0}$ except that we compute $\widetilde{C}_v$ and $\mathsf{lab}^{(v)}$ using the simulator of bGC as follows

$$\mathsf{ct}'_v \leftarrow \mathsf{BBE.Enc}(\mathsf{crs}_i, h_v, \mathbf{M}^{(b)}; r^{(v)})$$
$$(\widetilde{C}_v, \mathsf{lab}^{(v)}) \leftarrow \mathsf{bGC.Sim}(1^\lambda, 1^{|C_v[\mathsf{crs}_i, h_v, \mathbf{M}^{(b)}, r^{(v)}]|}, 1^\beta, \mathsf{ct}'_v).$$

We have that for $v \in \{0,1\}^{d-1}$ such that $v = i$, where i is the challenge index, $\mathsf{G}_{v,0} \approx_c \mathsf{G}_{v,1}$ due to the simulation security of bGC (Definition 3.14), because only the collection of labels $\mathsf{lab}^{(v)} := \left(\mathsf{lab}^{(v)}_{k,h_v[k]}\right)_{k\in[\beta]}$ are used.

Game $\mathsf{G}_{v_{\mathbf{next}},0}$ for $v \in \{0,1\}^{d-1}$ such that $v = i$: This is the same as $\mathsf{G}_{v,1}$ except that we replace $\mathbf{M}^{(b)}$ by $\mathbf{M}^{(c)}$ for an independent uniformly random bit $c \xleftarrow{\$} \{0,1\}$. We have that for $v \in \{0,1\}^{d-1}$ such that $v = i$, where i is the challenge index, $\mathsf{G}_{v,1} \approx_c \mathsf{G}_{v_{\text{next}},0}$ due to the adaptive security of BBE (Definition 3.21). This is because of the fact $\mathbf{M}^{(0)}[k, x[(i-1)\lambda + k]] = \mathbf{M}^{(1)}[k, x[(i-1)\lambda + k]]$ for $k \in [\lambda]$, where $\mathbf{M}^{(b)} := \begin{bmatrix} (\mathbf{m}_0^{(b)})^\top & (\mathbf{m}_1^{(b)})^\top \end{bmatrix} \in \{0,1\}^{\lambda\times 2}$ for $b \in \{0,1\}$ are the challenge messages and $x \in \{0,1\}^N$ is the challenge secret key.

We have proved that $\mathsf{G}_{v,0} \approx_c \mathsf{G}_{v,1} \approx_c \mathsf{G}_{v_{\text{next}},0}$ for every $v \in \{0,1\}^{<d-1}$ and for $v \in \{0,1\}^{d-1}$ such that $v = i$, where i is the challenge index. Finally, we observe that in hybrid $\mathsf{G}_{0^d,0}$ the probability that the adversary wins is $\frac{1}{2}$ because the challenge ciphertext received by the adversary is independent of the bit b. This concludes the adaptive security proof.

Adaptive Blindness. The adaptive blindness proof follows along the lines of the previous adaptive security proof. That is, we set $\mathsf{G}_0 = \mathsf{G}_{\epsilon,0}$ to be the original adaptive blindness game (Definition 3.22) where the challenge bit is $b = 0$, and define the hybrids $\mathsf{G}_{v,1}, \mathsf{G}_{v_{\text{next}},0}$ for all $v \in \{0,1\}^{<d-1}$ analogous as in the previous proof, where we have that $\mathsf{G}_{\epsilon,0} \approx_c \mathsf{G}_{\epsilon,1} \approx_c \cdots \approx_c \mathsf{G}_{1^{d-2},1} \approx_c \mathsf{G}_{0^{d-1},0}$ using the same arguments. We also define $\mathsf{G}_{v,1}$ for $v \in \{0,1\}^{d-1}$ such that $v = i$ as in the previous security proof, and we skip the step that changes the message $\mathbf{M}$ (as in the adaptive blindness game $\mathbf{M}$ is already random). Hence, we have here $\mathsf{G}_{v,0} \approx_c \mathsf{G}_{v,1} \equiv \mathsf{G}_{v_{\text{next}},0}$ for $v \in \{0,1\}^{d-1}$ such that $v = i$. Next, we consider two additional hybrids $\mathsf{G}_{v,2}$ and $\mathsf{G}_{v,3}$ for all $v \in \{0,1\}^{<d-1}$ and $v \in \{0,1\}^{d-1}$ such that $v = i$, and define v_{prev} to be the opposite of v_{next}. We define the hybrids in the reverse order, where we have that $\mathsf{G}_{1^{d-1},2} \equiv \mathsf{G}_{1^{d-1},0}$ by definition.

Game $\mathsf{G}_{v,3}$ for $v \in \{0,1\}^{d-1}$ such that $v = i$: This is the same as $\mathsf{G}_{v,2}$ except that instead of computing $\mathsf{ct}_v \leftarrow \mathsf{BBE.Enc}(\mathsf{crs}_{d-1}, h_v, \mathbf{M}; r^{(v)})$, we pick it uniformly at random from the ciphertext space as $\mathsf{ct}'_v \xleftarrow{\$} \mathsf{BBE}.\mathcal{CT}$. We have that for $v \in \{0,1\}^{d-1}$ such that $v = i$, where i is the challenge index, $\mathsf{G}_{v,2} \approx_c \mathsf{G}_{v,3}$. This follows from the adaptive blindness of BBE (Definition 3.22) via the fact that $\mathbf{M}$ is a uniformly random and independent of the rest of the adaptive blindness experiment of BBE′.

Game $\mathsf{G}_{v_{\mathbf{prev}},2}$ for $v \in \{0,1\}^{d-1}$ such that $v = i$: This game is the same as $\mathsf{G}_{v,3}$ except that the simulator's output $(\widetilde{C}_v, \mathsf{lab}^{(v)}) \leftarrow \mathsf{bGC.Sim}(1^\lambda, 1^{|C_v[\mathsf{crs}_j, h_v, \mathbf{M}, r^{(v)}]|}, 1^\beta, \mathsf{ct}'_v)$ is replaced with a uniformly random bit string $(\widetilde{C}_v, \mathsf{lab}^{(v)}) \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{bGC}}} \times (\{0,1\}^\lambda)^\beta$, where ℓ_{bGC} is the bit length of the garbled circuit $\widetilde{C}_v$. We have that for $v \in \{0,1\}^{d-1}$ such that $v = i$, where i is the challenge index, $\mathsf{G}_{v,3} \approx_c \mathsf{G}_{v_{\text{prev}},2}$ due to the blindness of bGC (Definition 3.15). This holds because the ciphertext ct'_v that is input to the simulator bGC.Sim in $\mathsf{G}_{v,3}$ is uniformly random.

Game $\mathsf{G}_{v,3}$ for all $v \in \{0,1\}^{<d-1}$: This is the same as $\mathsf{G}_{v,2}$ except that instead of computing $\mathsf{ct}_v \leftarrow \mathsf{BBE.Enc}(\mathsf{crs}_j, h_v, \mathbf{A}^{(v)}; r^{(v)})$, where $j = |v|$, we pick it uniformly at random from the ciphertext space as $\mathsf{ct}'_v \xleftarrow{\$} \mathsf{BBE}.\mathcal{CT}$. We have that for all $v \in \{0,1\}^{<d-1}$, $\mathsf{G}_{v,2} \approx_c \mathsf{G}_{v,3}$. This follows from the adaptive blindness of BBE (Definition 3.22) via the fact that $\mathbf{A}^{(v)}$ is a uniformly random and independent of the rest of the adaptive blindness experiment of BBE′.

Game $\mathsf{G}_{v_{\mathbf{prev}},2}$ for $v \in \{0,1\}^{<d-1}$: This game is the same as $\mathsf{G}_{v,3}$ except that we replace the simulator's output $(\widetilde{C}_v, \mathsf{lab}^{(v)}) \leftarrow \mathsf{bGC.Sim}(1^\lambda, 1^{|C_v[\mathsf{crs}_j, h_v, \mathbf{A}^{(v)}, r^{(v)}]|}, 1^\beta, \mathsf{ct}'_v)$ with a uniformly random bit string $(\widetilde{C}_v, \mathsf{lab}^{(v)}) \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{bGC}}} \times (\{0,1\}^\lambda)^\beta$. We have that for $v \in \{0,1\}^{<d-1}$, $\mathsf{G}_{v,3} \approx_c \mathsf{G}_{v_{\text{prev}},2}$ due to the blindness of bGC (Definition 3.15). This holds because the ciphertext ct'_v that is input to the simulator bGC.Sim in $\mathsf{G}_{v,3}$ is uniformly random.

We started with the adaptive blindness experiment (Definition 3.22) with challenge bit $b = 0$, and have proven that $\mathsf{G}_{v,2} \approx_c \mathsf{G}_{v,3} \approx_c \mathsf{G}_{v_{\text{prev}},2}$ for $v \in \{0,1\}^{d-1}$ such that $v = i$ and for every $v \in \{0,1\}^{<d-1}$. Hence, we have that

in hybrid $\mathsf{G}_{\epsilon,3}$ we have reached the security experiment with challenge bit $b = 1$. This concludes the proof of Proposition A.3. □

B Laconic pPRIO with Global Setup

B.1 Overview

Our sRFE constructions given in Section 4 require a laconic pPRIO with a deterministic Digest algorithm, which is not satisfied by the existing laconic pPRIO construction of Agrawal et al. [AKY24b]. Since we are in the registration-based setting, a natural way to derandomize the Digest algorithm is to introduce a one-time global setup that produces re-usable randomness in the form of a common reference string (CRS). This leads to our new primitive called *laconic pPRIO with global setup*, which extends the definition of (plain) laconic pPRIO with an additional Setup algorithm, that on input the security parameter λ outputs a re-usable crs (see Definition 3.41). Then, the *deterministic* Digest algorithm on input the crs and a domain $X = \{X_i\}_{i\in[N]}$ outputs a digest dig. We can construct such a laconic pPRIO with global setup by modifying the plain laconic pPRIO construction from [AKY24b] as detailed below.

Agrawal et al. [AKY24b] constructed laconic pPRIO by combining (plain) pPRIO (Section 3.13) with blind garbled circuit (bGC, Section 3.7) and blind batch encryption (BBE, Section 3.8). Given these primitives, the laconic pPRIO construction from [AKY24b] is as follows (see Section 3.14 for the syntax). To restrict the circuit evaluation to the input domain X, they treat X as the secret key of the BBE scheme, and inside the Digest algorithm they use the BBE.Gen algorithm to compute the hash h of X, which becomes the digest itself. To obfuscate a circuit E, they provide a pPRIO obfuscation of another circuit E', which has both the digest and circuit E hardcoded. The circuit E' garbles the circuit E and generates BBE encryptions of its labels. Then, during the evaluation procedure, given the input domain X, which acts as the secret key, one can decrypt the BBE ciphertexts to recover the labels corresponding to X, and finally, evaluate the garbled circuit. In a bit more detail, the scheme is as follows:

$\mathsf{LprIO.Digest}(1^\lambda, X)$**:** Samples $\mathsf{crs} \leftarrow \mathsf{BBE.Setup}(1^\lambda, 1^{\ell N})$, compresses the input domain $X = \{X_i \in \{0,1\}^\ell\}_{i\in[N]}$ by hashing it as $h := \mathsf{BBE.Gen}(\mathsf{crs}, X)$, and outputs $\mathsf{dig} := (\mathsf{crs}, h)$.

$\mathsf{LprIO.Obf}(\mathsf{dig}, E)$**:** Outputs a pPRIO obfuscation of the circuit $E'[\mathsf{dig}, E]$. The circuit $E'[\mathsf{dig}, E]$ takes as input an index $i \in [N]$, computes the garbled circuit $\widetilde{E}_i$ and input labels $\mathsf{lab}_{i,j,b}$ for $j \in [\ell], b \in \{0,1\}$, performs BBE encryption of the labels $(\mathsf{lab}_{i,j,0}, \mathsf{lab}_{i,j,1})$ using the crs and public key h that forms the digest dig, and outputs the BBE ciphertexts along with the garbled circuit $\widetilde{E}_i$.

$\mathsf{LprIO.Eval}(\widehat{E}'[\mathsf{dig}, E], X)$**:** It first runs the pPRIO evaluation of $\widehat{E}'[\mathsf{dig}, E]$ on index $i \in [N]$ to obtain the garbled circuit $\widetilde{E}_i$ and BBE ciphertexts $\mathsf{ct}_{i,j}$ for $j \in [\ell]$, and then, runs BBE decryption using X and index i to obtain the labels $\{\mathsf{lab}_{i,j}\}_{j\in[\ell]}$ corresponding to X_i. Finally, it runs bGC evaluation on $\widetilde{E}_i$ and $\{\mathsf{lab}_{i,j}\}_{j\in[\ell]}$ to obtain $E(X_i)$, for all $i \in [N]$.

It is easy to see that the above LprIO.Digest algorithm is *randomized* because the underlying BBE.Setup algorithm is randomized. Hence, in order to obtain a *deterministic* LprIO.Digest algorithm, that can be used inside the deterministic sRFE.Agg algorithm, we need to move CRS generation outside of LprIO.Digest, and hence, the need for a *global setup*. For this reason, we define an additional (randomized) algorithm called LprIO.Setup, which simply runs BBE.Setup algorithm to obtain crs and uses it as the CRS of the laconic pPRIO with global setup scheme. To accommodate this change, we also modify all the other algorithms to take the crs as an input as well.

Unfortunately, at this point we face another issue. The existing BBE schemes, such as from [BLSV18, AKY24b], are *bounded*, and require us to fix the length of the secret key during the BBE.Setup algorithm. In the aforedescribed (plain) laconic pPRIO scheme of [AKY24b] this is not an issue since the secret key X is passed as an input to the LprIO.Digest algorithm, and hence, we can run BBE.Setup inside LprIO.Digest with

a parameter depending on the length of X. However, in our laconic pPRIO with global setup scheme we have to run BBE.Setup before running LprIO.Digest, and hence, before learning the input domain X or its length. In order to circumvent this issue, we use the *unbounded* BBE scheme that we previously developed in Section A, which during the BBE.Setup algorithm does not fix the length of the secret keys, and consequently, can support arbitrary (polynomial) length secret keys.

B.2 Construction

In this section we construct a laconic pPRIO scheme with global setup and a *deterministic* Digest algorithm by extending the plain laconic pPRIO scheme given in [AKY24b].

Construction B.1 (Laconic pPRIO with Global Setup)**.** The construction uses the following building blocks:

- A blind garbling scheme $\mathsf{bGC} = \mathsf{bGC}.(\mathsf{Garble}, \mathsf{Eval}, \mathsf{Sim})$. We assume that the labels and the random coins used by $\mathsf{bGC}.\mathsf{Garble}$ are in $\{0,1\}^\lambda$. The former is guaranteed by Definition 3.12 and the latter can be achieved without loss of generality by using a PRF to derive longer (pseudo-)random coins if needed. We can instantiate bGC with the required properties assuming one-way functions (Fact 3.17).
- A pseudorandom function $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ with key space, input space and output spaces being $\{0,1\}^\lambda$. The input to the PRF will be of the form $(i \| j \| b)$, where $i \in [N]$, $j \in [\ell]$ and $b \in \{0,1\}$. As N and ℓ are polynomials in λ, the input space of the PRF of size $2^\lambda > \mathsf{poly}(\lambda)$ is large enough to embed such inputs via an appropriate encoding. PRF can be constructed assuming one-way functions.
- An unbounded blind batch encryption scheme $\mathsf{BBE} = \mathsf{BBE}.(\mathsf{Setup}, \mathsf{Gen}, \mathsf{SingleEnc}, \mathsf{SingleDec})$ with message space $\mathcal{M} = \{0,1\}^\lambda$ and ciphertext space $\mathsf{BBE}.\mathcal{CT} = \{0,1\}^{\ell_{\mathsf{ct}}}$. Without loss of generality, we assume that the random coins used by SingleEnc are in $\{0,1\}^\lambda$. We also require that BBE.Gen be determinsitc. We note that our BBE construction from Section A satisfies these properties (see Theorem A.4).
- A pPRIO scheme $\mathsf{pPRIO} = \mathsf{pPRIO}.(\mathsf{Obf}, \mathsf{Eval})$ for polynomial-size circuits. We can build pPRIO assuming prFE and LWE (Fact 3.40).

The construction is as follows:

$\mathsf{Setup}(1^\lambda)$**:** Run

$$\mathsf{crs} \leftarrow \mathsf{BBE.Setup}(1^\lambda),$$

and output crs.

$\mathsf{Digest}(\mathsf{crs}, X = \{X_i\}_{i\in[N]})$**:** Compute

$$h := \mathsf{BBE.Gen}(\mathsf{crs}, (X_1 \| \cdots \| X_N) \otimes \mathbf{1}_{1\times\lambda}),$$

where $(X_1 \| \cdots \| X_N) \in \{0,1\}^{\ell N}$ is the concatenation of the bit strings $X_1, \ldots, X_N \in \{0,1\}^\ell$ and $\mathbf{1}_{1\times\lambda}$ denotes an all 1 vector of length λ, and output $\mathsf{dig} := (h, N, \ell)$.

$\mathsf{Obf}(\mathsf{crs}, \mathsf{dig}, C)$**:** We divide the obfuscation algorithm into two steps which do the following.

$\mathsf{ObfOff}(\mathsf{crs}, 1^S)$**:** On input a common reference string crs and a size bound S, run

$$(\widehat{C}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{pPRIO.ObfOff}(1^\lambda, 1^{S'}),$$

where $S' = \mathsf{poly}(S, \lambda)$ is the maximum circuit size of the circuit $C'[\mathsf{crs}, \mathsf{dig}, C, \mathsf{sd}]$ defined in Figure 18, where the size of C is bounded by S. Output $(\widehat{C}_{\mathsf{off}}, \mathsf{st})$.

Input: an index $i \in [N]$

Output:
- a set of BBE ciphertexts $\{\mathsf{ct}_{i,j}\}_{j\in[\ell]}$
- a garbled circuit $\widetilde{C}_i$

Hardwired Values:
- a BBE common reference string crs
- a pPRIO digest $\mathsf{dig} := (h, N, \ell)$
- a circuit C of size at most S
- a PRF seed $\mathsf{sd} \in \{0,1\}^\lambda$

- Compute $R_i := \mathsf{PRF}(\mathsf{sd}, (i \,\|\, 1 \,\|\, 0))$ and $S_{i,j} := \mathsf{PRF}(\mathsf{sd}, (i \,\|\, j \,\|\, 1))$ for $j \in [\ell]$.
- Compute

$$\left(\{\mathsf{lab}_{i,j,b}\}_{j\in[\ell],b\in\{0,1\}}, \widetilde{C}_i\right) \leftarrow \mathsf{bGC.Garble}(1^\lambda, C; R_i)$$
$$\mathsf{ct}_{i,j} \leftarrow \mathsf{BBE.SingleEnc}\left(\mathsf{crs}, h, (i-1)\ell + j, (\mathsf{lab}_{i,j,0}, \mathsf{lab}_{i,j,1}); S_{i,j}\right) \text{ for } j \in [\ell].$$

- Output $(\{\mathsf{ct}_{i,j}\}_{j\in[\ell]}, \widetilde{C}_i)$.

Figure 18: Definition of the circuit $C'[\mathsf{crs}, \mathsf{dig}, C, \mathsf{sd}]$.

$\mathsf{ObfOn}(\mathsf{crs}, \mathsf{st}, \mathsf{dig}, C)$**:** Parse $\mathsf{dig} := (h, N, \ell)$, sample $\mathsf{sd} \xleftarrow{\$} \{0,1\}^\lambda$, construct the circuit $C'[\mathsf{crs}, \mathsf{dig}, C, \mathsf{sd}]$ as in Figure 18 and compute

$$\widehat{C}_{\mathsf{on}} \leftarrow \mathsf{pPRIO.ObfOn}(\mathsf{st}, C'[\mathsf{crs}, \mathsf{dig}, C, \mathsf{sd}]).$$

Output $\widehat{C}_{\mathsf{on}}$.

The final output of $\mathsf{Obf}(\mathsf{crs}, \mathsf{dig}, C)$ is $\widehat{C} := (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on}})$.

$\mathsf{Eval}(\mathsf{crs}, X, \widehat{C})$**:** Parse $X = \{X_i\}_{i\in[N]}$ and run

$$y_i := \mathsf{pPRIO.Eval}(\widehat{C}, i) \quad \text{for } i \in [N].$$

Parse $y_i := \left(\{\mathsf{ct}_{i,j}\}_{j\in[\ell]}, \widetilde{C}_i\right)$ for each $i \in [N]$ and compute

$$\mathsf{lab}_{i,j} := \mathsf{BBE.SingleDec}(\mathsf{crs}, X, \ell(i-1) + j, \mathsf{ct}_{i,j}) \quad \text{for } i \in [N], j \in [\ell].$$

Set $\mathsf{lab}_i := \{\mathsf{lab}_{i,j}\}$ and compute

$$z_i := \mathsf{bGC.Eval}(\mathsf{lab}_i, \widetilde{C}_i) \quad \text{for } i \in [N].$$

Output $\{z_i\}_{i\in[N]}$.

Proposition B.2 (Correctness and Compactness)**.** *The laconic pPRIO scheme with global setup in Construction B.1 is correct and compact. Specifically, it has the following parameters:*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda), \quad |\mathsf{dig}| = O(\lambda), \quad |\widehat{C}_{\mathsf{off}}| = \mathsf{poly}(S, \lambda), \quad |\widehat{C}_{\mathsf{on}}| = \mathsf{poly}(S, \lambda).$$

Proposition B.3 (Security)**.** *Assume* pPRIO *is secure (Definition 3.39),* BBE *is adaptively secure and adaptively blind (Definitions 3.21 and 3.22),* bGC *is simulation secure and blind (Definitions 3.14 and 3.15) and* PRF *is secure. Then, Construction B.1 satisfies security as per Definition 3.44.*

The proofs of Propositions B.2 and B.3 are provided in Sections B.3 and B.4. We summarize our result in the following theorem.

Theorem B.4. *Assuming LWE and prFE, there exists a secure laconic pPRIO scheme with global setup and deterministic* Digest *algorithm satisfying*

$$|\mathsf{crs}| = \mathsf{poly}(\lambda), \quad |\mathsf{dig}| = O(\lambda), \quad |\widehat{C}_{\mathsf{off}}| = \mathsf{poly}(S,\lambda), \quad |\widehat{C}_{\mathsf{on}}| = \mathsf{poly}(S,\lambda),$$

where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{dig} \leftarrow \mathsf{Digest}(\mathsf{crs}, X = \{X_i\}_{i\in[N]})$, $\widehat{C} := (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on}}) \leftarrow \mathsf{Obf}(\mathsf{crs}, \mathsf{dig}, C)$ *for a given circuit* $C\colon \{0,1\}^\ell \to \{0,1\}^{\ell'}$*, whose size is bounded by* $S = S(\lambda)$.

B.3 Proof of Correctness and Compactness

Proof of Proposition B.3. We argue that Construction B.1 is correct and compact.

Correctness. For crs, $X = \{X_i\}_{i\in[N]}$ and $\widehat{C} := (\widehat{C}_{\mathsf{off}}, \widehat{C}_{\mathsf{on}})$, we have that

$$\mathsf{crs} \leftarrow \mathsf{BBE.Setup}(1^\lambda)\,, \quad (\widehat{C}_{\mathsf{off}}, \mathsf{st}) \leftarrow \mathsf{pPRIO.ObfOff}(1^\lambda, 1^S)\,, \quad \widehat{C}_{\mathsf{on}} \leftarrow \mathsf{pPRIO.ObfOn}(\mathsf{st}, C'[\mathsf{crs}, \mathsf{dig}, C, \mathsf{sd}])\,.$$

From the correctness of pPRIO and the definition of the circuit $C'[\mathsf{crs}, \mathsf{dig}, C, \mathsf{sd}]$, we have

$$\mathsf{pPRIO.Eval}(\widehat{C}, i) := y_i := \left(\{\mathsf{ct}_{i,j}\}_{j\in[\ell]}, \widetilde{C}_i\right) \quad \text{for } i \in [N],$$

where

$$\mathsf{ct}_{i,j} \leftarrow \mathsf{BBE.SingleEnc}\left(\mathsf{crs}, h, (i-1)\ell + j, (\mathsf{lab}_{i,j,0}, \mathsf{lab}_{i,j,1}); S_{i,j}\right)$$
$$\left(\{\mathsf{lab}_{i,j,b}\}_{j\in[\ell], b\in\{0,1\}}, \widetilde{C}_i\right) \leftarrow \mathsf{bGC.Garble}\left(1^\lambda, C; R_i\right)\,.$$

By the perfect correctness of BBE we get that

$$\mathsf{BBE.SingleDec}(\mathsf{crs}, X, \ell(i-1)+j, \mathsf{ct}_{i,j}) := \mathsf{lab}_{i,j} \quad \text{for } i \in [N] \text{ and } j \in [\ell].$$

Let $\mathsf{lab}_i := \{\mathsf{lab}_{i,j}\}$, which are the labels corresponding to input X_i for $i \in [N]$. Then, from the correctness of bGC and definition of C_i, we get

$$\mathsf{bGC.Eval}(\mathsf{lab}_i, \widetilde{C}_i) := C(X_i),$$

as expected.

Compactness. We make the following observations:

1. Instantiating BBE as in Theorem A.4, we have that $|\mathsf{crs}| = \mathsf{poly}(\lambda)$ and $|\mathsf{ct}| = \mathsf{poly}(\lambda, \log N, \log \ell)$, which are bounded by a fixed polynomial $\mathsf{poly}(\lambda)$ for any polynomially bounded N and ℓ.
2. Instantiating pPRIO as in Fact 3.40, we have that $|\widehat{C}_{\mathsf{off}}| = \mathsf{poly}(S,\lambda)$ and $|\widehat{C}_{\mathsf{on}}| = \mathsf{poly}(S,\lambda)$.

Hence, it follows that our laconic pPRIO with global setup scheme satisfies

$$|\mathsf{crs}| = \mathsf{poly}(\lambda),\ |\mathsf{dig}| = O(\lambda),\ |\widehat{C}_{\mathsf{off}}| = \mathsf{poly}(S,\lambda),\ |\widehat{C}_{\mathsf{on}}| = \mathsf{poly}(S,\lambda).$$

□

B.4 Proof of Security

Proof of Proposition B.3. Sample $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ and invoke the sampler $\mathsf{Samp}(1^\lambda, \mathsf{crs})$ that outputs

$$\left(\mathsf{aux}, 1^S, \left\{X_k = \{X_{k,i} \in \{0,1\}^{\ell_k}\}_{i\in[N_k]}\right\}_{k\in[Q]}, \{C_k\}_{k\in[Q]}\right)\,,$$

where $C_k\colon \{0,1\}^{\ell_k} \to \{0,1\}^{\ell'_k}$. In order to prove the theorem we have to show that

$$\left(\mathsf{aux}, \mathsf{crs}, \widehat{C}_{\mathsf{off}}, \{X_k, \widehat{C}_{\mathsf{on},k}\}_{k\in[Q]}\right) \approx_c \left(\mathsf{aux}, \mathsf{crs}, \widehat{C}_{\mathsf{off}}, \{X_k, \delta_k \xleftarrow{\$} \mathcal{O}_{\mathsf{on}}\}_{k\in[Q]}\right)$$

holds assuming

$$\left(\mathsf{aux}, 1^S, \left\{X_k, \{C_k(X_{k,i})\}_{i\in[N_k]}\right\}_{k\in[Q]}\right) \approx_c \left(\mathsf{aux}, 1^S, \left\{X_k, \{\Delta_{k,i}\}_{i\in[N_k]}\right\}_{k\in[Q]}\right), \tag{33}$$

where $\mathcal{O}_{\mathsf{on}}$ is the co-domain of ObfOn algorithm.

We recall that in our construction $\widehat{C}_{\mathsf{off}}$ and $\widehat{C}_{\mathsf{on}}$ are outputs of pPRIO.ObfOff and pPRIO.ObfOn algorithms, respectively. To prove that $\{\widehat{C}_{\mathsf{on},k}\}_{k\in[Q]}$ is pseudorandom, we invoke the security of pPRIO scheme with respect to a sampler $\mathsf{Samp}_{\mathsf{pPRIO}}(1^\lambda)$, which first samples $\mathsf{crs} \leftarrow \mathsf{BBE.Setup}(1^\lambda)$, invokes $\mathsf{Samp}(1^\lambda, \mathsf{crs})$ to obtain

$$\left(\mathsf{aux}, 1^S, \left\{X_k = \{X_{k,i} \in \{0,1\}^{\ell_k}\}_{i\in[N_k]}\right\}_{k\in[Q]}, \{C_k\}_{k\in[Q]}\right),$$

and finally outputs

$$\left(1^{N_1+\cdots+N_Q}, 1^S, \mathsf{aux}_{\mathsf{pPRIO}} := \left(\mathsf{aux}, \mathsf{crs}, \{X_k\}_{k\in[Q]}\right), \{C'_k[\mathsf{crs}, \mathsf{dig}_k, C_k, \mathsf{sd}_k]\}_{k\in[Q]}\right),$$

where $h_k := \mathsf{BBE.Setup}(\mathsf{crs}, X_{1,k} \| \cdots \| X_{N_k,k})$, $\mathsf{dig}_k := (h_k, N_k, \ell_k)$ and $\mathsf{sd}_k \xleftarrow{\$} \{0,1\}^\lambda$. By the security of pPRIO, it suffices to prove

$$\left(\mathsf{aux}, \mathsf{crs}, \{X_k\}_{k\in[Q]}, \{C'_k[\mathsf{crs}, \mathsf{dig}_k, C_k, \mathsf{sd}_k](i)\}_{k\in[Q], i\in[N_k]}\right) \approx_c \left(\mathsf{aux}, \mathsf{crs}, \{X_k\}_{k\in[Q]}, \{\gamma_{k,i}\}_{k\in[Q], i\in[N_k]}\right), \tag{34}$$

where $C'_k[\mathsf{crs}, \mathsf{dig}_k, C_k, \mathsf{sd}_k](i) := (\{\mathsf{ct}_{k,i,j}\}_{j\in[\ell_k]}, \widetilde{C}_{k,i})$, for $i \in [N_k]$, $k \in [Q]$ and $\gamma_{k,i} \xleftarrow{\$} \mathsf{BBE}.\mathcal{CT}^{\ell_k} \times \{0,1\}^{\ell_{\mathsf{bGC},k}}$, such that $\ell_{\mathsf{bGC},k}$ is the bit length of the garbled circuit $\widetilde{C}_{k,i}$. To prove Equation (34), we consider the following series of hybrids.

Game G_0: This is the left side distribution from Equation (34), which by expanding the circuit description we can rewrite as

$$\left(\mathsf{aux}, \mathsf{crs}, \left\{X_k, \left\{\{\mathsf{ct}_{k,i,j}\}_{j\in[\ell_k]}, \widetilde{C}_{k,i}\right\}_{i\in[N_k]}\right\}_{k\in[Q]}\right),$$

where

$$\begin{aligned}\left(\{\mathsf{lab}_{k,i,j,b}\}_{j\in[\ell_k], b\in\{0,1\}}, \widetilde{C}_{k,i}\right) &\leftarrow \mathsf{bGC.Garble}\left(1^\lambda, C_k; R_{k,i}\right)\\ \mathsf{ct}_{k,i,j} &\leftarrow \mathsf{BBE.SingleEnc}\left(\mathsf{crs}, h_k, (i-1)\ell_k + j, (\mathsf{lab}_{k,i,j,0}, \mathsf{lab}_{k,i,j,1}); S_{k,i,j}\right)\end{aligned}$$

for $R_{k,i} := \mathsf{PRF}(\mathsf{sd}_k, (i \| 1 \| 0))$ and $S_{k,i,j} := \mathsf{PRF}(\mathsf{sd}_k, (i \| j \| 1))$.

Game G_1: This is the same as G_0 except that we sample $R_{k,i}, S_{k,i,j} \xleftarrow{\$} \{0,1\}^\lambda$ for all $j \in [\ell_k]$, $i \in [N_k]$ and $k \in [Q]$. We have that $\mathsf{G}_0 \approx_c \mathsf{G}_1$ due to the security of PRF (Definition 3.11).

Game G_2: This is the same as G_1 except that we compute BBE ciphertext $\mathsf{ct}_{k,i,j}$ as follows

$$\mathsf{ct}_{k,i,j} \leftarrow \mathsf{BBE.SingleEnc}\left(\mathsf{crs}, h_k, (i-1)\ell_k + j, (\widetilde{\mathsf{lab}}_{k,i,j,0}, \widetilde{\mathsf{lab}}_{k,i,j,1}); S_{k,i,j}\right),$$

where $\widetilde{\mathsf{lab}}_{k,i,j,b} = \mathsf{lab}_{k,i,j,b}$ if $b = X_{k,i}[j]$, and $\widetilde{\mathsf{lab}}_{k,i,j,b} \xleftarrow{\$} \{0,1\}^\lambda$ otherwise. Here, $X_{k,i}[j]$ denotes the j-th bit of $X_{k,i}$. We have that $\mathsf{G}_1 \approx_c \mathsf{G}_2$ due to the adaptive security of BBE (Definition 3.21), because the decryption outputs the labels corresponding to the j-th bit of $X_{k,i}$ and we only substitute $\widetilde{\mathsf{lab}}_{k,i,j,b} \xleftarrow{\$} \{0,1\}^\lambda$ when $b \neq X_{k,i}[j]$.

Game G_3: This is the same as G_2 except that we compute the garbled circuit $\widetilde{C}_{k,i}$ and labels $\widetilde{\mathsf{lab}}_{k,i,j}$ as follows

$$\left(\{\mathsf{lab}_{k,i,j,b}\}_{j\in[\ell_k], b\in\{0,1\}}, \widetilde{C}_{k,i}\right) \leftarrow \mathsf{bGC.Sim}\left(1^\lambda, 1^{|C_k|}, 1^{\ell_k}, C_k(X_{k,i})\right).$$

Then, as before, we set $\widetilde{\mathsf{lab}}_{k,i,j,b} = \mathsf{lab}_{k,i,j,b}$ if $b = X_{k,i}[j]$ and $\widetilde{\mathsf{lab}}_{k,i,j,b} \xleftarrow{\$} \{0,1\}^\lambda$ otherwise. We have that $\mathsf{G}_2 \approx_c \mathsf{G}_3$ due to the simulation security of bGC (Definition 3.14). For this, we note that only the labels $\{\mathsf{lab}_{k,i,j,X_{k,i}[j]}\}_{i,j,k}$ are required for simulating G_2 and the labels $\{\mathsf{lab}_{k,i,j,1-X_{k,i}[j]}\}_{i,j,k}$ are not necessary.

Game G_4**:** This is the same as G_3 except that we compute the garbled circuit $\widetilde{C}_{k,i}$ and labels $\mathsf{lab}_{k,i,j}$ as follows

$$\left(\{\mathsf{lab}_{k,i,j,b}\}_{j\in[\ell_k],b\in\{0,1\}}, \widetilde{C}_{k,i}\right) \leftarrow \mathsf{bGC.Sim}\left(1^{\lambda}, 1^{|C_k|}, 1^{\ell_k}, \Delta_{k,i}\right),$$

where $\Delta_{k,i} \xleftarrow{\$} \{0,1\}^{\ell'_k}$ is a uniformly random bit string. We have that $\mathsf{G}_3 \approx_c \mathsf{G}_4$ by Equation (33), as it implies $\{C_k(X_{k,i})\}_{i\in[N_k]} \approx_c \{\Delta_{k,i}\}_{i\in[N_k]}$, for $k \in [Q]$, given $\left(\mathsf{aux}, \mathsf{crs}, 1^S, \{X_k\}_{k\in[Q]}\right)$.

Game G_5**:** This is the same as G_4 except that we sample $\widetilde{C}_{k,i} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{bGC},k}}$ and $\mathsf{lab}_{k,i,j} \xleftarrow{\$} \{0,1\}^{\lambda}$ as uniformly random bit strings. We have that $\mathsf{G}_4 \approx_c \mathsf{G}_5$ due to the blindness of bGC (Definition 3.15). Concretely, in G_4 the simulator bGC.Sim takes as input uniformly random bit strings $\Delta_{k,i}$, and hence, by the blindness property of bGC we can replace the output of bGC.Sim with a completely random bit string.

Game G_6**:** This is the same as G_5 except that we sample $\mathsf{ct}_{k,i,j} \xleftarrow{\$} \mathsf{BBE}.\mathcal{CT}$ for all $i \in [N_k]$, $j \in [\ell_k]$, $k \in [Q]$. We have that $\mathsf{G}_5 \approx_c \mathsf{G}_6$ due to the adaptive blindness of BBE (Definition 3.22), because from G_5 we have that $\mathsf{ct}_{k,i,j}$ encrypts random bit strings $\widetilde{\mathsf{lab}}_{k,i,j,0} \xleftarrow{\$} \{0,1\}^{\lambda}$ and $\widetilde{\mathsf{lab}}_{k,i,j,1} \xleftarrow{\$} \{0,1\}^{\lambda}$. The view of the adversary after G_6 is as follows

$$\left(\mathsf{aux}, \mathsf{crs}, \left\{X_k, \left\{\{\mathsf{ct}_{k,i,j} \xleftarrow{\$} \mathsf{BBE}.\mathcal{CT}\}_{j\in[\ell_k]}, \widetilde{C}_{k,i} \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{bGC},k}}\right\}_{i\in[N_k]}\right\}_{k\in[Q]}\right).$$

By rearranging the terms we observe that the distribution in G_6 corresponds to the right side distribution of Equation (34). This concludes the proof of Proposition B.3. □