Functional Encryption: Deterministic to Randomized Functions from Simple Assumptions

Shashank Agrawal[*] David J. Wu[†]

Abstract

Functional encryption (FE) enables fine-grained control of encrypted data by allowing users to compute only those functions for which they have a key. Until recently, FE schemes could only support deterministic functions, but this changed with the work of Goyal et al. (TCC 2015), who formalized the study of functional encryption for the more general class of *randomized functionalities* with *security against malicious encrypters*. While public-key functional encryption can be constructed from many assumptions, the only known construction for randomized functionalities in the public-key setting is due to Goyal et al. and achieves selective security from indistinguishability obfuscation.

Our key contribution in this work is a *generic transformation* that converts any general-purpose, public-key FE scheme for deterministic functionalities into one that supports randomized functionalities. Our transformation uses the underlying FE scheme in a black-box way and can be instantiated using very standard number-theoretic assumptions (for instance, the DDH and RSA assumptions suffice). When applied to existing FE constructions, we obtain several *adaptively-secure*, public-key functional encryption schemes for randomized functionalities with security against malicious encrypters from many different assumptions such as concrete assumptions on multilinear maps, indistinguishability obfuscation, and in the bounded-collusion setting, the existence of public-key encryption, together with standard number-theoretic assumptions.

Additionally, we introduce a new, stronger definition for malicious security as the existing one falls short of capturing an important class of malleability attacks. In realizing this definition, our compiler combines ideas from disparate domains like related-key security for pseudorandom functions and deterministic encryption in a novel way. We believe that our techniques could be useful in expanding the scope of new variants of functional encryption (e.g., multi-input, hierarchical, and others) to support randomized functionalities.

[*]University of Texas at Austin. Email: `sagrawal@cs.utexas.edu`. Part of this work was done while the author was at the University of Illinois, Urbana-Champaign.

[†]Stanford University. Email: `dwu4@cs.stanford.edu`. This work was supported by NSF, DARPA, the Simons foundation, a grant from ONR, and an NSF Graduate Research Fellowship. Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

Contents

1 Introduction

Traditionally, encryption schemes have provided an all-or-nothing approach to data access: a user who holds the secret key can completely recover the message from a ciphertext while a user who does not hold the secret key learns nothing at all from the ciphertext. In the last fifteen years, numerous paradigms, such as identity-based encryption [Sha84, BF01, Coc01], attribute-based encryption [SW05, GPSW06, BSW07], predicate encryption [BW07, KSW08, LOS$^+$10, OT10], and more have been introduced to enable more fine-grained access control on encrypted data. More recently, the cryptographic community has worked to unify these different paradigms under the general umbrella of functional encryption (FE) [SS10, BSW11, O'N10].

At a high level, an FE scheme enables delegation of decryption keys that allow users to learn specific functions of the data, and nothing else. More precisely, given a ciphertext for a message x and a secret key for a function f, one can only learn the value $f(x)$. In the last few years, numerous works have explored different security notions [BSW11, O'N10, AGVW13, BO13, BF13, AAB$^+$15, AAP15] as well as constructions from a wide range of assumptions [GVW13, ABF$^+$13, DIJ$^+$13, GKP$^+$13, GGH$^+$13, Wat15, ABSV15, AS15]. Until very recently, the vast majority of work in functional encryption has focused on *deterministic functionalities*, i.e., on schemes that issue keys for deterministic functions only. However, there are many natural scenarios where the functionality of interest is more naturally captured by a *randomized function*.

Supporting randomized functionalities. Consider the problem of conducting audits on an encrypted database, an interesting question first raised by Goyal et al. [GJKS15]. For instance, a government oversight agency is interested in auditing banks to check for improper behavior. On the one hand, it is unsafe for a bank to give its entire database to an external party, and on the other, agencies do not have time to verify every record. Ideally, banks would encrypt their databases before providing access, and an auditor would be able to sample and verify a few *randomly* chosen records from each database. We can satisfy this seemingly conflicting set of requirements with an FE scheme for randomized functionalities. Specifically, the bank encrypts its database under the FE scheme, and the auditor is given a secret key that allows it to sample records from the database.

Constructing an encryption scheme that supports this sampling functionality is a nontrivial task. There are several aspects of this problem that make it challenging. For instance, if the auditor has two keys, then applying each key to the database should give an *independent* draw from the database. Thus, with multiple keys, the auditor can sample a random subset of records. Furthermore, applying the same key again and again to a particular database should not reveal multiple records. On the flip side, if the same key is applied to two different databases, the auditor should get an independent sample from each.

There are several other areas where the ability to issue keys for randomized functions is very useful, such as differentially-private data release. Motivated by these applications, Alwen et al. [ABF$^+$13] and Goyal et al. [GJKS15] were the first to formally study the problem of FE for randomized functionalities. Here, a secret key for a randomized function f and an encryption of a message x should produce *a single sample from the output distribution of* $f(x)$. Moreover, given a collection of secret keys $\mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_n}$ for functions $f_1, \ldots, f_n$, and ciphertexts $\mathsf{ct}_{x_1}, \ldots, \mathsf{ct}_{x_n}$ corresponding to messages $x_1, \ldots, x_n$, where neither the functions nor the messages need to be distinct, each secret key sk_{f_i} and ciphertext ct_{x_j} should reveal an *independent* draw from the output distribution of $f_i(x_j)$, and nothing more.

In supporting randomized functionalities, handling *malicious encrypters* is a central issue: a malicious encrypter may construct a ciphertext for a message x such that when decrypted with a key for f, the resulting distribution differs significantly from that of $f(x)$. For instance, in the auditing application discussed earlier, a malicious bank could manipulate the randomness used to sample records in its database, thereby compromising the integrity of the audit. We refer to [GJKS15] for a more thorough discussion on the importance of handling malicious encrypters.

1.1 Our Contributions

To date, the only known construction of public-key FE for randomized functionalities secure against malicious encrypters is due to Goyal et al. [GJKS15] and relies on indistinguishability obfuscation ($i\mathcal{O}$) [BGI$^+$01, GGH$^+$13]. However, $i\mathcal{O}$ is not a particularly appealing assumption since the security of existing $i\mathcal{O}$ constructions either rely on an exponential number of assumptions [BR14, BGK$^+$14, PST14, Zim15, AB15], or on a polynomial set of assumptions but with an exponential loss in the security reduction [GLW14, GLSW15]. This shortcoming may even be inherent, as suggested by [GGSW13]. Moreover, numerous recent attacks on multilinear maps (the underlying primitive on which all candidate constructions $i\mathcal{O}$ are based) [CHL$^+$15, BWZ14, CGH$^+$15, CLLT15, HJ16, CFL$^+$16, CJL16, MSZ16] have reduced the community's confidence in the security of existing constructions of $i\mathcal{O}$.

On the other hand, functional encryption for deterministic functions can be realized from a variety of assumptions such as the existence of public-key encryption [SS10, GVW12], learning with errors [GKP$^+$13], indistinguishability obfuscation [GGH$^+$13, Wat15], multilinear maps [GGHZ16], and more. Thus, there is a very large gap between the assumptions needed to build FE schemes for deterministic functionalities and those needed for randomized functionalities. Hence, it is important to ask:

Does extending public-key FE to support the richer class of randomized functions require strong additional assumptions such as $i\mathcal{O}$?[1]

If there was a general transformation that we could apply to any FE scheme for deterministic functions, and obtain one that supported randomized functions, then we could leverage the extensive work on FE for the former to build FE for the latter with various capabilities and security guarantees. In this paper, we achieve exactly this. We bridge the gap between FE schemes for deterministic and randomized functionalities by showing that any general-purpose FE scheme for deterministic functionalities can be extended to support randomized functionalities with security against malicious encrypters. Our generic transformation applies to any general-purpose FE scheme with perfect correctness and only requires fairly mild additional assumptions (e.g., the decisional Diffie-Hellman (DDH) [Bon98] and the RSA [RSA78, Bon99] assumptions suffice). Moreover, our transformation is tight in the sense that it preserves the security of the underlying FE scheme. Because our transformation relies only on simple additional assumptions, future work in constructing general-purpose FE can primarily focus on handling deterministic functions rather than devising specialized constructions to support randomized functions. We now give an informal statement of our main theorem:

Theorem 1.1 (Informal)**.** *Under standard number-theoretic assumptions, given any general-purpose, public-key functional encryption scheme for deterministic functions, there exists a general-purpose, public-key functional encryption scheme for randomized functions secure against malicious encrypters.*

In this work, we focus on simulation-based notions of security for FE. As shown by several works [BSW11, O'N10, AGVW13], game-based formulations of security are inadequate if the function family under consideration has some computational hiding properties. Moreover, there are generic ways to boost the security

[1]A similar type of question was also raised recently in the context of positional accumulators [OPWW15], FE for deterministic functions [GGHZ16], multi-input functional encryption [BLR$^+$15], and attribute-based fully homomorphic encryption [CM16].

of FE for deterministic functionalities from game-based to best-possible simulation-based [DIJ$^+$13]. Hence, it is not only prudent but also sufficient to design a transformation technique that works on FE schemes with simulation-based security.

Recently, Komargodski et al. [KSY15] considered a variant of the central question we ask in the private-key setting. They show that starting from any "function-private" secret-key FE scheme for deterministic functionalities, they can construct a secret-key FE for randomized functionalities. Combining their transformation with the Brakerski-Segev function-privacy transformation [BS15a], their technique can be applied to any general-purpose FE scheme in the secret-key setting. Due to the reliance on function privacy [SSW09, BRS13], however, the Komargodski et al. transformation does not translate to the public-key setting. Moreover, their transformation does not address the problem of malicious encrypters.

Concrete instantiations. Instantiating Theorem 1.1 with existing FE schemes such as [GVW13, GGHZ16, GGH$^+$13] and applying transformations like [BV16, DNR04, DIJ$^+$13, ABSV15] to boost correctness and/or security, we obtain several new public-key FE schemes for randomized functionalities with *adaptive* simulation-based security against malicious encrypters. For example, if we start with

- the GVW scheme [GVW12], we obtain a scheme secure under bounded collusions assuming the existence of semantically-secure public-key encryption and low-depth pseudorandom generators.
- the GGHZ scheme [GGHZ16], we obtain a scheme with best-possible simulation security relying on the polynomial hardness of concrete assumptions on composite-order multilinear maps [BS02, CLT13, CLT15].
- the GGHRSW scheme [GGH$^+$13], we obtain a scheme with best-possible simulation security from indistinguishability obfuscation.

The second and third schemes above should be contrasted with the one given by Goyal et al. [GJKS15], which achieves *selective* security assuming the existence of $i\mathcal{O}$. We describe these instantiations in greater detail in Section 5.

Security definition. We also propose a strong simulation-based definition for security against malicious encrypters, strengthening the one given by Goyal et al. [GJKS15]. We first give a brief overview of their definition in Section 1.2 and then show why it does not capture an important class of malleability attacks. We also discuss the subtleties involved in extending their definition.

Our techniques. At a very high level, we must balance between two conflicting goals in order to achieve our strengthened security definition. On the one hand, the encryption and key-generation algorithms must be randomized to ensure that the decryption operation induces the correct output distribution, or even more fundamentally, that the scheme is semantically-secure. On the other hand, a malicious encrypter could exploit its freedom to choose the randomness when constructing ciphertexts in order to induce correlations when multiple ciphertexts or keys are operated upon. We overcome this barrier by employing ideas from disparate domains like related-key security for pseudorandom functions and deterministic encryption in a novel way. We discuss our transformation and the tools involved in more detail in Section 1.3.

We believe that our techniques could be used to extend the capability of new variants of functional encryption like multi-input FE [GGG$^+$14, BLR$^+$15], hierarchical or delegatable FE [ABG$^+$13, CGJS15, BS15b], and others so that they can support randomized functionalities with security against malicious encrypters as well.

1.2 Security Against Malicious Encrypters

Simulation security. Informally, simulation security for FE schemes supporting randomized functionalities states that the output of any efficient adversary with a secret key for a randomized function f and an encryption of a message x can be simulated given only $f(x;r)$, where the randomness r used to evaluate f is independently and uniformly sampled. Goyal et al. [GJKS15] extend this notion to include security against malicious encrypters by further requiring that the output of any efficient adversary holding a secret key for a function g and a (possibly dishonestly-generated) ciphertext $\hat{\mathsf{ct}}$ should be simulatable given only $g(\hat{x};r)$, where $\hat{x}$ is a message that is information-theoretically fixed by $\hat{\mathsf{ct}}$, and the randomness r is uniform and unknown to the adversary. This captures the notion that a malicious encrypter is unable to influence the randomness used to evaluate the function during decryption.

More formally, in the simulation-based definitions of security [BSW11, O'N10], an adversary tries to distinguish its interactions in a real world where ciphertexts and secret keys are generated according to the specifications of the FE scheme from its interactions in an ideal world where they are constructed by a simulator given only a minimal amount of information. To model security against malicious encrypters, Goyal et al. give the adversary access to a decryption oracle in the security game (similar to the formulation of IND-CCA2 security [RS92]) that takes as input a *single* ciphertext ct along with a function f. In the real world, the challenger first extracts a secret key sk_f for f and then outputs the decryption of ct with sk_f. In the ideal world, the challenger invokes the simulator on ct. The simulator then outputs a value x (or a special symbol $\perp$), at which point the challenger replies to the adversary with an independently uniform value drawn from the distribution $f(x)$ (or $\perp$).

Limitations of the existing definition. While the definition in [GJKS15] captures security against dishonest encrypters when dealing with deterministic functionalities, it does not fully capture the desired security goals in the randomized setting. Notably, the security definition only considers *one* ciphertext. However, when extending functional encryption to randomized functionalities, we are also interested in the joint distribution of *multiple* ciphertexts and secret keys. Thus, while it is the case that in any scheme satisfying the security definition in [GJKS15], the adversary cannot produce any single ciphertext that decrypts improperly, a malicious encrypter could still produce a collection of ciphertexts such that when the same key is used for decryption, the outputs are correlated. In the auditing application discussed before, it is imperative to prevent this type of attack, for otherwise, the integrity of the audit can be compromised.

Strengthening the definition. A natural way to strengthen Goyal et al.'s definition is to allow the decryption oracle to take in a set of (polynomially-many) ciphertexts along with a function f. In the real world, the challenger extracts a single key sk_f for f and applies the decryption algorithm with sk_f to each ciphertext. In the ideal world, the simulator is given the set of ciphertexts and is allowed to query the evaluation oracle $\mathcal{O}_f$ once for each ciphertext submitted. On each query x, the oracle responds with a fresh evaluation of $f(x)$. This direct extension, however, is too strong, and not achievable by any existing scheme. Suppose that an adversary could efficiently find two ciphertexts $\mathsf{ct}_1 \neq \mathsf{ct}_2$ such that for all secret keys sk, $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}_1) = \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}_2)$, then it can easily distinguish the real and ideal distributions. When queried with $(f, (\mathsf{ct}_1, \mathsf{ct}_2))$, the decryption oracle always replies with two identical values in the real world irrespective of what f is. In the ideal world, however, it replies with two independent values since fresh randomness is used to evaluate f every time.

While we might want to preclude this type of behavior with our security definition, it is also one that arises naturally. For example, in both Goyal et al.'s and our construction, ciphertexts have the form (ct', π) where ct' is the ciphertext component that is actually combined with the decryption key and π is a proof of the well-formedness of ct'. Decryption proceeds only if the proof verifies. Since the proofs are randomized, an adversary can construct a valid ciphertext component ct' and two distinct proofs π_1, π_2 and submit the pair of ciphertexts (ct', π_1) and (ct', π_2) to the decryption oracle. Since π_1 and π_2 do not participate in the decryption process after verification, these two ciphertexts are effectively identical from the perspective of the decryption function. However, as noted above, an adversary that can construct such ciphertexts can trivially distinguish between the real and ideal worlds.

Intuitively, if the adversary submitted the *same* ciphertext multiple times in a decryption query, it does not make sense for the decryption oracle to respond with independently distributed outputs in the ideal experiment. The expected behavior is that the decryption oracle responds with the same value on all identical ciphertexts. In our setting, we generalize the notion of "ciphertext equivalence." In particular, when the adversary submits a decryption query, the decryption oracle in the ideal experiment responds consistently on all equivalent ciphertexts that appear in the query. Formally, we capture this by introducing an efficiently-checkable equivalence relation on the ciphertext space of the FE scheme. For example, if the ciphertexts have the form (ct', π), one valid equivalence relation on ciphertexts is equality of the ct' components. To respond to a decryption query, the challenger first groups the ciphertexts according to their equivalence class, and responds consistently for all ciphertexts belonging to the same class. Thus, without loss of generality, it suffices to just consider adversaries whose decryption queries contain at most one representative from each equivalence class. We provide a more thorough discussion of our strengthened definition in Section 3.

As far as we understand, the Goyal et al. construction remains secure under our strengthened notion of security against malicious encrypters, but it was only shown to be selectively secure assuming the existence of $i\mathcal{O}$ (and one-way functions). Our transformation, on the other hand, provides a *generic* way of building *adaptively-secure* schemes from both $i\mathcal{O}$ as well as plausibly weaker assumptions such as those on composite-order multilinear maps (Section 5). Finally, we note that not all schemes satisfying the Goyal et al. security notion satisfy our strengthened definition. In fact, a simplified version of our transformation yields a scheme secure under their original definition, but not our new definition (Remark 4.2).

1.3 Overview of Our Generic Transformation

Our primary contribution in this work is giving a generic transformation from any simulation-secure general-purpose (public-key) FE scheme[2] for deterministic functionalities to a corresponding simulation-secure (public-key) FE scheme for randomized functionalities. In this section, we provide a brief overview of our generic transformation. The complete construction is given in Section 4.

Derandomization. Our starting point is the generic transformation of Alwen et al. [ABF+13] who use a pseudorandom function (PRF) to "derandomize" functionalities. In their construction, an encryption of a message x consists of an FE encryption of the pair (x, k) where k is a uniformly chosen PRF key. A secret key for a randomized functionality f is constructed by first choosing a random point t in the domain of

[2]Our transformation requires that the underlying FE scheme be *perfectly correct.* Using the transformations in [DNR04, BV16], approximately correct FE schemes can be converted to FE schemes that satisfy our requirement.

the PRF and then extracting an FE secret key for the derandomized functionality $g_t(x,k) = f(x; \mathsf{PRF}(k,t))$, that is, the evaluation of f using randomness derived from the PRF. Evidently, this construction is not robust against malicious encrypters, since by reusing the same PRF key when constructing the ciphertexts, a malicious encrypter can induce correlations in the function evaluations. In fact, the problem is more severe since there could be certain "weak" PRF keys k where the behavior of $\mathsf{PRF}(k,\cdot)$ is easily distinguishable from that of a truly random function.[3]

Secret sharing the PRF key. In our transformation, we start with the same derandomization approach. Since allowing the encrypter full control over the PRF key is problematic, we instead secret share the PRF key across the ciphertext and the decryption key. Suppose the key-space $\mathcal{K}$ of the PRF forms a group under an operation $\diamond$. As before, an encryption of a message x corresponds to an FE encryption of the pair (x,k), but now k is just a single share of the PRF key. To issue a key for f, another random key-share k' is chosen from $\mathcal{K}$. The key sk_f is then an FE key for the derandomized functionality $f(x; \mathsf{PRF}(k \diamond k', x))$. In this scheme, a malicious encrypter is able to influence the PRF key, but does not have full control. However, because the malicious encrypter can induce correlated PRF keys in the decryption queries, the usual notion of PRF security no longer suffices. Instead, we require the stronger property that the outputs of the PRF appear indistinguishable from random even if the adversary observes PRF outputs under *related keys*. Security against related-key attacks (RKA-security) for PRFs has been well-studied [Bih94, Knu93, BK03, BC10, BCM11, LMR14, ABPP14, ABP15] in the last few years, and for our particular application, a variant of the Naor-Reingold PRF is related-key secure for the class of group-induced transformations [BC10].

Applying deterministic encryption. By secret-sharing the PRF key and using a PRF secure against related-key attacks, we obtain robustness against malicious encrypters that only requests the decryption of unique (x,k) pairs (in this case, either k or x is unique, so by related-key security, the output of the PRF appears uniformly random). However, a malicious encrypter can encrypt the same pair (x,k) multiple times, using freshly generated randomness for the base FE scheme each time. Since each of these ciphertexts encrypt the *same* underlying value, in the real world, the adversary receives the same value from the decryption oracle. In the ideal world, the adversary receives independent draws from the distribution $f(x)$. This problem arises because the adversary is able to choose additional randomness when constructing the ciphertexts that does not affect the output of the decryption algorithm. As such, it can construct ciphertexts that induce correlations in the outputs of the decryption process.

To protect against the adversary that encrypts the same (x,k) pair, we note that in the honest-encrypter setting, the messages that are encrypted have high entropy (since the key-share is sampled uniformly at random). Thus, instead of having the adversary choose its randomness for each encryption arbitrarily, we instead force the adversary to derive the randomness from the message. This is similar to what is done when constructing deterministic public-key encryption [BBO07, BFOR08, BS11, FOR12]. Specifically, we sample a one-way permutation h on the key-space of the PRF, set the key-share in the ciphertext to $h(k)$ where k is uniform over $\mathcal{K}$, and then derive the randomness used in the encryption using a hard-core function of h. In addition, we require the adversary to include a non-interactive zero-knowledge (NIZK) argument that each ciphertext is properly constructed. In this way, we guarantee that for each pair (x,k), there is exactly a *single* ciphertext that is valid. By our admissibility requirement, the adversary is required

[3] Recall that the PRF security property only states that for a randomly chosen key, the behavior of $\mathsf{PRF}(k,\cdot)$ looks indistinguishable from that of a truly random function. Thus, it is possible that there exists a negligible fraction of "weak" PRF keys that are easy to find.

to submit distinct ciphertexts (since matching ciphertexts belong to the same equivalence class). Thus, the underlying messages encrypted by each ciphertext in a decryption query necessarily differ in either the key-share or the message component. Security then follows by RKA-security.

2 Preliminaries

For $n \geq 1$, we write $[n]$ to denote the set of integers $\{1, \ldots, n\}$. For bit-strings $a, b \in \{0,1\}^*$, we write $a \| b$ to denote the concatenation of a and b. For a finite set S, we write $x \xleftarrow{\text{R}} S$ to denote that x is sampled uniformly from S. We denote the evaluation of a randomized function f on input x with randomness r by $f(x; r)$. We write $\mathsf{Funs}[\mathcal{X}, \mathcal{Y}]$ to denote the set of all functions mapping from a domain $\mathcal{X}$ to a range $\mathcal{Y}$. We use λ to denote the security parameter. We say a function $f(\lambda)$ is negligible in λ, denoted by $\mathsf{negl}(\lambda)$, if $f(\lambda) = o(1/\lambda^c)$ for all $c \in \mathbb{N}$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We use $\mathsf{poly}(\lambda)$ (or just poly) to denote a quantity whose value is bounded by *some* polynomial in λ.

We now formally define the tools we need to build FE schemes for randomized functionalities with security against malicious encrypters. In Appendix A, we also review the standard definitions of non-interactive zero-knowledge (NIZK) arguments of knowledge [BFM88, FLS90, Gro06, GOS06], and one-way permutations [Gol01].

2.1 RKA-Secure PRFs

We first review the standard definition of a pseudorandom function (PRF) and the notion of related-key security [Bih94, Knu93, BK03, BC10, BCM11, LMR14, ABPP14, ABP15] for PRFs.

Definition 2.1 (Pseudorandom Function [GGM84]). Let $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda \in \mathbb{N}}$, $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$, and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$ be ensembles where $\mathcal{K}_\lambda$, $\mathcal{X}_\lambda$, and $\mathcal{Y}_\lambda$ are finite sets and represent the key-space, domain, and range, respectively. Let $F : \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$ be an efficient computable family of functions. Then F is a PRF if for all efficient non-uniform adversaries $\mathcal{A}$,

$$\left| \Pr\left[k \xleftarrow{\text{R}} \mathcal{K}_\lambda : \mathcal{A}^{F(k,\cdot)}(1^\lambda) = 1 \right] - \Pr\left[f \xleftarrow{\text{R}} \mathsf{Funs}[\mathcal{X}_\lambda, \mathcal{Y}_\lambda] : \mathcal{A}^{f(\cdot)}(1^\lambda) = 1 \right] \right| = \mathsf{negl}(\lambda).$$

Definition 2.2 (RKA-Secure PRF [BK03, BC10]). Let $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda \in \mathbb{N}}$, $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$, and $\mathcal{Y}_\lambda = \{\mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$ be ensembles as in Definition 2.1, and let $F : \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$ be an efficiently computable family of pseudorandom functions. Let $\Phi \subseteq \mathsf{Funs}[\mathcal{K}_\lambda, \mathcal{K}_\lambda]$ be a family of key derivation functions. We say that F is Φ-RKA secure if for all efficient non-uniform adversaries $\mathcal{A}$,

$$\left| \Pr\left[k \xleftarrow{\text{R}} \mathcal{K}_\lambda : \mathcal{A}^{\mathcal{O}(k,\cdot,\cdot)}(1^\lambda) = 1 \right] - \Pr\left[f \xleftarrow{\text{R}} \mathsf{Funs}[\Phi \times \mathcal{X}_\lambda, \mathcal{Y}_\lambda] : \mathcal{A}^{f(\cdot,\cdot)}(1^\lambda) = 1 \right] \right| = \mathsf{negl}(\lambda),$$

where the oracle $\mathcal{O}(k, \cdot, \cdot)$ outputs $F(\phi(k), x)$ on input $(\phi, x) \in \Phi \times \mathcal{X}_\lambda$.

Definition 2.3 (Group Induced Classes [Luc04, BC10]). If the key space $\mathcal{K}$ forms a group under an operation $\diamond$, then the group-induced class $\Phi_\diamond$ is the class of functions $\Phi_\diamond = \left\{ \phi_b : a \in \mathcal{K} \mapsto a \diamond b \mid b \in \mathcal{K} \right\}$.

2.2 Functional Encryption

The notion of functional encryption was first formalized by Boneh et al. [BSW11] and O'Neill [O'N10]. The work of Boneh et al. begins with a natural indistinguishability-based notion of security. They then describe some example scenarios where these game-based definitions of security are inadequate (in the sense that a trivially insecure FE scheme can be proven secure under the standard game-based definition). To address these limitations, Boneh et al. defined a stronger simulation-based notion of security, which has subsequently been the subject of intense study [GVW12, AGVW13, DIJ$^+$13, GKP$^+$13, GJKS15]. In this work, we focus on this stronger security notion.

Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be ensembles where $\mathcal{X}_\lambda$ and $\mathcal{Y}_\lambda$ are finite sets and represent the input and output domains, respectively. Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble where each $\mathcal{F}_\lambda$ is a finite collection of (deterministic) functions from $\mathcal{X}_\lambda$ to $\mathcal{Y}_\lambda$. A functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{KeyGen}, \mathsf{Decrypt})$ for a (deterministic) family of functions $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ with domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ is specified by the following four efficient algorithms:

- **Setup**: $\mathsf{Setup}(1^\lambda)$ takes as input the security parameter λ and outputs a public key $\textsc{mpk}$ and a master secret key $\textsc{msk}$.

- **Encryption**: $\mathsf{Encrypt}(\textsc{mpk}, x)$ takes as input the public key $\textsc{mpk}$ and a message $x \in \mathcal{X}_\lambda$, and outputs a ciphertext ct.

- **Key Generation**: $\mathsf{KeyGen}(\textsc{msk}, f)$ takes as input the master secret key $\textsc{msk}$, a function $f \in \mathcal{F}_\lambda$, and outputs a secret key sk.

- **Decryption**: $\mathsf{Decrypt}(\textsc{mpk}, \mathsf{sk}, \mathsf{ct})$ takes as input the public key $\textsc{mpk}$, a ciphertext ct, and a secret key $\textsc{sk}$, and either outputs a string $y \in \mathcal{Y}_\lambda$, or a special symbol $\perp$. We can assume without loss of generality that this algorithm is deterministic.

First, we state the correctness and security definitions for an FE scheme for deterministic functions.

Definition 2.4 (Perfect Correctness)**.** A functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{KeyGen}, \mathsf{Decrypt})$ for a deterministic function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ with message space $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ is *perfectly correct* if for all $f \in \mathcal{F}_\lambda$, $x \in \mathcal{X}_\lambda$,

$$\Pr[(\textsc{mpk}, \textsc{msk}) \leftarrow \mathsf{Setup}(1^\lambda); \mathsf{Decrypt}(\textsc{mpk}, \mathsf{KeyGen}(\textsc{msk}, f), \mathsf{Encrypt}(\textsc{mpk}, x)) = f(x)] = 1.$$

Our simulation-based security definition is similar to the one in [AGVW13], except that we allow an adversary to submit a vector of messages in its challenge query (as opposed to a single message). Our definition is stronger than the one originally proposed by Boneh et al. [BSW11] because we do not allow the simulator to rewind the adversary. On the other hand, it is weaker than [GVW12, DIJ$^+$13] since the simulator is allowed to program the public parameters and the responses to the pre-challenge secret key queries.

Definition 2.5 (SIM-Security)**.** An FE scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{KeyGen}, \mathsf{Decrypt})$ for a deterministic function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in N}$ with message space $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ is (q_1, q_c, q_2)-SIM-*secure* if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4)$ such that for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ makes at most q_1 oracle queries and $\mathcal{A}_2$ makes at most q_2 oracle queries, the outputs of the following two experiments are computationally indistinguishable:

Experiment $\mathsf{Real}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda)$:
$(\textsc{mpk}, \textsc{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathbf{x}, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathcal{O}_1(\textsc{msk},\cdot)}(\textsc{mpk})$ for $\mathbf{x} \in \mathcal{X}_\lambda^{q_c}$
$\mathsf{ct}_i^* \leftarrow \mathsf{Encrypt}(\textsc{mpk}, x_i)$ for $i \in [q_c]$
$\alpha \leftarrow \mathcal{A}_2^{\mathcal{O}_2(\textsc{msk},\cdot)}(\textsc{mpk}, \{\mathsf{ct}_i^*\}_{i\in[q_c]}, \mathsf{st})$
Output $(\mathbf{x}, \{f\}, \alpha)$

Experiment $\mathsf{Ideal}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda)$:
$(\textsc{mpk}, \mathsf{st}') \leftarrow \mathcal{S}_1(1^\lambda)$
$(\mathbf{x}, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathcal{O}'_1(\mathsf{st}',\cdot)}(\textsc{mpk})$ where $\mathbf{x} \in \mathcal{X}_\lambda^{q_c}$
- Let $f_1, \dots, f_{q_1}$ be $\mathcal{A}_1$'s oracle queries
- Let $y_{ij} = f_j(x_i)$ for $i \in [q_c]$, $j \in [q_1]$

$(\{\mathsf{ct}_i^*\}_{i\in[q_c]}, \mathsf{st}') \leftarrow \mathcal{S}_3(\mathsf{st}', \{y_{ij}\}_{i\in[q_c], j\in[q_1]})$
$\alpha \leftarrow \mathcal{A}_2^{\mathcal{O}'_2(\mathsf{st}',\cdot)}(\textsc{mpk}, \{\mathsf{ct}_i^*\}_{i\in[q_c]}, \mathsf{st})$
Output $(\mathbf{x}, \{f'\}, \alpha)$

where $\mathcal{O}_1(\textsc{msk},\cdot)$ and $\mathcal{O}'_1(\mathsf{st}',\cdot)$ are pre-challenge key-generation oracles, and $\mathcal{O}_2(\textsc{msk},\cdot)$ and $\mathcal{O}'_2(\mathsf{st}',\cdot)$ are post-challenge ones. The oracles take a function $f \in \mathcal{F}_\lambda$ as input and behave as follows:

- **Real experiment:** Oracles $\mathcal{O}_1(\textsc{msk},\cdot)$ and $\mathcal{O}_2(\textsc{msk},\cdot)$ both implement the key-generation function $\mathsf{KeyGen}(\textsc{msk},\cdot)$. The set $\{f\}$ is the (ordered) set of key queries made to $\mathcal{O}_1(\textsc{msk},\cdot)$ in the pre-challenge phase and to $\mathcal{O}_2(\textsc{msk},\cdot)$ in the post-challenge phase.

- **Ideal experiment:** Oracles $\mathcal{O}'_1(\mathsf{st}',\cdot)$ and $\mathcal{O}'_2(\mathsf{st}',\cdot)$ are the simulator algorithms $\mathcal{S}_2(\mathsf{st}',\cdot)$ and $\mathcal{S}_4(\mathsf{st}',\cdot)$, respectively. On each invocation, the post-challenge simulator $\mathcal{S}_4$ is also given oracle access to the ideal functionality $\mathsf{KeyIdeal}(\mathbf{x},\cdot)$. The functionality $\mathsf{KeyIdeal}$ accepts key queries $f' \in \mathcal{F}_\lambda$ and returns $f'(x_i)$ for every $x_i \in \mathbf{x}$. Both algorithms $\mathcal{S}_2$ and $\mathcal{S}_4$ are stateful. In particular, after each invocation, they update their state st', which is carried over to the next invocation. The (ordered) set $\{f'\}$ denotes the key queries made to $\mathcal{O}'_1(\mathsf{st}',\cdot)$ in the pre-challenge phase, and the queries $\mathcal{S}_4$ makes to $\mathsf{KeyIdeal}$ in the post-challenge phase.

3 Functional Encryption for Randomized Functionalities

In a functional encryption scheme that supports randomized functionalities, the function class $\mathcal{F}_\lambda$ is expanded to include randomized functions from the domain $\mathcal{X}_\lambda$ to the range $\mathcal{Y}_\lambda$. Thus, we now view the functions $f \in \mathcal{F}_\lambda$ as taking as input a domain element $x \in \mathcal{X}_\lambda$ and randomness $r \in \mathcal{R}_\lambda$, where $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ is the randomness space. As in the deterministic setting, the functional encryption scheme still consists of the same four algorithms, but the correctness and security requirements differ substantially.

For instance, in the randomized setting, whenever the decryption algorithm is invoked on a fresh encryption of a message x or a fresh key for a function f, we would expect that the resulting output is indistinguishable from evaluating $f(x)$ with fresh randomness. Moreover, this property should hold regardless of the number of ciphertexts and keys one has. To capture this property, the correctness requirement for an FE scheme supporting randomized functions must consider multiple keys and ciphertexts. In contrast, in the deterministic setting, correctness for a single key-ciphertext pair implies correctness for multiple.

Definition 3.1 (Correctness)**.** A functional encryption scheme $\mathsf{rFE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{KeyGen}, \mathsf{Decrypt})$ for a randomized function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ over a message space $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and a randomness space $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ is *correct* if for every polynomial $n = n(\lambda)$, every $\mathbf{f} \in \mathcal{F}_\lambda^n$ and every $\mathbf{x} \in \mathcal{X}_\lambda^n$, the following two distributions are computationally indistinguishable:

1. **Real**: $\{\mathsf{Decrypt}(\textsc{mpk}, \mathsf{sk}_i, \mathsf{ct}_j)\}_{i,j\in[n]}$, where:

- $(\text{MPK}, \text{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$;
- $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\text{MSK}, f_i)$ for $i \in [n]$;
- $\mathsf{ct}_j \leftarrow \mathsf{Encrypt}(\text{MPK}, x_j)$ for $j \in [n]$.

2. **Ideal**: $\{f_i(x_j; r_{i,j})\}_{i,j\in[n]}$ where $r_{i,j} \xleftarrow{\text{R}} \mathcal{R}_\lambda$.

As discussed in Section 1.2, formalizing and achieving security against malicious encrypters in the randomized setting is considerably harder than in the deterministic case. A decryption oracle that takes a *single* ciphertext along with a function f does not suffice in the randomized setting, since an adversary could still produce a *collection* of ciphertexts such that when the same key is used for decryption, the outputs are correlated. We could strengthen the security definition by allowing the adversary to query with multiple ciphertexts instead of just one, but as noted in Section 1.2, this direct extension is too strong. In order to obtain a realizable definition, we instead restrict the adversary to submit ciphertexts that do not *behave* in the same way. This is formally captured by defining an *admissible* equivalence relation on the space of ciphertexts.

Definition 3.2 (Admissible Relation on Ciphertext Space)**.** Let $\mathsf{rFE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{KeyGen}, \mathsf{Decrypt})$ be an FE scheme for randomized functions with ciphertext space $\mathcal{T} = \{\mathcal{T}_\lambda\}_{\lambda\in\mathbb{N}}$. Let $\sim$ be an equivalence relation on $\mathcal{T}$. We say that $\sim$ is *admissible* if $\sim$ is efficiently checkable and for all $\lambda \in \mathbb{N}$, all (MPK, MSK) output by $\mathsf{Setup}(1^\lambda)$, all secret keys sk output by $\mathsf{KeyGen}(\text{MSK}, \cdot)$, and all ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{T}_\lambda$, if $\mathsf{ct}_1 \sim \mathsf{ct}_2$, then one of the following holds:

- $\mathsf{Decrypt}(\text{MPK}, \mathsf{sk}, \mathsf{ct}_1) = \perp$ OR $\mathsf{Decrypt}(\text{MPK}, \mathsf{sk}, \mathsf{ct}_2) = \perp$.
- $\mathsf{Decrypt}(\text{MPK}, \mathsf{sk}, \mathsf{ct}_1) = \mathsf{Decrypt}(\text{MPK}, \mathsf{sk}, \mathsf{ct}_2)$.

We now define our strengthened requirement for security against malicious encrypters in the randomized setting. Like [GJKS15], we build on the usual simulation-based definition of security for functional encryption (Definition 2.5) by providing the adversary access to a decryption oracle. The definition we present here differs from that by Goyal et al. in two key respects. First, the adversary can submit multiple ciphertexts to the decryption oracle, and second, the adversary is allowed to choose its challenge messages adaptively (that is, after seeing the public parameters and making secret key queries).

Definition 3.3 (SIM-security for rFE)**.** Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a randomized function family over a domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and randomness space $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$. Let $\mathsf{rFE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{KeyGen}, \mathsf{Decrypt})$ be a randomized functional encryption scheme for $\mathcal{F}$ with ciphertext space $\mathcal{T}$. Let $\sim$ be an admissible equivalence relation associated with $\mathcal{T}$. Then, we say that rFE is (q_1, q_c, q_2)-SIM-*secure against malicious encrypters* if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4, \mathcal{S}_5)$ such that for all efficient adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_1$ makes at most q_1 key-generation queries and $\mathcal{A}_2$ makes at most q_2 key-generation queries, the outputs of the following experiments are computationally indistinguishable:[4]

[4]In the specification of the experiments, the indices i always range over $[q_c]$ and the indices j always range over $[q_1]$.

Experiment $\mathsf{Real}_{\mathcal{A}}^{\mathsf{rFE}}(1^\lambda)$:
$(\text{MPK}, \text{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathbf{x}, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathcal{O}_1(\text{MSK},\cdot), \mathcal{O}_3(\text{MSK},\cdot,\cdot)}(\text{MPK})$ where $\mathbf{x} \in \mathcal{X}_\lambda^{q_c}$
$\mathsf{ct}_i^* \leftarrow \mathsf{Encrypt}(\text{MPK}, x_i)$ for $i \in [q_c]$
$\alpha \leftarrow \mathcal{A}_2^{\mathcal{O}_2(\text{MSK},\cdot), \mathcal{O}_3(\text{MSK},\cdot,\cdot)}(\text{MPK}, \{\mathsf{ct}_i^*\}, \mathsf{st})$
Output $(\mathbf{x}, \{f\}, \{g\}, \{y\}, \alpha)$

Experiment $\mathsf{Ideal}_{\mathcal{A}}^{\mathsf{rFE}}(1^\lambda)$:
$(\text{MPK}, \mathsf{st}') \leftarrow \mathcal{S}_1(1^\lambda)$
$(\mathbf{x}, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathcal{O}_1'(\mathsf{st}',\cdot), \mathcal{O}_3'(\mathsf{st}',\cdot,\cdot)}(\text{MPK})$ where $\mathbf{x} \in \mathcal{X}_\lambda^{q_c}$
- Let $f_1, \ldots, f_{q_1}$ be $\mathcal{A}_1$'s oracle queries to $\mathcal{O}_1'(\mathsf{st}', \cdot)$
- Pick $r_{ij} \xleftarrow{\text{R}} \mathcal{R}_\lambda$, let $y_{ij} = f_j(x_i; r_{ij})$ for all $i \in [q_c]$, $j \in [q_1]$

$(\{\mathsf{ct}_i^*\}, \mathsf{st}') \leftarrow \mathcal{S}_3(\mathsf{st}', \{y_{ij}\})$
$\alpha \leftarrow \mathcal{A}_2^{\mathcal{O}_2'(\mathsf{st}',\cdot), \mathcal{O}_3'(\mathsf{st}',\cdot,\cdot)}(\text{MPK}, \{\mathsf{ct}_i^*\}, \mathsf{st})$
Output $(\mathbf{x}, \{f'\}, \{g'\}, \{y'\}, \alpha)$

where the oracles $\mathcal{O}_1(\text{MSK}, \cdot)$, $\mathcal{O}_1'(\mathsf{st}', \cdot)$, $\mathcal{O}_2(\text{MSK}, \cdot)$, and $\mathcal{O}_2'(\mathsf{st}', \cdot)$ are the analogs of the key-generation oracles from Definition 2.5:

- **Real experiment:** Oracles $\mathcal{O}_1(\text{MSK}, \cdot)$ and $\mathcal{O}_2(\text{MSK}, \cdot)$ implement $\mathsf{KeyGen}(\text{MSK}, \cdot)$, and $\{f\}$ is the (ordered) set of key queries made to oracles $\mathcal{O}_1(\text{MSK}, \cdot)$ and $\mathcal{O}_2(\text{MSK}, \cdot)$.
- **Ideal experiment:** Oracles $\mathcal{O}_1'(\mathsf{st}', \cdot)$ and $\mathcal{O}_2'(\mathsf{st}', \cdot)$ are the simulator algorithms $\mathcal{S}_2(\mathsf{st}', \cdot)$ and $\mathcal{S}_4(\mathsf{st}', \cdot)$, respectively. The simulator $\mathcal{S}_4$ is given oracle access to $\mathsf{KeyIdeal}(\mathbf{x}, \cdot)$, which on input a function $f' \in \mathcal{F}_\lambda$, outputs $f'(x_i; r_i)$ for every $x_i \in \mathbf{x}$ and $r_i \xleftarrow{\text{R}} \mathcal{R}_\lambda$. The (ordered) set $\{f'\}$ consists of the key queries made to $\mathcal{O}_1'(\mathsf{st}', \cdot)$, and the queries $\mathcal{S}_4$ makes to $\mathsf{KeyIdeal}$.

Oracles $\mathcal{O}_3(\text{MSK}, \cdot, \cdot)$ and $\mathcal{O}_3'(\mathsf{st}', \cdot, \cdot)$, are the decryption oracles that take inputs of the form (g, C) where $g \in \mathcal{F}_\lambda$ and $C = \{\mathsf{ct}_i\}_{i \in [m]}$ is a collection of $m = \mathsf{poly}(\lambda)$ ciphertexts. For queries made in the post-challenge phase, we additionally require that $\mathsf{ct}_i^* \notin C$ for all $i \in [q_c]$. Without loss of generality, we assume that for all $i, j \in [m]$, if $i \neq j$, then $\mathsf{ct}_i \not\sim \mathsf{ct}_j$. In other words, the set C contains at most one representative from each equivalence class of ciphertexts.

- **Real experiment**: On input (g, C), $\mathcal{O}_3$ computes $\mathsf{sk}_g \leftarrow \mathsf{KeyGen}(\text{MSK}, g)$. For $i \in [m]$, it sets $y_i = \mathsf{Decrypt}(\mathsf{sk}_g, \mathsf{ct}_i)$ and replies with the ordered set $\{y_i\}_{i \in [m]}$. The (ordered) set $\{g\}$ denotes the functions that appear in the decryption queries of $\mathcal{A}_2$ and $\{y\}$ denotes the set of responses of $\mathcal{O}_3$.
- **Ideal experiment**: On input (g', C'), $\mathcal{O}_3'$ does the following:
  1. For each $\mathsf{ct}_i' \in C'$, invoke the simulator algorithm $\mathcal{S}_5(\mathsf{st}', \mathsf{ct}_i')$ to obtain a value $x_i \in \mathcal{X}_\lambda \cup \{\perp\}$. Note that $\mathcal{S}_5$ is also stateful.
  2. For each $i \in [m]$, if $x_i = \perp$, then the oracle sets $y_i' = \perp$. Otherwise, the oracle choose $r_i \xleftarrow{\text{R}} \mathcal{R}_\lambda$ and sets $y_i' = g'(x_i; r_i)$.
  3. Output the ordered set of responses $\{y_i'\}_{i \in [m]}$.

  The (ordered) set $\{g'\}$ denotes the functions that appear in the decryption queries of $\mathcal{A}_2$ and $\{y'\}$ denotes the outputs of $\mathcal{O}_3'$.

Interpreting the admissibility requirement. It might seem that restricting the adversary to submit at most one ciphertext from each equivalence class weakens our security definition. However, we argue that this restriction is essentially without loss of generality. Consider an ideal model where the adversary is allowed to submit equivalent ciphertexts to the decryption oracle. In the extreme case where the adversary submits *identical* ciphertexts, it does not make sense for the decryption oracle to respond independently on each of the ciphertexts. Rather, it should respond consistently. In constructions of randomized FE that provide malicious security, there naturally arise ciphertexts that are not identical as bit-strings, but are identical from the perspective of the decryption function. In these cases, the expected behavior of the ideal functionality should again be to provide consistent, rather than independent, responses.

Consider now an adversary that submits a function f and a set C of ciphertexts to the decryption oracle, where some ciphertexts in C belong to the same equivalence class. To respond, the challenger can first group these ciphertexts by equivalence class. For each equivalence class C' of ciphertexts in C, the challenger invokes the simulator on C'. On input the collection C', the simulator outputs a *single* value x and indicates which ciphertexts in C', if any, are valid. If C' contains at least one valid ciphertext, the challenger samples a value z from the output distribution of $f(x)$. It then replies with the *same* value z on all ciphertexts marked valid by the simulator, and $\perp$ on all ciphertexts marked invalid. This is a natural generalization of how we would expect the decryption oracle to behave had the adversary submitted identical ciphertexts to it. Thus, it suffices to just consider adversaries that always submit at most one representative from each equivalence class of ciphertexts to the decryption oracle.

4 Our Generic Transformation

In this section, we give our generic construction of a functional encryption for randomized functions from a general-purpose FE scheme for deterministic functions. Fix a security parameter $\lambda \in \mathbb{N}$. Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a randomized function class over a domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$, randomness space $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$. Our construction requires the following ingredients:

- A non-interactive zero-knowledge argument system $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ that is simulation-sound extractable (Definition A.3).
- A Φ-RKA secure pseudorandom function PRF (Definition 2.2) with key-space $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$, domain $\mathcal{X}$, and range $\mathcal{Y}$, where Φ is group-induced (Definition 2.3). Let $\diamond$ denote the group operation on $\mathcal{K}$.
- A family of one-way permutations $\mathsf{OWP} = (\mathsf{OWP.Setup}, \mathsf{OWP.Eval})$ over $\mathcal{K}$ with associated hard-core function $\mathsf{hc} : \mathcal{K}_\lambda \to \{0,1\}^\rho$ (Definition A.5). The number of output bits $\rho = \rho(\lambda)$ is specified below.
- For all $f \in \mathcal{F}_\lambda$ and $k \in \mathcal{K}_\lambda$, let $g_k^f : \mathcal{X}_\lambda \times \mathcal{K}_\lambda \to \mathcal{Y}_\lambda$ be the derandomized function

$$g_k^f(x, k') = f(x; \mathsf{PRF}(k \diamond k', x)). \tag{1}$$

  Let $\mathcal{G}_{\mathcal{F},\lambda}$ be the derandomized function class $\left\{g_k^f \mid f \in \mathcal{F}_\lambda, k \in \mathcal{K}_\lambda\right\}$, and let $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Encrypt}, \mathsf{FE.KeyGen}, \mathsf{FE.Decrypt})$ be a functional encryption scheme for the derandomized class $\mathcal{G}_\mathcal{F} = \{\mathcal{G}_{\mathcal{F},\lambda}\}_{\lambda\in\mathbb{N}}$. By construction, the message space for FE is $\mathcal{X}_\lambda \times \mathcal{K}_\lambda$. Let $\rho = \rho(\lambda)$ be a bound on the number of bits of randomness $\mathsf{FE.Encrypt}$ takes.

We now describe our functional encryption scheme $\mathsf{rFE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{KeyGen}, \mathsf{Decrypt})$ for randomized functionalities:

- **The setup algorithm:** On input the security parameter 1^λ, the setup algorithm Setup samples $(\text{MPK}', \text{MSK}') \leftarrow \mathsf{FE.Setup}(1^\lambda)$, $t \leftarrow \mathsf{OWP.Setup}(1^\lambda)$, and $\sigma \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. It sets $h_t(\cdot) = \mathsf{OWP.Eval}(t, \cdot)$, and outputs the master public key $\text{MPK} = (\text{MPK}', t, \sigma)$ and master secret key $\text{MSK} = \text{MSK}'$.

- **The encryption algorithm:** On input the master public key $\text{MPK} = (\text{MPK}', t, \sigma)$, the encryption algorithm $\mathsf{Encrypt}$ samples $k \xleftarrow{\text{R}} \mathcal{K}_\lambda$ and sets $\mathsf{ct}' = \mathsf{FE.Encrypt}(\text{MPK}', (x, h_t(k)); \mathsf{hc}(k))$. Then, it runs $\mathsf{NIZK.Prove}(\sigma, s, (x, k))$ to obtain an argument π on the following statement s:

$$\exists\, x, k : \mathsf{ct}' = \mathsf{FE.Encrypt}(\text{MPK}', (x, h_t(k)); \mathsf{hc}(k)). \tag{2}$$

  Finally, it outputs the ciphertext $\mathsf{ct} = (\mathsf{ct}', \pi)$.

- **The key-generation algorithm:** On input the master secret key $\text{MSK} = \text{MSK}'$, the key-generation algorithm KeyGen samples a key $k \xleftarrow{\text{R}} \mathcal{K}_\lambda$ and outputs the secret key $\mathsf{sk}_f = \mathsf{FE.KeyGen}(\text{MSK}', g_k^f)$, where g_k^f is the derandomized function corresponding to f (Eq. (1)).

- **The decryption algorithm:** On input the master public key $\text{MPK} = (\text{MPK}', t, \sigma)$, a secret key sk, and a ciphertext $\mathsf{ct} = (\mathsf{ct}', \pi)$, the decryption algorithm $\mathsf{Decrypt}$ first runs $\mathsf{NIZK.Verify}(\sigma, s, \pi)$ where s is the statement from Eq. (2). If the argument verifies, the decryption outputs $\mathsf{FE.Decrypt}(\mathsf{sk}, \mathsf{ct}')$; otherwise, it outputs $\perp$.

Theorem 4.1. *If (1)* NIZK *is a simulation-sound extractable non-interactive zero-knowledge argument, (2)* PRF *is a* Φ*-RKA secure pseudorandom function where* Φ *is group-induced, (3)* OWP *is a family of one-way permutations with hard-core function* hc*, and (4)* FE *is a perfectly-correct* (q_1, q_c, q_2)-SIM *secure functional encryption scheme for the derandomized class* $\mathcal{G}_\mathcal{F}$*, then* rFE *is* (q_1, q_c, q_2)-SIM *secure against malicious encrypters for the class* $\mathcal{F}$ *of randomized functions.*

Before proceeding with the proof of Theorem 4.1, we remark that our strengthened definition of security against malicious encrypters (Definition 3.3) is indeed stronger than the original definition by Goyal et al. [GJKS15].

Remark 4.2. A simpler version of our generic transformation (Theorem 4.1) where we only secret share the RKA-secure PRF key used for derandomization and include a NIZK argument can be shown to satisfy the Goyal et al. [GJKS15] definition of security against malicious encrypters, but not our strengthened definition (Definition 3.3). In particular, if the randomness used in the base FE encryption is under the control of the adversary, a malicious encrypter can construct two fresh encryptions (under the base FE scheme) of the same (x, k) pair and submit them to the decryption oracle. In the real world, the outputs are identical (since the ciphertexts encrypt identical messages), but in the ideal world, the oracle replies with two independent outputs. This is an admissible query because if the underlying FE scheme is secure, one cannot *efficiently* decide whether two FE ciphertexts encrypt the same value without knowing any scheme parameters. But because each *individual* output is still properly distributed (by RKA-security of the PRF), security still holds in the Goyal et al. model.

We now proceed to give a proof of Theorem 4.1 in Sections 4.1 and 4.2. Then, in Section 4.3, we show that our transformed scheme is correct.

4.1 Proof of Theorem 4.1: Description of Simulator

To prove Theorem 4.1, and show that rFE is secure in the sense of Definition 3.3, we first define an equivalence relation $\sim$ over the ciphertext space $\mathcal{T} = \{\mathcal{T}_\lambda\}_{\lambda \in \mathbb{N}}$. Take two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{T}_\lambda$, and write $\mathsf{ct}_1 = (\mathsf{ct}_1', \pi_1)$ and $\mathsf{ct}_2 = (\mathsf{ct}_2', \pi_2)$. We say that $\mathsf{ct}_1 \sim \mathsf{ct}_2$ if $\mathsf{ct}_1' = \mathsf{ct}_2'$.

Certainly, $\sim$ is an efficiently-checkable equivalence relation over $\mathcal{T}_\lambda$. For the second admissibility condition, take any (MPK, MSK) output by Setup and any sk output by KeyGen$(\text{MSK}, \cdot)$. Suppose moreover that $\mathsf{Decrypt}(\text{MPK}, \mathsf{sk}, \mathsf{ct}_1) \neq \bot \neq \mathsf{Decrypt}(\text{MPK}, \mathsf{sk}, \mathsf{ct}_2)$. Then, by definition of $\mathsf{Decrypt}(\text{MPK}, \mathsf{sk}, \cdot)$,

$$\begin{aligned}\mathsf{Decrypt}(\text{MPK}, \mathsf{sk}, \mathsf{ct}_1) &= \mathsf{FE.Decrypt}(\text{MPK}', \mathsf{sk}, \mathsf{ct}_1') \\ &= \mathsf{FE.Decrypt}(\text{MPK}', \mathsf{sk}, \mathsf{ct}_2') = \mathsf{Decrypt}(\text{MPK}, \mathsf{sk}, \mathsf{ct}_2),\end{aligned}$$

where MPK' is the master public key for the underlying FE scheme (included in MPK). The second equivalence follows since $\mathsf{ct}_1' = \mathsf{ct}_2'$.

We now describe our ideal-world simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4, \mathcal{S}_5)$. Let $\mathcal{S}^{(\text{FE})} = (\mathcal{S}_1^{(\text{FE})}, \mathcal{S}_2^{(\text{FE})}, \mathcal{S}_3^{(\text{FE})}, \mathcal{S}_4^{(\text{FE})})$ be the simulator for the underlying FE scheme for deterministic functionalities. Let $\mathcal{S}^{(\text{NIZK})} = (\mathcal{S}_1^{(\text{NIZK})}, \mathcal{S}_2^{(\text{NIZK})})$ and $\mathcal{E}^{(\text{NIZK})} = (\mathcal{E}_1^{(\text{NIZK})}, \mathcal{E}_2^{(\text{NIZK})})$ be the simulation and extraction algorithms, respectively, for the NIZK argument system.

Algorithm $\mathcal{S}_1(1^\lambda)$. $\mathcal{S}_1$ simulates the setup procedure. On input a security parameter 1^λ, it operates as follows:

1. Invoke $\mathcal{S}_1^{(\text{FE})}(1^\lambda)$ to obtain a master public key MPK' and some state $\mathsf{st}^{(\text{FE})}$.
2. Invoke $\mathcal{E}_1^{(\text{NIZK})}(1^\lambda)$ to obtain a CRS σ, a simulation trapdoor τ, and an extraction trapdoor ξ.
3. Sample a one-way permutation $t \leftarrow \mathsf{OWP.Setup}(1^\lambda)$ and define $h_t(\cdot) = \mathsf{OWP.Eval}(t, \cdot)$.
4. Set $\text{MPK} \leftarrow (\text{MPK}', t, \sigma)$ and $\mathsf{st} \leftarrow (\mathsf{st}^{(\text{FE})}, \text{MPK}, \tau, \xi)$. Output $(\text{MPK}, \mathsf{st})$.

Algorithm $\mathcal{S}_2(\mathsf{st}_0, f)$. $\mathcal{S}_2$ simulates the pre-challenge key-generation queries. On input a state $\mathsf{st}_0 = (\mathsf{st}_0^{(\text{FE})}, \text{MPK}, \tau, \xi)$ and a function $f \in \mathcal{F}_\lambda$, it operates as follows:

1. Choose a random key $k \xleftarrow{\text{R}} \mathcal{K}_\lambda$ and construct the derandomized function g_k^f as defined in Eq. (1).
2. Invoke $\mathcal{S}_2^{(\text{FE})}(\mathsf{st}_0^{(\text{FE})}, g_k^f)$ to obtain a key sk and an updated state $\mathsf{st}_1^{(\text{FE})}$.
3. Output the key sk and an updated state $\mathsf{st}_1 = (\mathsf{st}_1^{(\text{FE})}, \text{MPK}, \tau, \xi)$.

Algorithm $\mathcal{S}_3(\mathsf{st}_0, \{y_{ij}\}_{i\in[q_c], j\in[q_1]})$. $\mathcal{S}_3$ constructs the challenge ciphertexts. Let $\mathbf{x} = (x_1, x_2, \dots, x_{q_c})$ be the challenge messages the adversary outputs. On input a state $\mathsf{st}_0 = (\mathsf{st}_0^{(\text{FE})}, \text{MPK}, \tau, \xi)$, where $\text{MPK} = (\text{MPK}', t, \sigma)$, and a collection of function evaluations $\{y_{ij}\}_{i\in[q_c], j\in[q_1]}$, $\mathcal{S}_3$ operates as follows:

1. Invoke $\mathcal{S}_3^{(\text{FE})}(\mathsf{st}_0^{(\text{FE})}, \{y_{ij}\}_{i\in[q_c], j\in[q_1]})$ to obtain a set of ciphertexts $\{\mathsf{ct}_i'\}_{i\in[q_c]}$ and an updated state $\mathsf{st}_1^{(\text{FE})}$.
2. For $i \in [q_c]$, let s_i be the statement

$$\exists x, k : \mathsf{ct}_i' = \mathsf{FE.Encrypt}(\text{MPK}', (x, h_t(k)); \mathsf{hc}(k)). \tag{3}$$

   Using the trapdoor τ in st_0, simulate an argument $\pi_i \leftarrow \mathcal{S}_2^{(\text{NIZK})}(\sigma, \tau, s_i)$, and set $\mathsf{ct}_i^* = (\mathsf{ct}_i', \pi_i)$.
3. Output the challenge ciphertexts $\{\mathsf{ct}_i^*\}_{i\in[q_c]}$ and the updated state $\mathsf{st}_1 = (\mathsf{st}_1^{(\text{FE})}, \text{MPK}, \tau, \xi)$.

Algorithm $\mathcal{S}_4(\mathsf{st}_0, f)$**.** $\mathcal{S}_4$ simulates the post-challenge key-generation queries with help from the ideal functionality $\mathsf{KeyIdeal}(\mathbf{x}, \cdot)$. On input a state $\mathsf{st}_0 = (\mathsf{st}_0^{(\text{FE})}, \text{MPK}, \tau, \xi)$ and a function $f \in \mathcal{F}_\lambda$, it operates as follows:

1. Chooses a random key $k \xleftarrow{\text{R}} \mathcal{K}$, and construct the derandomized function g_k^f as defined in Eq. (1).
2. Invoke $\mathcal{S}_4^{(\text{FE})}(\mathsf{st}_0^{(\text{FE})}, g_k^f)$. Here, $\mathcal{S}_4$ also simulates the $\mathsf{FE.KeyIdeal}(\mathbf{x}, \cdot)$ oracle for $\mathcal{S}_4^{(\text{FE})}$. Specifically, when $\mathcal{S}_4^{(\text{FE})}$ makes a query of the form $g_{k'}^{f'}$ to $\mathsf{FE.KeyIdeal}(\mathbf{x}, \cdot)$, $\mathcal{S}_4$ queries its own oracle $\mathsf{KeyIdeal}(\mathbf{x}, \cdot)$ on f' to obtain values z_i for each $i \in [q_c]$.[5] It replies to $\mathcal{S}_4^{(\text{FE})}$ with the value z_i for all $i \in [q_c]$. Let sk and $\mathsf{st}_1^{(\text{FE})}$ be the output of $\mathcal{S}_4^{(\text{FE})}$.
3. Output the key sk and an updated state $\mathsf{st}_1 = (\mathsf{st}_1^{(\text{FE})}, \text{MPK}, \tau, \xi)$.

Algorithm $\mathcal{S}_5(\mathsf{st}, \mathsf{ct})$**.** $\mathcal{S}_5$ handles the decryption queries. On input a state $\mathsf{st} = (\mathsf{st}^{(\text{FE})}, \text{MPK}, \tau, \xi)$ and a ciphertext ct, it proceeds as follows:[6]

1. Parse MPK as (MPK', t, σ) and ct as (ct', π). Let s be the statement

$$\exists x, k : \mathsf{ct} = \mathsf{FE.Encrypt}(\text{MPK}', (x, h_t(k)); \mathsf{hc}(k)).$$

   If $\mathsf{NIZK.Verify}(\sigma, s, \pi) = 0$, then stop and output $\perp$.
2. Otherwise, invoke the extractor $\mathcal{E}_2^{(\text{NIZK})}(\sigma, \xi, s, \pi)$ using the extraction trapdoor ξ to obtain a witness $(x, k) \in \mathcal{X}_\lambda \times \mathcal{K}_\lambda$. Output x and state st.

4.2 Proof of Theorem 4.1: Hybrid Argument

To prove security, we proceed via a series of hybrid experiments between an adversary $\mathcal{A}$ and a challenger. Each experiment consists of the following phases:

1. **Setup phase.** The challenger begins by generating the public parameters of the rFE scheme, and sends those to the adversary $\mathcal{A}$.
2. **Pre-challenge queries.** In this phase of the experiment, $\mathcal{A}$ can issue key-generation queries of the form $f \in \mathcal{F}_\lambda$ and decryption queries of the form $(f, C) \in \mathcal{F}_\lambda \times \mathcal{T}_\lambda^m$ to the challenger. For all decryption queries (f, C), we require that for any $\mathsf{ct}_i, \mathsf{ct}_j \in C$, $\mathsf{ct}_i \not\sim \mathsf{ct}_j$ if $i \neq j$. In other words, each set of ciphertexts C can contain at most one representative from each equivalence class.
3. **Challenge phase.** The adversary $\mathcal{A}$ submits a vector of messages $\mathbf{x} \in \mathcal{X}_\lambda^{q_c}$ to the challenger, who replies with ciphertexts $\{\mathsf{ct}_i^*\}_{i \in [q_c]}$.
4. **Post-challenge queries.** In this phase, $\mathcal{A}$ is again allowed to issue key-generation and decryption queries, with a further restriction that no decryption query can contain any of the challenge ciphertexts (i.e., for any query (f, C), $\mathsf{ct}_i^* \notin C$ for all $i \in [q_c]$).

[5] The underlying FE scheme is for the derandomized class $\mathcal{G}_\mathcal{F}$, so the only permissible functions $\mathcal{S}_4^{(\text{FE})}$ can issue to $\mathsf{FE.KeyIdeal}$ are of the form $g_{k'}^{f'}$ for some k' and f'.

[6] Recall that in the security definition (Definition 3.3), the decryption oracle accepts *multiple* ciphertexts, and invokes the simulator on each one individually. Thus, the simulator algorithm operates on a single ciphertext at a time.

5. **Output.** At the end of the experiment, $\mathcal{A}$ outputs a bit $b \in \{0,1\}$.

We now describe our sequence of hybrid experiments. Note that in defining a new hybrid, we only describe the phases that differ from the previous one. If one or more of the above phases are omitted, the reader should assume that they are exactly the same as in the previous hybrid.

Hybrid Hyb_0. In this experiment, the challenger responds to $\mathcal{A}$ according to the specification of the real experiment $\mathsf{Real}_{\mathcal{A}}^{\mathsf{rFE}}$.

- **Setup phase.** The challenger samples $(\text{MPK}, \text{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$ and sends MPK to $\mathcal{A}$.
- **Pre-challenge queries.** The challenger responds to each query as follows:
  - **Key-generation queries.** On a key-generation query $f \in \mathcal{F}_\lambda$, the challenger responds with $\mathsf{KeyGen}(\text{MSK}, f)$.
  - **Decryption queries.** On a decryption query $(f, C) \in \mathcal{F}_\lambda \times \mathcal{T}_\lambda^m$, the challenger samples $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\text{MSK}, f)$. For each $\mathsf{ct}_i \in C$, the challenger sets $y_i = \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}_i)$, and sends $\{y_i\}_{i \in [m]}$ to the adversary.
- **Challenge phase.** When the challenger receives a vector $\mathbf{x} \in \mathcal{X}_\lambda^{q_c}$, it sets $\mathsf{ct}_i^* = \mathsf{Encrypt}(\text{MPK}, x_i)$ for each $i \in [q_c]$ and replies to $\mathcal{A}$ with $\{\mathsf{ct}_i^*\}_{i \in [q_c]}$.
- **Post-challenge queries.** This is identical to the pre-challenge phase.

Hybrid Hyb_1. This is the same as Hyb_0, except the challenger simulates the CRS in the setup phase and the arguments in the challenge ciphertexts in the challenge phase. Let $\mathcal{S}^{(\text{NIZK})} = (\mathcal{S}_1^{(\text{NIZK})}, \mathcal{S}_2^{(\text{NIZK})})$ be the simulator for NIZK (Definition A.2). Note that we omit the description of the pre- and post-challenge phases in the description below because they are identical to those phases in Hyb_0.

- **Setup phase.** The challenger generates the public parameters as in Hyb_0, except it uses $\mathcal{S}_1^{(\text{NIZK})}$ to generate the CRS. Specifically, it does the following:
  1. Sample $(\text{MPK}', \text{MSK}') \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
  2. Run $\mathcal{S}_1^{(\text{NIZK})}(1^\lambda)$ to obtain a CRS σ and a simulation trapdoor τ.
  3. Sample a one-way permutation $t \leftarrow \mathsf{OWP.Setup}(1^\lambda)$, and define $h_t(\cdot) = \mathsf{OWP.Eval}(t, \cdot)$.
  4. Set $\text{MPK} = (\text{MPK}', t, \sigma)$ and send MPK to $\mathcal{A}$.
- **Challenge phase.** The challenger constructs the challenge ciphertexts as in Hyb_0, except it uses $\mathcal{S}_2^{(\text{NIZK})}$ to simulate the NIZK arguments. Let $\mathbf{x} \in \mathcal{X}_\lambda^{q_c}$ be the adversary's challenge. For $i \in [q_c]$, the challenger samples $k_i^* \xleftarrow{\text{R}} \mathcal{K}_\lambda$ and sets $\mathsf{ct}'_i \leftarrow \mathsf{FE.Encrypt}(\text{MPK}', (x_i, h_t(k_i^*)); \mathsf{hc}(k_i^*))$. It invokes $\mathcal{S}_2^{(\text{NIZK})}(\sigma, \tau, s_i)$ to obtain a simulated argument π_i, where s_i is the statement in Eq. (3). Finally, it sets $\mathsf{ct}_i^* = (\mathsf{ct}'_i, \pi_i)$ and sends $\{\mathsf{ct}_i^*\}_{i \in [q_c]}$ to $\mathcal{A}$.

Hybrid Hyb_2. This is the same as Hyb_1, except the challenger uses uniformly sampled randomness when constructing the challenge ciphertexts.

- **Challenge phase.** Same as in Hyb_1, except that for every $i \in [q_c]$, the challenger sets $\mathsf{ct}'_i = \mathsf{FE.Encrypt}(\text{MPK}', (x_i, h_t(k_i^*)); r_i)$ for a randomly chosen $r_i \xleftarrow{\text{R}} \{0,1\}^\rho$.

Hybrid Hyb_3**.** This is the same as Hyb_2, except the challenger answers the decryption queries by first extracting the message-key pair (m,k) from the NIZK argument and then evaluating the derandomized function on it. Let $\mathcal{E}^{(\text{NIZK})} = (\mathcal{E}_1^{(\text{NIZK})}, \mathcal{E}_2^{(\text{NIZK})})$ be the extraction algorithm for NIZK (Definition A.3).

- **Setup phase.** Same as in Hyb_2 (or Hyb_1), except the challenger runs $(\sigma,\tau,\xi) \leftarrow \mathcal{E}_1^{(\text{NIZK})}(1^\lambda)$ to obtain the CRS σ, the simulation trapdoor τ, and the extraction trapdoor ξ.
- **Pre-challenge queries.** The key-generation queries are handled as in Hyb_2, but the decryption queries are handled as follows.
  - **Decryption queries.** On input (f,C), where $C = \{\mathsf{ct}_i\}_{i\in[m]}$,
    1. Choose a random key $k \xleftarrow{\text{R}} \mathcal{K}_\lambda$.
    2. For $i \in [m]$, parse ct_i as (ct'_i, π_i), and let s_i be the statement in Ea. (3). If $\mathsf{NIZK.Verify}(\sigma, s_i, \pi_i) = 0$, set $y_i = \perp$. Otherwise, invoke the extractor $\mathcal{E}_2^{(\text{NIZK})}(\sigma,\xi,s_i,\pi_i)$ to obtain a witness (x_i,k_i), and set $y_i = f(x_i; \mathsf{PRF}(k \diamond h_t(k_i), x_i))$.
    3. Send the set $\{y_i\}_{i\in[m]}$ to $\mathcal{A}$.
- **Post-challenge queries.** This is identical to the pre-challenge phase.

Hybrid Hyb_4**.** This is the same as Hyb_3, except the challenger uses the simulator $\mathcal{S}^{(\text{FE})} = (\mathcal{S}_1^{(\text{FE})}, \mathcal{S}_2^{(\text{FE})}, \mathcal{S}_3^{(\text{FE})}, \mathcal{S}_4^{(\text{FE})})$ for the underlying FE scheme to respond to queries. Let $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4, \mathcal{S}_5)$ be the simulator described in Section 4.1.

- **Setup phase.** Same as in Hyb_3, except the challenger invokes the base FE simulator $\mathcal{S}_1^{(\text{FE})}$ to construct MPK. The resulting setup algorithm corresponds to the simulation algorithm $\mathcal{S}_1$. Hence, we can alternately say that the challenger runs $\mathcal{S}_1(1^\lambda)$ to obtain $\text{MPK} = (\text{MPK}', t, \sigma)$ and $\mathsf{st} = (\mathsf{st}^{(\text{FE})}, \text{MPK}, \tau, \xi)$, and sends MPK to $\mathcal{A}$.
- **Pre-challenge queries.** The decryption queries are handled as described in Hyb_3, but key-generation queries are handled as follows.
  - **Key-generation queries.** On a key-generation query $f \in \mathcal{F}_\lambda$,
    1. Sample a key $k \xleftarrow{\text{R}} \mathcal{K}_\lambda$. Let g_k^f be the derandomized function corresponding to f.
    2. Run $\mathcal{S}_2^{(\text{FE})}(\mathsf{st}^{(\text{FE})}, g_k^f)$ to obtain a secret key sk and an updated state.
    3. Update st accordingly and send sk to $\mathcal{A}$.

    Note that this is exactly how $\mathcal{S}_2$ behaves when given f and st as inputs.
- **Challenge phase.** The challenger constructs the challenge ciphertexts using the simulation algorithm $\mathcal{S}_3$. Specifically, it does the following on receiving $\mathbf{x} \in \mathcal{X}_\lambda^{q_c}$:
  1. For each $i \in [q_c]$, choose a key $k_i^* \xleftarrow{\text{R}} \mathcal{K}_\lambda$.
  2. Let $f_1, \ldots, f_{q_1} \in \mathcal{F}_\lambda$ be the pre-challenge key-generation queries made by $\mathcal{A}$ and $k_1, \ldots, k_{q_1} \in \mathcal{K}_\lambda$ be the keys chosen when responding to each query. For all $i \in [q_c]$ and $j \in [q_1]$, compute $r_{ij} = \mathsf{PRF}(k_j \diamond h_t(k_i^*), x_i)$ and set $y_{ij} = f_j(x_i; r_{ij})$.
  3. Invoke the simulator algorithm $\mathcal{S}_3(\mathsf{st}, \{y_{ij}\}_{i\in[q_c], j\in[q_1]})$ to obtain a collection of ciphertexts $\{\mathsf{ct}_i^*\}_{i\in[q_c]}$ and an updated state st.

4. Send $\{\mathsf{ct}_i^*\}_{i\in[q_c]}$ to $\mathcal{A}$.

- **Post-challenge queries.** The decryption queries are handled as in the pre-challenge phase, but key-generation queries are handled differently as follows.
  - **Key-generation queries.** The first step stays the same: a key k is picked at random and g_k^f is defined. The challenger then invokes $\mathcal{S}_4^{(\mathrm{FE})}$ with inputs $\mathsf{st}^{(\mathrm{FE})}$ and g_k^f, instead of $\mathcal{S}_2^{(\mathrm{FE})}$. In invoking $\mathcal{S}_4^{(\mathrm{FE})}$, it simulates the $\mathsf{FE.KeyIdeal}(\mathbf{x},\cdot)$ oracle as follows: on input a function of the form $g_{k'}^{f'}$, it computes $y_i = g_{k'}^{f'}(x_i, h_t(k_i^*)) = f'(x_i; \mathsf{PRF}(k' \diamond h_t(k_i^*), x_i))$ and replies with the set $\{y_i\}_{i\in[q_c]}$. The function key returned by $\mathcal{S}_4^{(\mathrm{FE})}$ is given to $\mathcal{A}$, and st is updated appropriately. This is the behavior of $\mathcal{S}_4$.

Hybrid Hyb_5. This is the same as Hyb_4, except the outputs of PRF are replaced by truly random strings. This matches the specification of the ideal experiment $\mathsf{Ideal}_{\mathcal{A}}^{\mathsf{rFE}}$. We highlight below the differences from the previous hybrid.

- **Pre-challenge queries.** While the key queries are handled as before, the decryption queries are handled as follows.
  - **Decryption queries.** Same as in Hyb_4, except the function f is evaluated using uniformly sampled randomness. In other words, on input f and $C = \{\mathsf{ct}_i\}_{i\in[m]}$, the challenger does the following:
    1. For every $\mathsf{ct}_i \in C$, invoke the simulator algorithm $\mathcal{S}_5(\mathsf{st}, \mathsf{ct}_i)$ to obtain a value $x_i \in \mathcal{X}_\lambda \cup \{\perp\}$ and an updated state st.
    2. If $x_i = \perp$, set y_i to $\perp$, else set it to $f(x_i; r_i)$, where $r_i \xleftarrow{\mathrm{R}} \mathcal{R}_\lambda$.
    3. Send the set of values $\{y_i\}_{i\in[m]}$ to $\mathcal{A}$.

- **Challenge phase.** The challenge ciphertexts are constructed as in the ideal experiment. Specifically, instead of using PRF to generate the randomness for evaluating y_{ij} in the first and second steps of the challenge phase, the challenger simply computes $f_j(x_i; r_{ij})$ for $r_{ij} \xleftarrow{\mathrm{R}} \mathcal{R}_\lambda$. The remaining two steps (third and fourth) stay the same.

- **Post-challenge queries.** The decryption queries are handled as in the pre-challenge phase, but key queries are handled as follows:
  - **Key-generation queries.** Same as Hyb_4, except the oracle $\mathsf{FE.KeyIdeal}(\mathbf{x},\cdot)$ is implemented using uniformly sampled randomness as in the ideal experiment. Specifically, if $\mathcal{S}_4^{(\mathrm{FE})}$ makes a query to $\mathsf{FE.KeyIdeal}(\mathbf{x},\cdot)$ with a derandomized function $g_{k'}^{f'}$, the challenger chooses an $r_i \xleftarrow{\mathrm{R}} \mathcal{R}_\lambda$ for every $i \in [q_c]$, and replies with $\{f'(x_i; r_i)\}_{i\in[q_c]}$.

We now state lemmas that each consecutive pair of hybrid experiments is computationally indistinguishable, but defer their proofs to Appendix B.

Lemma 4.3. *If* NIZK *is computational zero-knowledge (Definition A.2), then* Hyb_0 *and* Hyb_1 *are computationally indistinguishable.*

Lemma 4.4. *If* OWP *is a family of one-way permutations and* hc *is a hard-core function, then* Hyb_1 *and* Hyb_2 *are computationally indistinguishable.*

Lemma 4.5. *If* NIZK *is simulation-sound extractable (Definition A.3), and* FE *is perfectly correct, then* Hyb_2 *and* Hyb_3 *are computationally indistinguishable.*

Lemma 4.6. *If* FE *is a* (q_1, q_c, q_2)*-*SIM*-secure functional encryption scheme for* $\mathcal{G}_{\mathcal{F}}$ *(Definition 2.5), then* Hyb_3 *and* Hyb_4 *are computationally indistinguishable.*

Lemma 4.7. *If* PRF *is* $\Phi_\diamond$*-RKA secure and* FE *is a* (q_1, q_c, q_2)*-*SIM*-secure functional encryption scheme for* $\mathcal{G}_{\mathcal{F}}$*,*[7] *then* Hyb_4 *and* Hyb_5 *are computationally indistinguishable.*

Lemmas 4.3 through 4.7 suffice to show that the adversary's view in the real experiment $\mathsf{Real}_{\mathcal{A}}^{\mathsf{rFE}}$ is computationally indistinguishable from its view in the ideal experiment $\mathsf{Ideal}_{\mathcal{A}}^{\mathsf{rFE}}$ (Definition 3.3). In particular, this means that the tuple $(\mathbf{x}, \{g\}, \{y\}, \alpha)$ in the real experiment is computationally indistinguishable from the tuple $(\mathbf{x}, \{g'\}, \{y'\}, \alpha)$ in the ideal experiment.

To complete the security proof, we show that the remaining components $\{f\}$ and $\{f'\}$ in the outputs of the real and ideal experiments, respectively, are computationally indistinguishable given the other components of the joint distribution. Assuming that $\mathcal{S}^{(\mathrm{FE})}$ is a valid simulator for the underlying FE scheme, this follows directly from the specification of $\mathcal{S}$. By definition, the set $\{f'\}$ consists of the functions the adversary submits to the key-generation oracle in the pre-challenge phase and the queries $\mathcal{S}_4$ makes to the $\mathsf{KeyIdeal}$ oracle. Since the adversary's view in the two experiments are computationally indistinguishable, the pre-challenge function queries appearing in $\{f\}$ and $\{f'\}$ are computationally indistinguishable. Suppose then that $\mathcal{S}_4$ is invoked on a function f. Then, $\mathcal{S}_4$ constructs the derandomized functionality g_k^f for some $k \in \mathcal{K}$ and invokes the underlying FE simulator $\mathcal{S}_4^{(\mathrm{FE})}$ on g_k^f. Assuming that $\mathcal{S}^{(\mathrm{FE})}$ is a simulator for the underlying FE scheme, with overwhelming probability, it will query its oracle $\mathsf{FE.KeyIdeal}$ on the function g_k^f. In response, $\mathcal{S}_4$ queries $\mathsf{KeyIdeal}$ on f. We conclude that the outputs of the real and ideal experiments are computationally indistinguishable, which proves security.

4.3 Proof of Theorem 4.1: Correctness

The correctness proof for rFE follows from completeness of the NIZK argument system, correctness of the underlying FE scheme, and RKA-security of the PRF. We give the full proof in Appendix C.

5 Instantiating and Applying the Transformation

In this section, we describe one way to instantiate the primitives (the NIZK argument system, the RKA-secure PRF, and the one-way permutation) needed to apply the generic transformation from Section 4, Theorem 4.1. Then, in Section 5.2, we show how to obtain new general-purpose functional encryption schemes for randomized functionalities with security against malicious encrypters from a wide range of assumptions by applying our transformation to existing functional encryption schemes.

[7] The proof of this lemma relies on a concrete property of the simulator for a secure FE scheme, which is why we need SIM security for the underlying FE scheme. Alternatively, we can impose an admissibility requirement on the queries the FE simulator is allowed to make to the $\mathsf{FE.KeyIdeal}$ oracle, similar to what is done in the security definitions in [GVW12].

5.1 Instantiating Primitives

All of the primitives required by our generic transformation can be built from standard number-theoretic assumptions, namely the decisional Diffie-Hellman (DDH) assumption [Bon98], the hardness of discrete log in the multiplicative group $\mathbb{Z}_p^*$ (for prime p), and the RSA assumption [RSA78, Bon99]. The first two assumptions can be combined by assuming the DDH assumption holds in a prime-order subgroup of $\mathbb{Z}_p^*$, such as the subgroup of quadratic residues of $\mathbb{Z}_p^*$, where p is a safe prime ($p = 2q + 1$, where q is also prime).

Simulation-sound extractable NIZK arguments. The first ingredient we require is a simulation-sound extractable NIZK argument. De Santis et al. [DDO$^+$01, Theorem 2] give a construction for this from trapdoor one-way permutations and dense cryptosystems.[8] Both of these primitives can be instantiated using the RSA assumption.

RKA-PRFs. The next ingredient we require is a Φ-RKA-secure PRF where Φ is group-induced. One candidate construction from the DDH assumption is the Bellare-Cash PRF [BC10, §4].

Theorem 5.1 (Bellare-Cash PRF [BC10, Theorem 4.2], adapted)**.** *Let $\mathbb{G}$ be a group of prime order p where the DDH assumption holds. Then, there exists a Φ-RKA secure PRF $F_{\mathsf{bc}} : (\mathbb{Z}_p^*)^{n+1} \times \{0,1\}^n \to \mathbb{G}$, where Φ is group-induced and $n = \mathsf{poly}(\lambda)$. The group operation on the key-space $(\mathbb{Z}_p^*)^{n+1}$ is simply element-wise multiplication modulo p.*

One-way permutations. If we instantiate the RKA-secure PRF with the Bellare-Cash PRF, the next ingredient we require is a one-way permutation on the key-space $(\mathbb{Z}_p^*)^{n+1}$. This can be easily constructed from any one-way permutation over $\mathbb{Z}_p^*$. A well-known one-way permutation on $\mathbb{Z}_p^*$ is based on the conjectured intractability of the discrete log problem (DLP). More precisely, the mapping $x \mapsto g^x \pmod p$ where g is a random generator of $\mathbb{Z}_p^*$ is a one-way permutation assuming hardness of the DLP in $\mathbb{Z}_p^*$. Next, we review the Blum-Micali hard-core predicate [BM82] for this family of one-way permutations.

Theorem 5.2 (Blum-Micali Construction [BM82, §3.3])**.** *Fix a prime p and let g be a generator of $\mathbb{Z}_p^*$. Suppose the DLP is hard in $\mathbb{Z}_p^*$. Then, the following function $\mathsf{hc} : \mathbb{Z}_p^* \to \{0,1\}$ is hard-core for the mapping $x \mapsto g^x \pmod p$:*

$$\mathsf{hc}(x) = \begin{cases} 1 & \text{if there exists } 0 \le y < p/2 \text{ such that } g^y = x \\ 0 & \text{otherwise.} \end{cases}$$

In our construction, we require a hard-core function that outputs $\rho = \rho(\lambda)$ number of bits. This is possible by iterating the Blum-Micali construction.[9] In the following, we will write $f^{(i)}(x)$ to denote successively applying the function f on the input x for i iterations (i.e., $f^{(2)}(x) = f(f(x))$.) We now state a corollary to Theorem 5.2.

Corollary 5.3 (Iterated Blum-Micali Construction)**.** *Fix a prime p. Let g be a generator of $\mathbb{Z}_p^*$, $f : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ be the permutation $x \mapsto g^x \pmod p$, and hc_f be the hard-core predicate of f from Theorem 5.2. For*

[8]Dense cryptosystems were introduced by De Santis and Persiano [DP92] to construct proofs of knowledge. In the same work, they showed that dense cryptosystems could be constructed from assumptions such as the RSA assumption.

[9]Note that we can also use more efficient hard-core functions such as [PS98] which outputs multiple hard-core bits on each input. The Blum-Micali construction is just one example that suffices for our transformation.

$\rho = \rho(\lambda)$, *define the permutation* $g : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ *to be the mapping* $x \mapsto f^{(\rho)}(x)$. *Then, if* hc_f *is a hard-core function for* f*, the function* $\mathsf{hc}_g : \mathbb{Z}_p^* \to \{0,1\}^\rho$ *defined as follows is hard-core for* g*:*

$$\mathsf{hc}_g(x) = \mathsf{hc}_f(x) \| \mathsf{hc}_f(f(x)) \| \mathsf{hc}_f(f^{(2)}(x)) \cdots \| \mathsf{hc}_f(f^{(\rho-1)}(x)).$$

Proof. Follows from Theorem 5.2 by a standard hybrid argument. □

Given a one-way permutation g on $\mathbb{Z}_p^*$ and an associated hard-core function hc_g, it is easy to construct a one-way permutation h on $(\mathbb{Z}_p^*)^{n+1}$ and an associated hard-core function hc_h that outputs the same number of bits. We define h to be the function $h(x_1, \ldots, x_{n+1}) = (g(x_1), x_2, \ldots, x_n)$ and hc_h to be $\mathsf{hc}_h(x_1, \ldots, x_{n+1}) = \mathsf{hc}_g(x_1)$.

Thus, we can instantiate the group-induced RKA-PRF and one-way permutation needed by our generic transformation (Theorem 4.1) assuming only that DDH holds in a group of prime order p and the hardness of DLP in $\mathbb{Z}_p^*$. In summary, we obtain the following corollary to Theorem 4.1 from Section 4.

Corollary 5.4. *Assuming standard number-theoretic assumptions (that is, the DDH assumption in a prime-order subgroup of* $\mathbb{Z}_p^*$ *and the RSA assumption), and that* FE *is a perfectly-correct* (q_1, q_c, q_2)-SIM *secure functional encryption scheme for the derandomized function class* $\mathcal{G}_\mathcal{F}$*, then* rFE *is* (q_1, q_c, q_2)-SIM *secure against malicious encrypters for the class* $\mathcal{F}$ *of randomized functions.*

5.2 Applying the Transformation

In this section, we give three examples of how our generic transformation from Section 4 could be applied to existing functional encryption schemes to obtain schemes that support randomized functionalities. Our results show that functional encryption for randomized functionalities secure against malicious encrypters can be constructed from a wide range of assumptions such as public-key encryption, concrete assumptions over composite-order multilinear maps, or indistinguishability obfuscation, in conjunction with standard number-theoretic assumptions (Corollary 5.4). The examples we present here do not constitute an exhaustive list of the functional encryption schemes to which we could apply the transformation. For instance, the construction of single-key-secure, succinct FE from LWE by Goldwasser et al. [GKP+13] and the recent adaptively-secure construction from $i\mathcal{O}$ by Waters [Wat15] are also suitable candidates.

We note that the FE schemes for deterministic functions we consider below are secure (or can be made secure) under a slightly stronger notion of simulation security compared to Definition 2.5. Under the stronger notion (considered in [GVW12, DIJ+13]), the simulator is not allowed to program the public-parameters (they are generated by the Setup algorithm) or the pre-challenge key queries (they are generated using the KeyGen algorithm). Hence, when our transformation is applied to these schemes, there is a small loss in security. We believe that this loss is inherent because the new schemes are secure under malleability attacks while the original schemes are not. In particular, the construction of Goyal et al. [GJKS15] also suffers from this limitation.

The GVW scheme. In [GVW12], Gorbunov et al. give a construction of a general-purpose public-key FE scheme for a bounded number of secret key queries. More formally, they give both a $(q_1, 1, \mathsf{poly})$- and a $(q_1, \mathsf{poly}, 0)$-SIM[10] secure FE scheme for any class of deterministic functions computable by polynomial-size circuits based on the existence of semantically-secure public-key encryption and pseudorandom

[10]We write poly to denotes that the quantity does not have to be a-priori bounded, and can be any polynomial in λ.

generators (PRG) computable by low-degree circuits. These assumptions are implied by many concrete intractability assumptions such as factoring.

The GVW scheme can be made perfectly correct if we have the same guarantee from the two primitives it is based on: a semantically-secure public-key encryption scheme and a decomposable randomized encoding scheme [IK00]. There are many ways to get perfect correctness for the former, like ElGamal [ElG85] or RSA [RSA78]. For the latter, we can use Applebaum et al.'s construction [AIK06, Theorem 4.14]. We can now apply our generic transformation (Corollary 5.4) to the GVW scheme to obtain the following corollary:

Corollary 5.5. *Under standard number-theoretic assumptions, for any polynomial $q_1 = q_1(\lambda)$, there exists a $(q_1, 1, \mathsf{poly})$-SIM and a $(q_1, \mathsf{poly}, 0)$-SIM secure FE scheme for any class of randomized functions computable by polynomial-size circuits with security against malicious encrypters.*

The GGHZ scheme. For our second example, we show how to apply our generic transformation to the recent Garg et al. functional encryption scheme [GGHZ16] based on concrete assumptions over asymmetric multilinear maps. There are two challenges that arise when trying to directly apply our transformation to the GGHZ scheme. First, like many FE schemes, the GGHZ scheme only provides statistical correctness, while our transformation crucially relies on perfect correctness. However, it is easy to see that we can relax our requirement to only require perfect correctness to hold with overwhelming probability over the setup algorithm of the underlying FE scheme. This is the notion of "almost-all-keys perfect correctness" introduced by Dwork et al. [DNR04]. In the same work, Dwork et al. introduce a randomness sparsification technique to transform any encryption scheme with a sufficiently small decryption error probability into one that is perfectly correct with overwhelming probability over the choice of random coins in the setup algorithm. More recently, Bitanski and Vaikuntanathan [BV16, Section 4] also describe a general method for correcting errors in functional encryption schemes, and noted that the randomness sparsification technique of Dwork et al. could be applied to FE schemes to achieve almost-all-keys perfect correctness. Applying the Dwork et al. transformation, the GGHZ scheme gives an adaptively secure FE scheme that is almost-all-keys perfectly correct for general circuits from multilinear maps. Note that the Dwork et al. technique does not require any additional assumptions beyond the existence of one-way functions.

The second obstacle is that the GGHZ scheme was shown to be secure under an indistinguishability-based notion of security while our transformation applies to an FE scheme secure under a simulation-based notion of security. This is easily addressed by using the indistinguishability-to-simulation transformation by De Caro et al. [DIJ$^+$13]. Applying this transformation requires a symmetric encryption scheme with pseudorandom ciphertexts, which is implied by our number-theoretic assumptions. In addition, as long as the underlying symmetric encryption scheme is perfectly correct, the transformation preserves the correctness properties of the base FE scheme. Thus, under the GGHZ complexity assumptions on composite-order multilinear maps [GGHZ16, Section 2.3], there is a $(q_1, q_c, \mathsf{poly})$-$\mathsf{SIM}$ secure FE scheme that is almost-all-keys perfectly correct, where $q_1 = q_1(\lambda)$ and $q_c = q_c(\lambda)$. Applying our generic transformation to the transformed GGHZ scheme, we obtain the following corollary:

Corollary 5.6. *Under standard number-theoretic assumptions, and the GGHZ complexity assumptions on composite-order multilinear maps [GGHZ16, Section 2.3], for any polynomials $q_1 = q_1(\lambda)$ and $q_c = q_c(\lambda)$, there exists a $(q_1, q_c, \mathsf{poly})$-$\mathsf{SIM}$ secure functional encryption for all polynomial-sized randomized functionalities with security against malicious encrypters.*

The GGHRSW scheme. For our final example, we show that starting with the Garg et al. [GGH$^+$13] functional encryption scheme based on indistinguishability obfuscation, we can also obtain a functional encryption for randomized functionalities with the same level of security as above. As usual, we first verify that the the GGHRSW scheme satisfies perfect correctness (alternatively, we could apply the randomness sparsification technique from [DNR04] to obtain a scheme that is almost-all-keys perfectly correct). Correctness of the GGHRSW scheme follows immediately from the correctness of the indistinguishability obfuscator and the underlying public key encryption scheme used in the construction. Thus, instantiating with a perfectly correct public key encryption scheme yields a selectively-secure, general-purpose, public-key FE scheme with perfect correctness.

Another challenge in applying our transformation is that the GGHRSW scheme was shown only to be selectively secure under an indistinguishability-based definition of security. Thus, we cannot directly invoke the De Caro et al. indistinguishability-to-simulation transformation [DIJ$^+$13]. This can be addressed, however, by first applying the selective-to-adaptive transformation by Ananth et al. [ABSV15]. The additional primitives required for this transformation are all implied by any selectively-secure public-key FE scheme, and moreover, each of the primitives can be instantiated with one that provides perfect correctness. In doing so, the transformation preserves the correctness of the underlying scheme.

To conclude, if we apply the selective-to-adaptive and indistinguishability-to-simulation transformations by Ananth et al. and De Caro et al., respectively, to the GGHRSW scheme, we obtain a general-purpose, $(q_1, q_c, \mathsf{poly})$-$\mathsf{SIM}$ secure functional encryption scheme from indistinguishability obfuscation (and one-way functions), where $q_1 = q_1(\lambda)$ and $q_c = q_c(\lambda)$. Applying our generic transformation to the resulting scheme, we arrive at the following corollary:

Corollary 5.7. *Under standard number-theoretic assumptions, and the existence of an indistinguishability obfuscator, for any polynomials $q_1 = q_1(\lambda)$ and $q_c = q_c(\lambda)$, there exists a $(q_1, q_c, \mathsf{poly})$-$\mathsf{SIM}$ secure functional encryption for all polynomial-sized randomized functionalities with security against malicious encrypters.*

Comparison with the GJKS scheme. We note that $(q_1, q_c, \mathsf{poly})$-$\mathsf{SIM}$ security matches the known lower bounds for simulation-based security in the standard model [BSW11, AGVW13]. We remark also that the FE schemes from Corollaries 5.6 and 5.7 provide stronger security than the original FE scheme for randomized functionalities by Goyal et al. [GJKS15]. Their construction was shown to be selectively rather than adaptively secure. Specifically, in their security model, the adversary must commit to its challenge messages before seeing the master public key. On the contrary, when we apply our generic transformation to both the GGHZ scheme from composite-order multilinear maps as well as the GGHSRW scheme from indistinguishability obfuscation, we obtain an adaptive-secure FE scheme where the adversary can not only see the master public key, but also make secret key queries prior to issuing the challenge query.

6 Conclusions

In this work, we developed a generic transformation that converts any general-purpose public-key functional encryption scheme for deterministic functionalities into a corresponding functional encryption scheme that supports the richer class of randomized functionalities. Applying our transformation to existing FE schemes, we obtain the first adaptively-secure FE scheme for randomized functionalities from public-key encryption (and standard number-theoretic assumptions) in the bounded collusion setting, as well as the first adaptively-secure FE scheme for randomized functionalities from either concrete assumptions on multilinear maps or indistinguishability obfuscation. We conclude with several interesting open questions for further study:

- Can we construct an FE scheme for a more restrictive class of randomized functionalities (e.g., sampling from a database) without needing to go through our generic transformation? In other words, for simpler classes of randomized functionalities, can we construct a scheme that does not require a general-purpose FE scheme for deterministic functionalities?
- Is it possible to generically convert a public-key FE scheme for deterministic functionalities into one that supports randomized functionalities *without* making any additional assumptions? Komargodski, Segev, and Yogev [KSY15] show that this is possible in the secret-key setting.

Acknowledgments

We thank Venkata Koppula for many helpful conversations and discussions related to this work.

References

[AAB$^+$15] Shashank Agrawal, Shweta Agrawal, Saikrishna Badrinarayanan, Abishek Kumarasubramanian, Manoj Prabhakaran, and Amit Sahai. On the practical security of inner product functional encryption. In *PKC 2015*, pages 777–798, 2015.

[AAP15] Shashank Agrawal, Shweta Agrawal, and Manoj Prabhakaran. Cryptographic agents: Towards a unified theory of computing on encrypted data. In *EUROCRYPT 2015*, pages 501–531, 2015.

[AB15] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In *TCC*, pages 528–556, 2015.

[ABF$^+$13] Joël Alwen, Manuel Barbosa, Pooya Farshim, Rosario Gennaro, S. Dov Gordon, Stefano Tessaro, and David A. Wilson. On the relationship between functional encryption, obfuscation, and fully homomorphic encryption. In *IMA International Conference on Cryptography and Coding*, pages 65–84, 2013.

[ABG$^+$13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. `http://eprint.iacr.org/2013/689`.

[ABP15] Michel Abdalla, Fabrice Benhamouda, and Alain Passelègue. An algebraic framework for pseudorandom functions and applications to related-key security. In *CRYPTO*, pages 388–409, 2015.

[ABPP14] Michel Abdalla, Fabrice Benhamouda, Alain Passelègue, and Kenneth G. Paterson. Related-key security for pseudorandom functions beyond the linear barrier. In *CRYPTO*, pages 77–94, 2014.

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In *CRYPTO*, pages 657–677, 2015.

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *CRYPTO*, pages 500–518, 2013.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[AS15] Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. Cryptology ePrint Archive, Report 2015/776, 2015. `http://eprint.iacr.org/2015/776`.

[BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO*, pages 535–552, 2007.

[BC10] Mihir Bellare and David Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In *CRYPTO*, pages 666–684, 2010.

[BCM11] Mihir Bellare, David Cash, and Rachel Miller. Cryptography secure against related-key attacks and tampering. In *ASIACRYPT*, pages 486–503, 2011.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, pages 213–229, 2001.

[BF13] Manuel Barbosa and Pooya Farshim. On the semantic security of functional encryption schemes. In *PKC 2013*, pages 143–161, 2013.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *ACM STOC*, pages 103–112, 1988.

[BFOR08] Mihir Bellare, Marc Fischlin, Adam O'Neill, and Thomas Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In *CRYPTO*, pages 360–378, 2008.

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO*, pages 390–420, 1993.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGK$^+$14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, pages 221–238, 2014.

[Bih94] Eli Biham. New types of cryptoanalytic attacks using related keys (extended abstract). In *EUROCRYPT*, pages 398–409, 1994.

[BK03] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *EUROCRYPT*, pages 491–506, 2003.

[BLR$^+$15] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *EUROCRYPT 2015*, pages 563–594, 2015.

[BM82] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *FOCS*, pages 112–117, 1982.

[BO13] Mihir Bellare and Adam O'Neill. Semantically-secure functional encryption: Possibility results, impossibility results and the quest for a general definition. In *CANS*, pages 218–234, 2013.

[Bon98] Dan Boneh. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423, 1998. Invited paper.

[Bon99] Dan Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the American Mathematical Society*, 46(2):203–213, 1999.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[BRS13] Dan Boneh, Ananth Raghunathan, and Gil Segev. Function-private subspace-membership encryption and its applications. In *ASIACRYPT 2013*, pages 255–275, 2013.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *IACR Cryptology ePrint Archive*, 2002:80, 2002.

[BS11] Zvika Brakerski and Gil Segev. Better security for deterministic public-key encryption: The auxiliary-input setting. In *CRYPTO*, pages 543–560, 2011.

[BS15a] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. In *TCC*, pages 306–324, 2015.

[BS15b] Zvika Brakerski and Gil Segev. Hierarchical functional encryption. *IACR Cryptology ePrint Archive*, 2015:1011, 2015.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[BV16] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation: From approximate to exact. In *TCC 2016-A*, pages 67–95, 2016.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[BWZ14] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. Cryptology ePrint Archive, Report 2014/930, 2014. `http://eprint.iacr.org/2014/930`.

[CFL$^+$16] Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new CLT multilinear map over the integers. In *EUROCRYPT*, 2016.

[CGH$^+$15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *CRYPTO*, pages 247–266, 2015.

[CGJS15] Nishanth Chandran, Vipul Goyal, Aayush Jain, and Amit Sahai. Functional encryption: Decentralised and delegatable. *IACR Cryptology ePrint Archive*, 2015:1017, 2015.

[CHL$^+$15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT 2015*, pages 3–12, 2015.

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without a low level encoding of zero. *IACR Cryptology ePrint Archive*, 2016:139, 2016.

[CLLT15] Jean-Sebastien Coron, Moon Sung Lee, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. Cryptology ePrint Archive, Report 2015/1037, 2015. `http://eprint.iacr.org/2015/1037`.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, pages 476–493, 2013.

[CLT15] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *CRYPTO*, pages 267–286, 2015.

[CM16] Michael Clear and Ciaran McGoldrick. Attribute-based fully homomorphic encryption with a bounded number of inputs. *IACR Cryptology ePrint Archive*, 2016:099, 2016.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In Bahram Honary, editor, *Cryptography and Coding, 8th IMA International Conference*, volume 2260, pages 360–363, Cirencester, UK, December 17–19, 2001.

[DDO$^+$01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, pages 566–598, 2001.

[DIJ$^+$13] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In *CRYPTO*, pages 519–535, 2013.

[DNR04] Cynthia Dwork, Moni Naor, and Omer Reingold. Immunizing encryption schemes from decryption errors. In *EUROCRYPT*, pages 342–360, 2004.

[DP92] Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *FOCS*, pages 427–436, 1992.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *FOCS*, pages 308–317, 1990.

[FOR12] Benjamin Fuller, Adam O'Neill, and Leonid Reyzin. A unified approach to deterministic encryption: New constructions and a connection to computational entropy. In *TCC*, pages 582–599, 2012.

[GGG$^+$14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *EUROCRYPT*, pages 578–602, 2014.

[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 *FOCS*, pages 40–49, 2013.

[GGHZ16] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. In *TCC 2016-A*, pages 480–511, 2016.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *CRYPTO*, pages 276–288, 1984.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *ACM STOC*, pages 467–476, 2013.

[GJKS15] Vipul Goyal, Abhishek Jain, Venkata Koppula, and Amit Sahai. Functional encryption for randomized functionalities. In *TCC*, pages 325–351, 2015.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *ACM STOC*, pages 555–564, 2013.

[GLSW15] Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In *FOCS*, pages 151–170, 2015.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *CRYPTO*, pages 426–443, 2014.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In *EUROCRYPT*, pages 339–358, 2006.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006. Available as Cryptology ePrint Archive Report 2006/309.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, pages 444–459, 2006.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, 2012.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *ACM STOC*, pages 545–554, 2013.

[HJ16] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. In *EUROCRYPT*, 2016.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

[Knu93] Lars R. Knudsen. Cryptanalysis of LOKI91. In *AUSCRYPT*, pages 196–208, 1993.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[KSY15] Ilan Komargodski, Gil Segev, and Eylon Yogev. Functional encryption for randomized functionalities in the private-key setting from minimal assumptions. In *TCC*, pages 352–377, 2015.

[LMR14] Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Improved constructions of PRFs secure against related-key attacks. In *ACNS*, pages 44–61, 2014.

[LOS$^+$10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[Luc04] Stefan Lucks. Ciphers secure against related-key attacks. In *FSE*, pages 359–370, 2004.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. *IACR Cryptology ePrint Archive*, 2016:147, 2016.

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

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In *ASIACRYPT 2015*, pages 121–145, 2015.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.

[PS98] Sarvar Patel and Ganapathy S. Sundaram. An efficient discrete log pseudo random generator. In *CRYPTO*, pages 304–317, 1998.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO*, pages 500–517, 2014.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO*, pages 433–444, 1992.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS*, pages 463–472, 2010.

[SSW09] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *TCC*, pages 457–473, 2009.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[Wat15] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In *CRYPTO*, pages 678–697, 2015.

[Zim15] Joe Zimmerman. How to obfuscate programs directly. In *EUROCRYPT 2015*, pages 439–467, 2015.

A Additional Preliminaries

In this section, we review the standard definitions of two additional primitives we use in our construction: non-interactive zero-knowledge (NIZK) arguments of knowledge [BFM88, FLS90, Gro06, GOS06] and one-way permutations.

A.1 Non-Interactive Zero-Knowledge Arguments of Knowledge

Let R be an efficiently computable binary relation. For pairs $(s, w) \in R$, we refer to s as the statement and w as the witness. Let L be the language of statements in R.

Definition A.1 (Non-Interactive Arguments [BFM88, FLS90])**.** A non-interactive argument system for a relation R is a tuple of three efficient algorithms $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ defined as follows:

- $\mathsf{Setup}(1^\lambda)$ takes as input the security parameter λ and outputs a common reference string (CRS) σ of length $\Omega(\lambda)$.
- $\mathsf{Prove}(\sigma, s, w)$ takes as input a CRS σ, a statement s, and a witness w, and outputs an argument π.
- $\mathsf{Verify}(\sigma, s, \pi)$ takes as input a CRS σ, a statement s, and an argument π, and outputs a bit $b \in \{0,1\}$.

We say that $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is a non-interactive argument system for a relation R if it satisfies the following two properties:

- **Perfect Completeness**: An argument system is perfectly complete if for all adversaries $\mathcal{A}$,

$$\Pr\big[\sigma \leftarrow \mathsf{Setup}(1^\lambda); (s, w) \leftarrow \mathcal{A}(\sigma); \pi \leftarrow \mathsf{Prove}(\sigma, s, w) : \mathsf{Verify}(\sigma, s, \pi) = 1 \text{ if } (s, w) \in R\big] = 1.$$

- **Computational Soundness**: An argument system is computationally sound if for all efficient adversaries $\mathcal{A}$,

$$\Pr\left[\sigma \leftarrow \mathsf{Setup}(1^\lambda); (s, \pi) \leftarrow \mathcal{A}(\sigma) : \mathsf{Verify}(\sigma, s, \pi) = 1 \text{ if } s \notin L\right] = \mathsf{negl}(\lambda).$$

Definition A.2 (Zero-Knowledge [FLS90, Gro06])**.** Let $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a non-interactive argument system for a relation R, and let L be the language of statements for R. We say that NIZK is computational zero-knowledge if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for all efficient non-uniform adversaries $\mathcal{A}$,

$$\left| \Pr\left[\sigma \leftarrow \mathsf{Setup}(1^\lambda) : \mathcal{A}^{\mathcal{O}(\sigma,\cdot,\cdot)}(\sigma) = 1\right] - \Pr\left[(\sigma,\tau) \leftarrow \mathcal{S}_1(1^\lambda) : \mathcal{A}^{\mathcal{O}'(\sigma,\tau,\cdot,\cdot)}(\sigma) = 1\right]\right| = \mathsf{negl}(\lambda),$$

where the oracles $\mathcal{O}$ and $\mathcal{O}'$ are defined as follows:

- $\mathcal{O}(\sigma,\cdot,\cdot)$ is the prover algorithm. On input (s,w), $\mathcal{O}$ outputs $\mathsf{Prove}(\sigma,s,w)$ if $(s,w) \in R$, and $\perp$ otherwise.
- $\mathcal{O}'(\sigma,\tau,\cdot,\cdot)$ is the simulator algorithm. On input (s,w), $\mathcal{O}'$ outputs $\mathcal{S}_2(\sigma,\tau,s)$ if $(s,w) \in R$, and $\perp$ otherwise.

In addition to the usual notions of completeness, soundness, and zero-knowledge, we also require our argument system to satisfy a stronger property known as simulation-sound extractability. Simulation soundness [Sah99] is the property that an argument (or proof) system remains sound even if the adversary sees "simulated" arguments (that is, arguments constructed by the zero-knowledge simulator). Next, in an argument of knowledge [DDO$^+$01, BG93], there is the additional requirement of an efficient knowledge extractor that on input a valid argument π of some statement s, is able to extract a witness w such that $(s,w) \in R$. An argument system is simulation-sound extractable if it is both simulation-sound and an argument of knowledge. More formally, we have:

Definition A.3 (Simulation-Sound Extractability [DDO$^+$01, Gro06])**.** Let $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a NIZK argument system for a relation R. Let $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ be the simulator associated with NIZK (Definition A.2). Then, NIZK satisfies the notion of simulation-sound extractability if there exists an extraction algorithm $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ such that the following holds:

- The output of $\mathcal{E}_1(1^\lambda)$ is identically distributed as $\mathcal{S}_1(1^\lambda)$ when restricted to the first two components (σ,τ).
- For all non-uniform polynomial-time adversaries $\mathcal{A}$,

$$\Pr\Big[(\sigma,\tau,\xi) \leftarrow \mathcal{E}_1(1^\lambda); (s,\pi) \leftarrow \mathcal{A}^{\mathcal{S}_2(\sigma,\tau,\cdot)}(\sigma); w \leftarrow \mathcal{E}_2(\sigma,\xi,s,\pi) : \\ (s,\pi) \notin Q \text{ and } (s,w) \notin R \text{ and } \mathsf{Verify}(\sigma,s,\pi) = 1\Big] = \mathsf{negl}(\lambda),$$

  where Q is the set containing the queries $\mathcal{A}$ makes to $\mathcal{S}_2$ and their responses, in the form of (query, response) pairs.

A.2 One-Way Permutations

We review the standard definition of one-way permutations (OWP) and hard-core functions.

Definition A.4 (One-Way Permutations [Gol01])**.** A family of one-way permutations OWP over a space $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ is a pair of efficient algorithms $(\mathsf{Setup}, \mathsf{Eval})$ with the following properties:

- **Correctness**: On input 1^λ, the setup algorithm $\mathsf{Setup}(1^\lambda)$ outputs a string t, such that the algorithm $\mathsf{Eval}(t,\cdot)$ computes a permutation over $\mathcal{X}_\lambda$. We denote this permutation by $h_t(\cdot)$.

- **One-Wayness**: For all efficient, non-uniform adversaries $\mathcal{A}$,

$$\Pr\left[t \leftarrow \mathsf{Setup}(1^\lambda); x \xleftarrow{\text{R}} \mathcal{X}_\lambda : \mathcal{A}(t, h_t(x)) = x\right] = \mathsf{negl}(\lambda).$$

Definition A.5 (Hard-Core Functions [Gol01]). Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be two collections of finite sets. Let $\mathsf{OWP} = (\mathsf{Setup}, \mathsf{Eval})$ be a family of one-way permutations over $\mathcal{X}$. Let hc be a polynomial-time computable function from $\mathcal{X}_\lambda$ to $\mathcal{Y}_\lambda$. Then, hc is a hard-core function for OWP if for all efficient non-uniform adversaries $\mathcal{A}$, $t \leftarrow \mathsf{Setup}(1^\lambda)$, and $x \xleftarrow{\text{R}} \mathcal{X}_\lambda$,

$$\left|\Pr[\mathcal{A}(t, h_t(x), \mathsf{hc}(x)) = 1] - \Pr\left[y \xleftarrow{\text{R}} \mathcal{Y}_\lambda : \mathcal{A}(t, h_t(x), y) = 1\right]\right| = \mathsf{negl}(\lambda),$$

where $h_t(\cdot) = \mathsf{Eval}(t, \cdot)$.

B Hybrid Argument Proofs from Section 4.2

B.1 Proof of Lemma 4.3

The only difference between hybrids Hyb_0 and Hyb_1 is that in the latter case, the challenger uses the NIZK simulator to construct the CRS in the master public key and the arguments in the ciphertexts. Thus, the claim follows directly from computational zero-knowledge of the NIZK scheme (Definition A.2).

Concretely, let $\mathcal{A}$ be an efficient distinguisher for hybrid experiments Hyb_0 and Hyb_1. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ to distinguish between the real and simulated distributions in Definition A.2. Algorithm $\mathcal{B}$ is given as input a CRS σ and has access to an oracle $\mathcal{O}(\cdot,\cdot)$ that generates arguments for statements in the language. In the security reduction, algorithm $\mathcal{B}$ simulates a challenger for $\mathcal{A}$ as follows.

- **Setup phase.** $\mathcal{B}$ runs $\mathsf{FE.Setup}(1^\lambda)$ to obtain $(\textsc{mpk}', \textsc{msk}')$, samples a one-way permutation $t \leftarrow \mathsf{OWP.Setup}(1^\lambda)$, and defines $h_t(\cdot) = \mathsf{OWP.Eval}(t, \cdot)$. Finally, $\mathcal{B}$ sets $\textsc{mpk} = (\textsc{mpk}', t, \sigma)$ and $\textsc{msk} = \textsc{msk}'$. It sends $\textsc{mpk}$ to $\mathcal{A}$.
- **Challenge phase.** When $\mathcal{B}$ receives a challenge vector $\mathbf{x} \in \mathcal{X}_\lambda^{q_c}$ from $\mathcal{A}$, it samples a key $k_i^* \xleftarrow{\text{R}} \mathcal{K}_\lambda$, and sets $\mathsf{ct}_i' \leftarrow \mathsf{FE.Encrypt}(\textsc{mpk}', (x_i, h_t(k_i^*)); \mathsf{hc}(k_i^*))$ for each $i \in [q_c]$. Let s_i be the statement in Eq. (3). $\mathcal{B}$ queries its oracle $\mathcal{O}$ on statement s_i and witness (x_i, k_i^*) to obtain an argument π_i. Finally it sets $\mathsf{ct}_i = (\mathsf{ct}_i', \pi_i)$ and sends $\{\mathsf{ct}_i\}_{i\in[q_c]}$ to $\mathcal{A}$.

The key-generation and decryption queries before and after the challenge phase are handled in the same way as in Hyb_0 and Hyb_1. At the end of the experiment, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. By construction, if the CRS and NIZK arguments are generated honestly, then $\mathcal{B}$ has perfectly simulated Hyb_0. If instead they were constructed by the NIZK simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, then $\mathcal{B}$ has perfectly simulated Hyb_1. Thus, the distinguishing advantage of $\mathcal{B}$ in the computational zero-knowledge experiment is equal to the advantage of $\mathcal{A}$ in distinguishing Hyb_0 from Hyb_1. The lemma follows. □

B.2 Proof of Lemma 4.4

By construction, the only difference between Hyb_1 and Hyb_2 is how the challenge ciphertexts $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_{q_c}^*$ are constructed. We introduce $q_c = \mathsf{poly}(\lambda)$ intermediate hybrids $\mathsf{Hyb}_{1,i}$ for $0 \le i \le q_c$. In $\mathsf{Hyb}_{1,i}$, the first $q_c - i$ ciphertexts $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_{q_c-i}^*$ are constructed as in hybrid Hyb_1 (with randomness derived from the hard-core function), and the remaining i ciphertexts $\mathsf{ct}_{q_c-i+1}^*, \ldots, \mathsf{ct}_{q_c}^*$ are generated as in Hyb_2 (with randomness drawn uniformly at random). By construction, $\mathsf{Hyb}_{1,0}$ is identical to Hyb_1 and Hyb_{1,q_c} is identical to Hyb_2.

We show that if OWP is one-way and hc is a hard-core function for it, hybrids $\mathsf{Hyb}_{1,i}$ and $\mathsf{Hyb}_{1,i+1}$ are computationally indistinguishable for $0 \le i < q_c$. Specifically, we show that if there exists a PPT distinguisher $\mathcal{A}$ for $\mathsf{Hyb}_{1,i}$ and $\mathsf{Hyb}_{1,i+1}$, then there exists a PPT adversary $\mathcal{B}$ that can distinguish the output of the hard-core function from uniform. $\mathcal{B}$ is given a challenge (t, z, T) where $z = h_t(k) = \mathsf{OWP.Eval}(t, k)$ for some $t \leftarrow \mathsf{OWP.Setup}(1^\lambda)$ and $k \xleftarrow{\mathrm{R}} \mathcal{K}$, and must decide whether $T = \mathsf{hc}(k)$ or T is uniformly random. In the reduction, algorithm $\mathcal{B}$ plays the role of the challenger to $\mathcal{A}$. In the setup phase, the behavior of $\mathcal{B}$ is identical to the behavior of the challenger in Hyb_1 and Hyb_2, except it uses the t from the challenge to construct the public parameters. The pre- and post-challenge phases are identical in Hyb_1 and Hyb_2 and can be perfectly simulated given $h_t(\cdot)$. The challenge phase is simulated as follows:

- **Challenge phase.** Let $\mathbf{x} \in \mathcal{X}_\lambda^{q_c}$ be the vector of challenge messages from $\mathcal{A}$. Adversary $\mathcal{B}$ constructs the challenge ciphertexts as follows:

  1. For all $j < q_c - i$, construct ct_j^* as described in hybrid Hyb_1.
  2. For all $q_c - i < j \le q_c$, construct ct_j^* as described in hybrid Hyb_2.
  3. Let $\mathsf{ct}'_{q_c-i} = \mathsf{FE.Encrypt}(\textsc{mpk}', (x_{q_c-i}, z); T)$ be an encryption of (x_{q_c-i}, z) using randomness T (where z, T are from the challenge). Simulate an argument π_{q_c-i} by invoking $\mathcal{S}_2^{(\textsc{nizk})}(\sigma, \tau, s_{q_c-i})$, where s_{q_c-i} is the statement from Eq. (3). Set $\mathsf{ct}^*_{q_c-i} = (\mathsf{ct}'_{q_c-i}, \pi_{q_c-i})$ and send $\{\mathsf{ct}_i^*\}_{i\in[q_c]}$ to $\mathcal{A}$.

At the end of the experiment, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. When $T = \mathsf{hc}(k)$, the first $q_c - i$ ciphertexts are constructed as described in Hyb_1, while the rest are constructed as described in Hyb_2. This corresponds to hybrid $\mathsf{Hyb}_{1,i}$. Conversely, if T is uniform, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{1,i+1}$. Thus, if $\mathcal{A}$ can distinguish $\mathsf{Hyb}_{1,i}$ from $\mathsf{Hyb}_{1,i+1}$ with non-negligible probability, $\mathcal{B}$ can distinguish the output of the hard-core function from uniform with the same probability. $\square$

B.3 Proof of Lemma 4.5

Hybrids Hyb_2 and Hyb_3 are identical except in how the challenger responds to decryption queries, but assuming simulation-sound extractability of NIZK and perfect-correctness of FE, we can show that they are computationally indistinguishable. Let (f, C) be a decryption query, where $C = \{\mathsf{ct}_i\}_{i\in[m]}$. Write each ct_i as (ct'_i, π_i).

There are two cases. If the argument π_i does not verify, the challenger sets $y_i = \perp$ in both Hyb_2 and Hyb_3. Otherwise, we know that the adversary cannot submit any of its challenge ciphertexts in one of its decryption queries. Thus, the pair (ct'_i, π_i) was not generated by the challenger using $\mathcal{S}^{(\textsc{nizk})}$. Hence, by simulation-sound extractability, with probability at least $1 - \mathsf{negl}(\lambda)$, the extraction algorithm $\mathcal{E}_2^{(\textsc{nizk})}$ on ct'_i and π_i will produce a witness (x_i, k_i) such that $\mathsf{ct}'_i = \mathsf{FE.Encrypt}(\textsc{mpk}', (x_i, h_t(k_i)); \mathsf{hc}(k_i))$. Moreover, by perfect correctness of the underlying FE scheme, $(x_i, h_t(k_i))$ is the only pair that encrypts to ct'_i.

In both Hyb_2 and Hyb_3, the challenger first samples a key $k \xleftarrow{\mathrm{R}} \mathcal{K}_\lambda$. In Hyb_2, it computes $\mathsf{sk} \leftarrow \mathsf{FE.KeyGen}(\textsc{msk}, g_k^f)$, and then sets y_i to be $\mathsf{FE.Decrypt}(\mathsf{sk}, \mathsf{ct}'_i)$. Again, by perfect correctness of the underlying FE scheme,

$$y_i = g_k^f(x_i, h_t(k_i)) = f(x_i; \mathsf{PRF}(k \diamond h_t(k_i), x_i)),$$

which is exactly what is output in Hyb_3 after (x_i, k_i) is extracted. We conclude that with probability $1 - \mathsf{negl}(\lambda)$, the response to each decryption query in Hyb_2 and Hyb_3 is identically distributed. $\square$

B.4 Proof of Lemma 4.6

Suppose $\mathcal{A}$ is a distinguisher for Hyb_3 and Hyb_4. We use $\mathcal{A}$ to construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that distinguishes between the experiments $\mathsf{Real}_{\mathcal{B}}^{\mathsf{FE}}$ and $\mathsf{Ideal}_{\mathcal{B}}^{\mathsf{FE}}$. In the reduction, $\mathcal{B}$ will simulate the role of the challenger in Hyb_3 and Hyb_4 to $\mathcal{A}$. Moreover, it has access to the following oracles:

- $\mathcal{B}_1$ has access to a pre-challenge key-generation oracle $\mathcal{O}_{\mathsf{KeyGen}}^{(\mathsf{pre})}(\cdot)$ that corresponds to $\mathcal{O}_1(\text{MSK}, \cdot)$ in the real experiment and $\mathcal{O}_1'(\mathsf{st}', \cdot)$ in the ideal one.
- $\mathcal{B}_2$ has access to a post-challenge key-generation oracle $\mathcal{O}_{\mathsf{KeyGen}}^{(\mathsf{post})}(\cdot)$ that corresponds to $\mathcal{O}_2(\text{MSK}, \cdot)$ in the real experiment and $\mathcal{O}_2'(\mathsf{st}', \cdot)$ in the ideal one.

We now specify the operation of $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$:

Algorithm $\mathcal{B}_1(\text{MPK}')$. On input a public key MPK' for FE, the setup and pre-challenge query phases are simulated as follows:

- **Setup phase.** $\mathcal{B}_1$ runs $\mathcal{E}_1^{(\text{NIZK})}(1^\lambda)$ to obtain a simulated CRS σ, a simulation trapdoor τ, and an extraction trapdoor ξ. It then samples a one-way permutation $t \leftarrow \mathsf{OWP.Setup}(1^\lambda)$ and defines $h_t(\cdot) = \mathsf{OWP.Eval}(t, \cdot)$. Finally, it sets $\text{MPK} = (\text{MPK}', t, \sigma)$ and sends MPK to $\mathcal{A}$.
- **Pre-challenge queries.** Decryption queries are handled exactly as described in Hyb_3 and Hyb_4.
  - **Key-generation queries.** On input $f \in \mathcal{F}_\lambda$, $\mathcal{B}_1$ samples a key $k \xleftarrow{\text{R}} \mathcal{K}$, and queries $\mathcal{O}_{\mathsf{KeyGen}}^{(\mathsf{pre})}$ on g_k^f to obtain a key sk. It gives sk to $\mathcal{A}$.

When $\mathcal{A}$ outputs its challenge vector $\mathbf{x}' \in \mathcal{X}_\lambda^{q_c}$, $\mathcal{B}_1$ saves the current execution state $\mathsf{st}_{\mathcal{A}}$ of $\mathcal{A}$. Then for each $i \in [q_c]$, $\mathcal{B}_1$ samples a key $k_i^* \xleftarrow{\text{R}} \mathcal{K}_\lambda$ and sets $x_i = (x_i', h_t(k_i^*))$. It also sets $\mathsf{st} = (\mathsf{st}_{\mathcal{A}}, \{k_i^*\}_{i \in [q_c]})$ and outputs $\mathbf{x} = (x_1, \ldots, x_{q_c})$ along with st.

Algorithm $\mathcal{B}_2(\text{MPK}', \{\mathsf{ct}_i'\}_{i \in [q_c]}, \mathsf{st})$. On input the master public key MPK' and challenge ciphertexts $\{\mathsf{ct}_i'\}_{i \in [q_c]}$ for FE, and a state $\mathsf{st} = (\mathsf{st}_{\mathcal{A}}, \{k_i^*\}_{i \in [q_c]})$, $\mathcal{B}_2$ first resumes the execution of $\mathcal{A}$ using $\mathsf{st}_{\mathcal{A}}$. Then, it simulates the challenge phase and post-challenge queries as follows:

- **Challenge phase.** For each $i \in [q_c]$, $\mathcal{B}_2$ runs $\mathcal{S}_2^{(\text{NIZK})}(\sigma, \tau, s_i)$ to obtain an argument π_i and sets $\mathsf{ct}_i^* = (\mathsf{ct}_i', \pi_i)$. (As in Hyb_3 and Hyb_4, s_i is the statement from Eq. (3).) It sends $\{\mathsf{ct}_i^*\}_{i \in [q_c]}$ to $\mathcal{A}$.
- **Post-challenge queries.** They are handled in the same way as pre-challenge ones, except that $\mathcal{B}_2$ queries $\mathcal{O}_{\mathsf{KeyGen}}^{(\mathsf{post})}$ whenever $\mathcal{B}_1$ queried $\mathcal{O}_{\mathsf{KeyGen}}^{(\mathsf{pre})}$.

At the end of the experiment, $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which $\mathcal{B}_2$ simply echoes. We now show that if $\mathcal{B}$ is interacting in the real experiment $\mathsf{Real}_{\mathcal{B}}^{\mathsf{FE}}$, then the view it simulates for $\mathcal{A}$ is computationally indistinguishable from Hyb_3. Conversely, if $\mathcal{B}$ is in the ideal experiment $\mathsf{Ideal}_{\mathcal{B}}^{\mathsf{FE}}$, then the view it simulates for $\mathcal{A}$ is computationally indistinguishable from Hyb_4. The claim then follows from the assumption that FE is (q_1, q_c, q_2)-SIM-secure. We consider each case separately:

Real experiment. Suppose $\mathcal{B}$ is interacting in the real experiment $\mathsf{Real}_{\mathcal{B}}^{\mathsf{FE}}$. We show that in this case, $\mathcal{B}$ simulates Hyb_3 for $\mathcal{A}$.

- **Setup phase.** In the real experiment, the master public key MPK' is generated by calling $\mathsf{FE.Setup}$. This is the how MPK' is obtained in Hyb_3. The remainder of the setup procedure is identical to that in Hyb_3.
- **Pre-challenge queries.** We consider the two types of queries separately.
  - **Key-generation queries.** In the real experiment, on input a function g_k^f, the key-generation oracle $\mathcal{O}_1(\text{MSK}, \cdot)$ returns $\mathsf{FE.KeyGen}(\text{MSK}, g_k^f)$. In Hyb_3, when the adversary makes a query with a randomized function $f \in \mathcal{F}_\lambda$, it receives the output of $\mathsf{KeyGen}(\text{MSK}, f)$, which is nothing but $\mathsf{FE.KeyGen}(\text{MSK}, g_k^f)$ for $k \xleftarrow{\text{R}} \mathcal{K}_\lambda$.
  - **Decryption queries.** We note that $\mathcal{B}$ knows all the quantities needed to respond to decryption queries as specified in Hyb_3.
- **Challenge phase.** In the reduction, $\mathcal{B}_1$ chooses $k_i^* \xleftarrow{\text{R}} \mathcal{K}_\lambda$ and outputs $x_i = (x_i', h_t(k_i^*))$ for each $i \in [q_c]$. In $\mathsf{Real}_{\mathcal{B}}^{\mathsf{FE}}$, $\mathcal{B}_2$ is given the set of ciphertexts $\mathsf{ct}_i' = \mathsf{FE.Encrypt}(\text{MPK}', x_i)$, and it outputs $\{(\mathsf{ct}_i', \pi_i)\}_{i \in [q_c]}$ by computing π_i using the NIZK simulator. This is exactly how the challenge ciphertexts are produced in Hyb_3.
- **Post-challenge queries.** Using the same argument as in the pre-challenge phase, we conclude that the queries in the post-challenge phase are correctly simulated.

Ideal experiment. Suppose $\mathcal{B}$ is interacting in the ideal experiment $\mathsf{Ideal}_{\mathcal{B}}^{\mathsf{FE}}$. We show that in this case, $\mathcal{B}$ simulates Hyb_4 for $\mathcal{A}$.

- **Setup phase.** In the ideal experiment, the master public key MPK' is generated by calling $\mathcal{S}_1^{(\text{FE})}$. This is the how MPK' is obtained in Hyb_4. The remainder of the setup procedure is identical to that in Hyb_4.
- **Pre-challenge queries.** We consider the two types of queries separately.
  - **Key-generation queries.** When $\mathcal{A}$ makes a query $f \in \mathcal{F}_\lambda$, $\mathcal{B}_1$ forwards g_k^f to $\mathcal{O}_{\mathsf{KeyGen}}^{(\mathsf{pre})}$ in the reduction, where $k \xleftarrow{\text{R}} \mathcal{K}_\lambda$. In $\mathsf{Ideal}_{\mathcal{B}}^{\mathsf{FE}}$, $\mathcal{B}_1$ receives the output of $\mathcal{S}_2^{(\text{FE})}(\mathsf{st}', g_k^f)$, which it then forwards to $\mathcal{A}$. This is precisely the behavior in Hyb_4.
  - **Decryption queries.** $\mathcal{B}_1$ answers the decryption queries exactly as prescribed in Hyb_4.
- **Challenge phase.** In the reduction, $\mathcal{B}_1$ constructs the challenge vector $\mathbf{x} = (x_1, \dots, x_{q_c})$ where $x_i = (x_i', h_t(k_i^*))$ and $k_i^* \xleftarrow{\text{R}} \mathcal{K}_\lambda$ for all $i \in [q_c]$. Let $f_1, \dots, f_{q_1} \in \mathcal{F}_\lambda$ be the pre-challenge key-generation queries submitted by $\mathcal{A}$, and $k_1, \dots, k_{q_1} \in \mathcal{K}_\lambda$ be the keys sampled by $\mathcal{B}_1$ when responding to them. Thus, $\mathcal{B}_1$ queried $\mathcal{O}_{\mathsf{KeyGen}}^{(\mathsf{pre})}$ on the derandomized functions $g_{k_1}^{f_1}, \dots, g_{k_{q_1}}^{f_{q_1}}$. In $\mathsf{Ideal}_{\mathcal{B}}^{\mathsf{FE}}$, the ciphertexts $\{\mathsf{ct}_i'\}_{i \in [q_c]}$ are constructed by invoking the simulator algorithm $\mathcal{S}_3^{(\text{FE})}$ on the state st' and function evaluations $\{y_{ij}\}_{i \in [q_c], j \in [q_1]}$. Here, for $i \in [q_c]$ and $j \in [q_1]$, we have that

$$y_{ij} = g_{k_j}^{f_j}(x_i) = g_{k_j}^{f_j}(x_i', h_t(k_i^*)) = f_j(x_i'; \mathsf{PRF}(k_j \diamond h_t(k_i^*), x_i')).$$

This is precisely how the values y_{ij} are constructed in Hyb_4. Then, $\mathcal{B}_2$ is given the simulated ciphertexts $\{\mathsf{ct}'_i\}_{i\in[q_c]}$ produced by $\mathcal{S}_3^{(\mathrm{FE})}$, and it outputs $\{(\mathsf{ct}'_i,\pi_i)\}_{i\in[q_c]}$ by computing π_i using the NIZK simulator. This is exactly how $\mathcal{S}_3$ behaves in Hyb_4.

- **Post-challenge queries.** We consider the two types of queries separately.
  - **Key-generation queries.** When $\mathcal{A}$ makes a query $f \in \mathcal{F}_\lambda$, $\mathcal{B}_2$ forwards g_k^f to $\mathcal{O}_{\mathsf{KeyGen}}^{(\mathsf{post})}$, where $k \xleftarrow{\mathrm{R}} \mathcal{K}_\lambda$. In $\mathsf{Ideal}_{\mathcal{B}}^{\mathsf{FE}}$, $\mathcal{B}_2$ receives the output of $\mathcal{S}_4^{(\mathrm{FE})}(\mathsf{st}', g_k^f)$, which it then forwards to $\mathcal{A}$. Lastly we note that the $\mathsf{FE.KeyIdeal}$ oracle in Hyb_4 is simulated exactly as in $\mathsf{Ideal}_{\mathcal{B}}^{\mathsf{FE}}$.
  - **Decryption queries.** $\mathcal{B}_2$ answers the decryption queries exactly as prescribed in Hyb_4. □

B.5 Proof of Lemma 4.7

Let q_d be the number of decryption queries the adversary makes (across both the pre-challenge and post-challenge phases of the experiment). We define some intermediate hybrids:

Hybrid $\mathsf{Hyb}_{4,i}$. For each $0 \le i \le q_d$, the challenger responds to the first i decryption queries as specified in Hyb_5, and the remaining decryption queries as specified in Hyb_4. Rest of the hybrid remains same as Hyb_4.

Hybrid $\mathsf{Hyb}'_{4,j}$. For $0 \le j \le q_c$, $\mathsf{Hyb}'_{4,j}$ is identical to Hyb_{4,q_d}, except the challenger proceeds as follows in the challenge phase and when responding to a post-challenge key-generation query:

- **Challenge phase.** Let $f_1,\ldots,f_{q_1} \in \mathcal{F}_\lambda$ be the pre-challenge key-generation queries $\mathcal{A}$ makes, and let $k_1,\ldots,k_{q_1} \in \mathcal{K}_\lambda$ be the keys the challenger used to respond to each query. For each $i \in [q_c]$ and $\ell \in [q_1]$, the challenger chooses $r_{i\ell} \xleftarrow{\mathrm{R}} \mathcal{R}_\lambda$ and $k_i^* \xleftarrow{\mathrm{R}} \mathcal{K}_\lambda$ and sets

$$y_{i\ell} = \begin{cases} f_\ell(x_i; r_{i\ell}) & \text{if } i \le j \\ f_\ell(x_i; \mathsf{PRF}(k_\ell \diamond h_t(k_i^*), x_i)) & \text{otherwise,} \end{cases}$$

  It then carries out the third and fourth steps of the challenge phase of Hyb_4, which are same as that of Hyb_5.

- **Post-challenge queries.** The challenger replies to decryption queries as described in Hyb_{4,q_d}. For a key-generation query, we only describe how the oracle $\mathsf{FE.KeyIdeal}(\mathbf{x},\cdot)$ is implemented. The rest of the procedure is identical to that in Hyb_4 and Hyb_5. On an input $g_{f'}^{k'}$ to the $\mathsf{FE.KeyIdeal}(\mathbf{x},\cdot)$ oracle, the challenger chooses $r_i \xleftarrow{\mathrm{R}} \mathcal{R}_\lambda$ for each $i \in [q_c]$. Then, it sets

$$y_i = \begin{cases} f'(x_i; r_i) & \text{if } i \le j \\ f'(x_i; \mathsf{PRF}(k' \diamond h_t(k_i^*), x_i)) & \text{otherwise.} \end{cases}$$

  As usual, it replies with the set $\{y_i\}_{i\in[q_c]}$.

By construction, hybrids Hyb_4 and $\mathsf{Hyb}_{4,0}$ are identical to each other; and so are Hyb_{4,q_d} and $\mathsf{Hyb}'_{4,0}$ as well as Hyb'_{4,q_c} and Hyb_5.

Claim B.1. *If* PRF *is* $\Phi_\diamond$*-RKA secure, then for all* $0 \le i < q_d$*, hybrids* $\mathsf{Hyb}_{4,i}$ *and* $\mathsf{Hyb}_{4,i+1}$ *are computationally indistinguishable.*

Proof. Let $\mathcal{A}$ be a distinguisher for hybrids $\mathsf{Hyb}_{4,i}$ and $\mathsf{Hyb}_{4,i+1}$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that distinguishes between the real and ideal distributions in the $\Phi_\diamond$-RKA security game, where it has access to an evaluation oracle $\mathcal{O}_{\mathsf{Eval}}(\cdot,\cdot)$. In the reduction, $\mathcal{B}$ simulates the role of the challenger for $\mathcal{A}$. Since the setup, challenge, pre- and post-challenge key-generation phases stay the same across the hybrids $\mathsf{Hyb}_{4,0},\dots,\mathsf{Hyb}_{4,q_d}$, $\mathcal{B}$ simulates them in the same way. The decryption queries are handled differently as shown below.

- **Decryption queries.** Let $(f, C = \{\mathsf{ct}_j\}_{j\in[m]})$ be $\mathcal{A}$'s decryption query. Let q be the total number of queries $\mathcal{A}$ has made so far. If $q < i$, then $\mathcal{B}$ replies as described in Hyb_4, and if $q > i$, it replies as in Hyb_5. If $q = i$, then $\mathcal{B}$ does the following for each $j \in [m]$:

  1. Parse ct_j as (ct'_j, π_j), and let s_j be the statement from Eq. (3). If π_j is not a valid argument for s_j, then set $y_j = \perp$. Otherwise, invoke the extractor $\mathcal{E}_2^{(\mathrm{NIZK})}(\sigma,\xi,s_j,\pi_j)$ to obtain a witness (x_j,k_j).
  2. Define a key-transformation function $\phi_j : \mathcal{K}_\lambda \to \mathcal{K}_\lambda$ where $\phi_j(k) = k \diamond h_t(k_j)$. Then, let $r_j \leftarrow \mathcal{O}_{\mathsf{Eval}}(\phi_j, x_j)$, and set $y_j = f(x_j; r_j)$.

  Finally, algorithm $\mathcal{B}$ outputs the ordered set $\{y_j\}_{j\in[m]}$.

At the end of the experiment, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. We claim that if $\mathcal{B}$ is interacting in the real experiment of $\Phi_\diamond$-RKA security game, then it perfectly simulates $\mathsf{Hyb}_{4,i}$ for $\mathcal{A}$. Conversely, if $\mathcal{B}$ is in the ideal experiment, then it perfectly simulates $\mathsf{Hyb}_{4,i+1}$ for $\mathcal{A}$. We consider both cases in detail:

- In the real world, $\mathcal{O}_{\mathsf{Eval}}(\phi, x) = \mathsf{PRF}(\phi(k), x)$ for a randomly chosen key $k \in \mathcal{K}_\lambda$. Then, on the i^{th} decryption query, for all $j \in [m]$, $r_j = \mathsf{PRF}(k \diamond h_t(k_j), x_j)$. This is exactly how randomness is sampled in $\mathsf{Hyb}_{4,i}$.

- In the ideal world, $\mathcal{O}_{\mathsf{Eval}}(\phi, x) = F(\phi, x)$ for $F \xleftarrow{\text{R}} \mathsf{Funs}[\Phi \times \mathcal{X}_\lambda, \mathcal{Y}_\lambda]$. Since we require that $\mathsf{ct}_j \not\sim \mathsf{ct}_\ell$ for all $j \ne \ell$ in a decryption query, it must be the case that if ct_j and ct_ℓ are valid ciphertexts (i.e., π_j and π_ℓ are valid arguments of their respective statements s_j and s_ℓ), then either $k_j \ne k_\ell$ or $x_j \ne x_\ell$. This latter fact follows from the fact that an efficient adversary can only find a single *valid* ciphertext for each pair (x,k). Since F is a truly random function from $\Phi \times \mathcal{X}_\lambda$ to $\mathcal{R}_\lambda$, r_j must be uniform over $\mathcal{R}_\lambda$ for all $j \in [m]$. This corresponds to the distribution in $\mathsf{Hyb}_{4,i+1}$.

We conclude that if $\mathcal{A}$ can distinguish hybrids $\mathsf{Hyb}_{4,i}$ from $\mathsf{Hyb}_{4,i+1}$ for any $0 \le i < q_d$, then $\mathcal{B}$ can break the $\Phi_\diamond$-RKA security of PRF with the same advantage. The claim follows. □

Claim B.2. *If* PRF *is* $\Phi_\diamond$*-RKA secure and* FE *is a* (q_1, q_c, q_2)*-*SIM*-secure functional encryption scheme for* $\mathcal{G}_\mathcal{F}$*, then for all* $0 \le i < q_c$*, hybrids* $\mathsf{Hyb}'_{4,j}$ *and* $\mathsf{Hyb}'_{4,j+1}$ *are computationally indistinguishable.*

Proof. The proof of this claim is similar to the previous one. Specifically, if $\mathcal{A}$ is a distinguisher for hybrids $\mathsf{Hyb}'_{4,j}$ and $\mathsf{Hyb}'_{4,j+1}$, then we can construct an adversary $\mathcal{B}$ that breaks the $\Phi_\diamond$-RKA security of PRF. As before, $\mathcal{B}$ has access to an evaluation oracle $\mathcal{O}_{\mathsf{Eval}}(\cdot,\cdot)$. We only focus on the challenge and post-challenge phases below; the rest are carried out in the same way as in $\mathsf{Hyb}'_{4,j}$ or $\mathsf{Hyb}'_{4,j+1}$.

- **Challenge phase.** When $\mathcal{B}$ receives a vector $\mathbf{x} \in \mathcal{X}_\lambda^{q_c}$ from $\mathcal{A}$, it proceeds as follows:

1. Let $f_1, \dots, f_{q_1}$ be the pre-challenge key-generation queries made by $\mathcal{A}$, and let $k_1, \dots, k_{q_1} \in \mathcal{K}_\lambda$ be the keys $\mathcal{B}$ used to responds to each such query.
2. For each $i \in [q_c]$, choose $k_i^* \xleftarrow{\text{R}} \mathcal{K}_\lambda$.
3. For all $i \le j$ and $\ell \in [q_1]$, choose $r_{i\ell} \xleftarrow{\text{R}} \mathcal{R}_\lambda$. For all $i > j+1$ and $\ell \in [q_1]$, set $r_{i\ell} = \mathsf{PRF}(k_\ell \diamond h_t(k_i^*), x_i)$.
4. For $\ell \in [q_1]$, define the key-transformation function $\phi_\ell : \mathcal{K}_\lambda \to \mathcal{K}_\lambda$ where $\phi_\ell(k) = k \diamond k_\ell$. For all $\ell \in [q_1]$, let $r_{j+1,\ell} = \mathcal{O}_{\mathsf{Eval}}(\phi_\ell, x_{j+1})$.
5. For all $i \in [q_c]$ and $\ell \in [q_1]$, let $y_{i\ell} = f_\ell(x_i; r_{i\ell})$. Invoke the simulator theorem $\mathcal{S}_3(\mathsf{st}, \{y_{ij}\}_{i \in [q_c], j \in [q_1]})$ to obtain a collection of ciphertexts $\{\mathsf{ct}_i^*\}_{i \in [q_c]}$ and an updated state st. Send $\{\mathsf{ct}_i^*\}_{i \in [q_c]}$ to $\mathcal{A}$.

- **Post-challenge queries.** Algorithm $\mathcal{B}$ responds to each query as follows:
  - **Key-generation queries.** Let $f \in \mathcal{F}_\lambda$ be the key-generation query. Algorithm $\mathcal{B}$ then does the following:
    1. Choose a random key $k \xleftarrow{\text{R}} \mathcal{K}_\lambda$, and define the derandomized functionality g_k^f as in Eq. (1).
    2. Invoke $\mathcal{S}_4^{(\mathrm{FE})}(\mathsf{st}^{(\mathrm{FE})}, g_k^f)$. Algorithm $\mathcal{B}$ simulates the $\mathsf{FE.KeyIdeal}(\mathbf{x}, \cdot)$ oracle in the following way. On input a derandomized functionality of the form $g_{k'}^{f'}$, the challenger does the following for each $i \in [q_c]$.
       * If $i \le j$, choose $r_i \xleftarrow{\text{R}} \mathcal{R}_\lambda$. If $i > j+1$, let $r_i = \mathsf{PRF}(k' \diamond h_t(k_i^*), x_i)$.
       * If $i = j+1$, define the key-transformation function $\phi : \mathcal{K}_\lambda \to \mathcal{K}_\lambda$ where $\phi(k) = k \diamond k'$. Let $r_i \leftarrow \mathcal{O}_{\mathsf{Eval}}(\phi, x_i)$.
    3. Output $\{f'(x_i; r_i)\}_{i \in [q_c]}$.
  - **Decryption queries**. Same as in Hyb_5.

At the end of the experiment, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. We claim that if $\mathcal{B}$ is interacting in the real experiment, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}'_{4,j}$ and if $\mathcal{B}$ is interacting in the ideal experiment, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}'_{4,j+1}$ for $\mathcal{A}$. We consider both cases:

- Suppose $\mathcal{B}$ is interacting in the real world, in which case $\mathcal{O}_{\mathsf{Eval}}(\phi, x) = \mathsf{PRF}(\phi(k), x)$, where $k \in \mathcal{K}_\lambda$ is a uniformly random PRF key. In the challenge phase, for all $\ell \in [q_1]$, $r_{j+1,\ell} = \mathcal{O}_{\mathsf{Eval}}(\phi_\ell, x_{j+1}) = \mathsf{PRF}(k \diamond k_\ell, x_{j+1})$. When responding to the post-challenge key-generation queries, $r_{j+1} = \mathcal{O}_{\mathsf{Eval}}(\phi, x_{j+1}) = \mathsf{PRF}(k \diamond k', x_{j+1})$. The PRF key k plays the role of $h_t(k_{j+1}^*)$ in the simulation. Note that since h_t is a permutation and k_{j+1}^* is sampled uniformly from $\mathcal{K}_\lambda$, $h_t(k_{j+1}^*)$ is uniformly distributed in the real scheme. We conclude that $\mathcal{B}$ has perfectly simulated $\mathsf{Hyb}'_{4,j}$.

- Suppose $\mathcal{B}$ is interacting in the ideal world, in which case $\mathcal{O}_{\mathsf{Eval}}(\phi, x) = F(\phi, x)$ where $F \xleftarrow{\text{R}} \mathsf{Funs}[\Phi \times \mathcal{X}_\lambda, \mathcal{Y}_\lambda]$. Let q_1 be the number of pre-challenge key-generation queries $\mathcal{A}$ makes and let q_2 be the number of post-challenge key-generation queries $\mathcal{A}$ makes. Let $m = q_1 + q_2$, and $k_1, \dots, k_m \in \mathcal{K}_\lambda$ be the keys algorithm $\mathcal{B}$ samples in response to each key-generation query (across both phases). Since all of the keys $k_1, \dots, k_m$ are sampled independently and $m = \mathsf{poly}(\lambda)$, with probability $1 - \mathsf{negl}(\lambda)$, all of the keys $k_1, \dots, k_m$ are distinct. For $\ell \in [m]$, let $\phi_\ell : \mathcal{K}_\lambda \to \mathcal{K}_\lambda$ be the key-transformation function $\phi_\ell(k) = k \diamond k_\ell$.

For $1 \le \ell \le q_1$, let $r_\ell = r_{j+1,\ell} = \mathcal{O}_{\mathsf{Eval}}(\phi_\ell, x_{j+1})$ be the randomness $\mathcal{B}$ uses in the challenge phase of the simulation to compute the value $y_{j+1,\ell} = f_\ell(x_{j+1}; r_{j+1,\ell})$. Similarly, for $q_1 + 1 \le \ell \le q_1 + q_2$, let $r_\ell = \mathcal{O}_{\mathsf{Eval}}(\phi_\ell, x_{j+1})$ be the randomness used to evaluate f_ℓ on x_{j+1} when simulating the FE.KeyIdeal oracle on the ℓ^{th} key-generation query (equivalently, the $(\ell - q_1)^{\text{th}}$ post-challenge key-generation query). Note that since $\mathcal{S}^{(\mathsf{FE})}$ is a simulator for an FE scheme and $\mathcal{S}_4^{(\mathsf{FE})}$ is only invoked on the derandomized functionalities $g_{k_{q_1+1}}^{f_{q_1+1}}, \ldots, g_{k_{q_1+q_2}}^{f_{q_2}}$, these derandomized functionalities are the only legal queries that $\mathcal{S}_4^{(\mathsf{FE})}$ can make to FE.KeyIdeal. Here, we rely on the fact that $\mathcal{S}^{(\mathsf{FE})}$ is a simulator for a (q_1, q_c, q_2)-secure functional encryption scheme (Definition 2.5). Specifically, Definition 2.5 requires that the (ordered) set of key-generation queries $\{f\}$ the simulator $\mathcal{S}^{(\mathsf{FE})}$ is invoked on is computationally indistinguishable from the (ordered) set of functions $\{f'\}$ the simulator submits to the FE.KeyIdeal oracle.

Since $k_1, \ldots, k_m$ are distinct with probability $1 - \mathsf{negl}(\lambda)$, the functions $\phi_1, \ldots, \phi_m$ are distinct with the same probability. Finally, since F is uniform over $\mathsf{Funs}[\Phi_\diamond \times \mathcal{X}_\lambda, \mathcal{R}_\lambda]$, we have that $r_1, \ldots, r_m$ are uniform in $\mathcal{R}_\lambda$. This is precisely the distribution $\mathsf{Hyb}'_{4,j+1}$.

Thus, we conclude that if $\mathcal{A}$ can distinguish $\mathsf{Hyb}'_{4,j}$ from $\mathsf{Hyb}'_{4,j+1}$ for any $0 \le j < q_c$, then $\mathcal{B}$ can break the $\Phi_\diamond$-RKA security of PRF with the same advantage. The claim follows. □

Combining Claims B.1 and B.2, we conclude that if PRF is $\Phi_\diamond$-RKA secure, then hybrids Hyb_4 and Hyb_5 are computationally indistinguishable. □

C Correctness Proof

The correctness proof follows from completeness of the NIZK argument system, correctness of the underlying FE scheme, and related-key security of the PRF. Take any collection of $n = n(\lambda)$ functions $f_1, \ldots, f_n \in \mathcal{F}_\lambda$ and n points $x_1, \ldots, x_n \in \mathcal{X}_\lambda$. We now proceed via a hybrid argument.

Hybrid Hyb_0. This is the real distribution (Definition 3.1).

Hybrid Hyb_1. Same as Hyb_0, except on input MPK, sk, and $\mathsf{ct} = (\mathsf{ct}', \pi)$, the decryption algorithm Decrypt simply outputs $\mathsf{FE.Decrypt}(\mathsf{sk}, \mathsf{ct}')$ *without* verifying the argument π.

Hybrid Hyb_2. This is the ideal distribution, except the functions are evaluated using pseudorandom strings rather than truly random strings. More precisely, this is the distribution $\{f_i(x_j; r_{i,j})\}_{i,j \in [n]}$ where

1. For all $i, j \in [n]$, $k_i \xleftarrow{\text{R}} \mathcal{K}_\lambda$ and $k'_j \xleftarrow{\text{R}} \mathcal{K}_\lambda$.
2. For all $i, j \in [n]$, $r_{i,j} \leftarrow \mathsf{PRF}(k_i \diamond k'_j, x_j)$

Hybrid Hyb_3. This is the ideal distribution (Definition 3.1).

Lemma C.1. *If* NIZK *is perfectly complete (Definition A.1), then hybrids* Hyb_0 *and* Hyb_1 *are identical.*

Proof. Since the CRS σ and the arguments π are all generated honestly, by perfect completeness, whenever the decryption algorithm invokes $\mathsf{NIZK.Verify}$ in Hyb_0, the output is always 1. Thus, Hyb_0 and Hyb_1 are identical. □

Lemma C.2. *If* FE *is a perfectly-correct functional encryption scheme (Definition 2.4), then hybrids* Hyb_1 *and* Hyb_2 *are identical.*

Proof. This follows from correctness of the underlying FE scheme for deterministic functionalities. Consider the distribution in Hyb_1. Write $\textsc{mpk} = (\textsc{mpk}', t, \sigma)$ and for each $j \in [n]$, write $\mathsf{ct}_j = (\mathsf{ct}'_j, \pi_j)$. By construction, sk_i is a secret key for FE corresponding to the deterministic function $g_{k_i}^{f_i}$ where $k_i \xleftarrow{\text{R}} \mathcal{K}_\lambda$. Similarly, ct_j is some FE encryption of the message $(x_j, h_t(k'_j))$ where $k'_j \xleftarrow{\text{R}} \mathcal{K}_\lambda$. Since $\mathsf{Decrypt}(\textsc{mpk}, \mathsf{sk}_i, \mathsf{ct}_j)$ simply outputs $\mathsf{FE.Decrypt}(\mathsf{sk}_i, \mathsf{ct}_j)$, we have, by perfect correctness of FE,

$$\left\{\mathsf{Decrypt}(\textsc{mpk}, \mathsf{sk}_{f_i}, \mathsf{ct}_j)\right\}_{i,j\in[n]} \equiv \left\{g_{k_i}^{f_i}(x_j, h_t(k'_j))\right\}_{i,j\in[n]}.$$

By definition of g_k^f from Eq. (1),

$$\left\{g_{k_i}^{f_i}(x_j, h_t(k'_j))\right\}_{i,j\in[n]} \equiv \left\{f_i(x_j; \mathsf{PRF}(k_i \diamond h_t(k'_j), x_j))\right\}_{i,j\in[n]},$$

where k_i, k'_j are uniformly random over $\mathcal{K}_\lambda$ for all $i, j \in [n]$. Finally, since h_t is a permutation and k'_j is uniform over $\mathcal{K}_\lambda$, $h_t(k'_j)$ is correspondingly uniform over $\mathcal{K}_\lambda$. This yields the distribution in Hyb_2. □

Lemma C.3. *If* PRF *is* $\Phi_\diamond$*-RKA secure, then hybrids* Hyb_2 *and* Hyb_3 *are computationally indistinguishable.*

Proof. We introduce $n = \mathsf{poly}(\lambda)$ intermediate hybrids $\mathsf{Hyb}_{2,i}$ for $0 \le i \le n$. Hybrid $\mathsf{Hyb}_{2,i}$ is identical to Hyb_2, except for all $\ell \le i$ and $j \in [n]$, $r_{\ell,j} \xleftarrow{\text{R}} \mathcal{R}_\lambda$. For all $\ell > i$ and $j \in [n]$, $r_{\ell,j} = \mathsf{PRF}(k_\ell \diamond k'_j, x_j)$. By construction, $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_{2,0}$ and $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_{2,n}$. We now show that if PRF is $\Phi_\diamond$-RKA-secure, then $\mathsf{Hyb}_{2,i}$ is computationally indistinguishable from $\mathsf{Hyb}_{2,i+1}$ for all $0 \le i < n$.

Let $\mathcal{A}$ be a distinguisher for $\mathsf{Hyb}_{2,i}$ and $\mathsf{Hyb}_{2,i+1}$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ the distinguishes the real and ideal distributions in the $\Phi_\diamond$-RKA security game. In the $\Phi_\diamond$-RKA security game, $\mathcal{B}$ is given access to an oracle $\mathcal{O}'$. Adversary $\mathcal{B}$ operates as follows:

1. For all $\ell > i+1$, choose $k_\ell \xleftarrow{\text{R}} \mathcal{K}_\lambda$. For all $j \in [n]$, choose $k'_j \xleftarrow{\text{R}} \mathcal{K}_\lambda$.
2. For all $\ell \le i$ and $j \in [n]$, choose $r_{i,j} \xleftarrow{\text{R}} \mathcal{R}_\lambda$. For all $\ell > i+1$ and $j \in [n]$, set $r_{i,j} \leftarrow \mathsf{PRF}(k_\ell \diamond k'_j, x_j)$.
3. Let $\phi_j : \mathcal{K}_\lambda \to \mathcal{K}_\lambda$ be the function $k \mapsto k \diamond k'_j$. For $j \in [n]$, set $r_{i+1,j} \leftarrow \mathcal{O}'(\phi_j, x_j)$.
4. Invoke $\mathcal{A}$ on the set $\{f_i(x_j; r_{i,j})\}_{i,j\in[n]}$. Output whatever $\mathcal{A}$ outputs.

In the real experiment, the oracle $\mathcal{O}' = \mathcal{O}(k,\cdot,\cdot)$ where $k \leftarrow \mathcal{K}_\lambda$. In this case, for all $j \in [n]$, $r_{i+1,j} = \mathsf{PRF}(\phi_j(k), x_j) = \mathsf{PRF}(k \diamond k'_j, x_j)$. Thus, $\mathcal{B}$ perfectly simulated $\mathsf{Hyb}_{2,i}$ for $\mathcal{A}$. In the ideal experiment, the oracle $\mathcal{O}' = G(\cdot,\cdot)$ where $G \xleftarrow{\text{R}} \mathsf{Funs}[\Phi_\diamond \times \mathcal{X}_\lambda, \mathcal{R}_\lambda]$. Since for all $j \in [n]$, the k'_j are drawn independently and randomly from $\mathcal{K}_\lambda$ and $n = \mathsf{poly}(\lambda)$, with probability $1 - \mathsf{negl}(\lambda)$, all of the k'_j are unique. This means that with the same overwhelming probability, all of the ϕ_j are also unique. Thus, we conclude that for all $j \in [n]$, $r_{i+1,j}$ is uniform in $\mathcal{R}_\lambda$, in which case $\mathcal{B}$ has perfectly simulated $\mathsf{Hyb}_{2,i+1}$ for $\mathcal{A}$. We conclude that if there exists a distinguisher for $\mathsf{Hyb}_{2,i}$ and $\mathsf{Hyb}_{2,i+1}$, there exists an adversary that breaks the the $\Phi_\diamond$-RKA security of PRF. □

Combining Lemmas C.1 through C.3, we conclude that rFE is a correct functional encryption scheme for randomized functionalities (Definition 3.1). $\square$