(Multi-Input) FE for Randomized Functionalities, Revisited

Pratish Datta[*] Jiaxin Guan[†] Alexis Korb[‡] Amit Sahai[§]

Abstract

Randomized functional encryption (rFE) generalizes functional encryption (FE) by incorporating randomized functionalities. Randomized multi-input functional encryption (rMIFE) extends rFE to accommodate multi-input randomized functionalities.

In this paper, we reassess the framework of rFE/rMIFE enhancing our understanding of this primitive and laying the groundwork for more secure and flexible constructions in this field. Specifically, we make three key contributions:

- ***New definition:*** We identify critical gap in the existing indistinguishability-based (IND) security definition for rFE/rMIFE. Notably, current definition fails to adequately address security against malicious encryptors—a crucial requirement for rFE/rMIFE since their introduction. We propose a novel, robust IND security definition that not only addresses threats from malicious decryptors but also quantifies the security against malicious encryptors effectively.
- ***Counterexample:*** To illustrate the importance of this definitional gap, we provide a counterexample of an insecure rFE scheme that meets IND security under the previous definition but explicitly fails in a natural setting (and where this failure would be precluded by our enhanced definition). Our counterexample scheme is non-trivial and meticulously designed using standard cryptographic tools, namely FE for deterministic functions, pseudorandom function (PRF), public key encryption (PKE), and simulation-sound non-interactive zero-knowledge (NIZK) proof systems.
- ***Adaptive unbounded-message secure construction:*** The only viable prior construction of rMIFE by Goldwasser et al. [EUROCRYPT 2014] (which uses indistinguishability obfuscation ($i\mathcal{O}$) and other standard assumptions) has significant limitations: it permits only a pre-defined number of messages per encryption slot and operates under selective-security constraints, requiring adversaries to declare challenge ciphertext queries and "corrupted" encryption keys in advance. We address these shortcomings by employing sub-exponentially secure $i\mathcal{O}$. Technically, we build on and adapt methods developed by Goyal et al.[ASIACRYPT 2016] for deterministic MIFE.

Keywords: Functional encryption, randomized functionalities, multi-input, simulation-based security, indistinguishability-based security

[*]NTT Research. Email: `pratish.datta@ntt-research.com`. ORCID: 0000-0002-3938-7594.
[†]New York University. Email: `jiaxin@guan.io`. ORCID: 0000-0003-1823-8845.
[‡]UCLA. Email: `alexiskorb@cs.ucla.edu`. ORCID: 0000-0001-6888-5296.
[§]UCLA. Email: `sahai@cs.ucla.edu`. ORCID: 0000-0003-2216-9600.

Contents

1 Introduction

Functional Encryption. Functional Encryption (FE) [BSW11, O'N10] enhances the traditional public-key encryption paradigm by enabling fine-grained access control over encrypted data. In an FE scheme, a central authority holds a master secret key and issues a corresponding master public key. This authority uses the master secret key to generate secret keys for various legitimate functions, while any party can encrypt data using the master public key. When provided with a secret key for a function f and a ciphertext of a message x, decryption reveals $f(x)$ without disclosing any additional information about x.

Since its introduction, a key focus of FE research has been to explore the functionalities that can be supported and the underlying cryptographic assumptions. A significant body of exciting work [BF01, BB04, BGW05, SW05, GPSW06, KSW08, Wat09, LOS+10, ABB10b, ABB10a, LW10, OT10, OT12, LW12, Wat12, GVW12, GVW13, BGG+14, Att14, Wee14, GVW15, ABDP15, ALS15, CGW15, LV16, AS16, DDM16, BCFG17, BBL17, GKW17, Agr17, Wee17, DOT18a, CLT18, CGKW18, TT18, GWW19, AV19, LL20a, LL20b, GW20, ACGU20, ALMT20, AY20, KW20, Wee20, AGW20, Gay20, Wee21, AMVY21, Wee22, KNT21, DP21, DPT22, AKM+22, GGLW22, Tom23, HLL23, CW23, Wee24, HLL24, AKY24] has investigated these questions in the context of deterministic functionalities. This culminated in the work of Jain, Lin, and Sahai [JLS21, JLS22], who constructed FE schemes for general polynomial-size deterministic circuits based on well-established cryptographic assumptions. However, many real-world applications naturally involve randomized functionalities, posing challenges since existing FE schemes for deterministic functionalities cannot be directly extended to accommodate them.

FE for randomized functionalities. To address these challenges, Alwen et al. [ABF+13] and Goyal et al. [GJKS15] initiated the study of FE for randomized functionalities (rFE). In an rFE scheme, secret keys correspond to randomized functions f, allowing decryption of a ciphertext containing message x to reveal only a single sample from the output distribution of $f(x)$. Additionally, for a collection of secret keys for functions $f_1, \ldots, f_q$ and ciphertexts for messages $x_1, \ldots, x_n$, each secret key-ciphertext pair should yield independent samples from the distribution of $f_i(x_j)$ without revealing further information.

rFE shows considerable promise in both practical and theoretical applications, including privacy-aware auditing, differentially-private data release [GJKS15], delegation of encrypted contents [AW17], fully homomorphic encryption [ABF+13], controlled homomorphic encryption [DIPV17], and even FE for deterministic functionalities [GGHZ16]. Acknowledging the vast potential of rFE, numerous studies [ABF+13, GJKS15, GGHZ16, ITZ16, AW17, KSY15, DIPV17, LZ20, BKMT21] have explored rFE from both definitional and construction perspectives.

Security against both Malicious Decryptors and Encryptors for rFE. Unlike deterministic FE, the notion of rFE must not only ensure that decryptors cannot gain any additional information about the encrypted data beyond the intended output, but also it is essential for rFE to prevent malicious encryptors from generating "bad" ciphertexts for a message x, which, when decrypted using secret keys from certain functions f, result in distributions that significantly diverge from those of $f(x)$.

To illustrate this, consider the scenario of privacy-aware auditing as described in [GJKS15]. A government agency is tasked with overseeing financial institutions to ensure compliance with federal regulations. However, these institutions are hesitant to grant full access to their confidential data to external auditors. Partial access poses its own risks, as institutions could selectively disclose information, potentially compromising the integrity of the audit process.

rFE offers an effective solution. Financial institutions can encrypt their databases using rFE, while the government agency provides auditors with an rFE secret key, allowing them to randomly sample a small, unbiased subset of records from each database. It is crucial that when an auditor receives two distinct keys for the same encrypted database, each key generates independent samples. Similarly, if the same key is applied to two different databases, the auditor should still obtain independent samples from each. Additionally, if malicious institutions can craft faulty ciphertexts that lead to biased or correlated samples, they could undermine the entire audit process and jeopardize its integrity.

For a more in-depth discussion on the importance of addressing both malicious decryptors and encryptors in rFE, please refer to [GJKS15].

Simulation-based (SIM) Security Definition for rFE. The above two intuitive security requirements for rFE have been formalized through a unified simulation-based approach, first introduced in [GJKS15] and later refined in [AW17].[1] Similar to functional encryption (FE) for deterministic functionalities [O'N10, BSW11, AGVW13], the SIM security experiment for rFE defines an adversary that attempts to distinguish between interactions in the real world (where ciphertexts and secret keys are generated according to the rFE scheme) and an ideal world (where these elements are produced by a simulator with only minimal information). To address security against malicious encryptors, [GJKS15, AW17] introduced a decryption oracle into the security game, akin to the framework for IND-CCA2 security [RS92]. This oracle takes a ciphertext ct and a function f as input. In the real world, the challenger extracts the secret key for f and decrypts ct using it. In contrast, in the ideal world, the challenger uses a simulator that outputs either a value x or a special symbol $\perp$. The challenger then responds with a random value drawn from the distribution of $f(x)$ or with $\perp$, based on the simulator's output.

The work in [AW17] further enhanced the decryption oracle by allowing it to handle a set of polynomially many ciphertexts at the time along with a function f. In the real world, the challenger generates a single secret key for f and decrypts each ciphertext using this key. In the ideal world, the simulator is given the set of ciphertexts and can query an evaluation oracle once per ciphertext. For each query x, the oracle provides a fresh evaluation of $f(x)$. This refinement addresses the limitation of the original definition in [GJKS15], which, while preventing the adversary from creating individual ciphertexts that decrypt incorrectly, allowed malicious encryptors to produce sets of ciphertexts that yield correlated outputs when decrypted with the same key.

Indistinguishability-based (IND) security for rFE. While simulation-based (SIM) security represents the strongest form of security for rFE [O'N10, BSW11, AGVW13], it has a significant limitation: it can only handle a bounded number of ciphertext and secret key queries before the ciphertext queries [O'N10, BSW11, AGVW13, DIJ+13, DI13]. To address this, [GJKS15] introduced an indistinguishability-based (IND) security formulation for rFE, generalizing the counterpart framework for deterministic FE [O'N10, BSW11]. The goal, as with SIM security, is to protect against both malicious decryptors and encryptors.

More specifically, the IND security experiment for rFE involves an adversary attempting to distinguish between the encryptions of two messages, given access to secret keys for randomized functions whose output distributions, when evaluated on those messages, are statistically close.[2]

[1] Notably, not all prior works on rFE have considered security against malicious encryptors, as this is not necessary for certain applications of rFE [ABF+13, GGHZ16, ITZ16, KSY15].

[2] Computationally close output distributions can also be supported, but only if all function queries occur before the master public key is issued. Otherwise, the IND security definition becomes vacuous (for more details, see Remark 2.8 in [GJKS15]).

To account for malicious encryptors, the IND security framework in [GJKS15] again provides the adversary with a decryption oracle. This oracle accepts a ciphertext ct and a function f as input, generates a secret key for f, and decrypts ct using this key—much like in the real-world setting of the SIM security definition.

One of the key distinctions of IND security is that it imposes no inherent bounds on the number of ciphertext or secret key queries—whether they occur before or after the ciphertext queries. Furthermore, while SIM security for a bounded number of ciphertext queries implies IND security for arbitrarily many queries, this does not extend to the number of supported key queries. In fact, the transformation from IND security to SIM security [O'N10,BSW11,GJKS15] preserves the bound on the number of secret key queries.

Thus, for applications of rFE that require support for arbitrarily many ciphertext and secret key queries, (possibly) both before and after ciphertext queries, the best security one can achieve is IND security. Additionally, as has been observed in case of FE for deterministic functionalities, IND security can often be realized under weaker cryptographic assumptions compared to those needed for SIM security.

Limitations of existing IND security definition for rFE. The purpose of providing a decryption oracle to the adversary is to capture security against malicious encryptors. In the case of the SIM security definition, as established in [AW17,GJKS15], the use of a decryption oracle effectively achieves this goal.This is because, in the ideal world, the decryption oracle is honestly simulated by producing uniform samples from the output distribution of the queried function applied to the message encrypted within the queried ciphertext. Thus, the indistinguishability of the decryption oracle's output in the real world from that in the ideal world ensures that the adversary cannot manipulate decryption results in the real-world to be non-uniform or correlated.

However, this ideal functionality does not exist in the context of IND security. As a result, there is no guarantee that an adversary cannot submit maliciously crafted ciphertexts to the decryption oracle and influence its output. Therefore, simply providing a decryption oracle in the IND security experiment-otherwise designed to target malicious decryptors-fails to effectively capture security against malicious encryptors.

This leads us to ask the following critical questions:

Open Problem 1. *Can we formulate an* IND*-based security definition for* rFE *that properly captures security against both malicious decryptors and encryptors? Moreover, is it possible to construct an* rFE *scheme that achieves this enhanced* IND *security notion?*

Multi-Input Functional Encryption. Multi-Input Functional Encryption (MIFE), introduced by Goldwasser et al. [GGG^{+}14], extends the concept of FE to accommodate multi-input functionalities. In this framework, a secret key corresponding to an n-input function (where $n > 1$ and can be polynomial in the security parameter) allows a user with n ciphertexts—each encrypting a message $x_1, \ldots, x_n$—to compute the output $f(x_1, \ldots, x_n)$ by jointly decrypting the ciphertexts using the secret key, without revealing any additional information about the individual messages x_i.

MIFE is highly relevant due to its wide range of applications that involve extracting aggregate information from multiple data sources. These applications include executing SQL queries on encrypted databases, performing computations over encrypted data streams, non-interactive differentially-private data release, order-revealing encryption, and multi-client delegation of computation.

Given the strong potential of MIFE, there has been significant effort within the cryptographic community to construct MIFE schemes and their variants, both for general multi-input functionalities [BGJS15, AJ15, GJO16, BKS16, BGJS16, Lin17, CMR17, KS17, AJS18, KNT18] and for specific, practically important classes of functionalities [BLR+15, CLWW16, BZ16, LW16, AGRW17, KLM+18, BJK+18, CDG+18a, CDG+18b, DOT18b, ACF+18, ABKW19, ABG19, LT19, Tom19, AGW20, ACGU20, AGW20, ACF+20, ABM+20, CDSG+20, AFS21, AGT21, AGT22, AYY22, NPP22, ATY23, FFMV23, SV23, NPP23a, NPP23b, ARYY23, FFMV24]. However, all existing constructions have been limited to handling deterministic functionalities only.

MIFE for randomized functionalities. Similar to single-input FE, many of the application scenarios for MIFE mentioned above often require the ability to handle randomized functionalities. Examples include privacy-preserving joint audits across multiple organizations, salary surveys of citizens within a country, randomized SQL queries, and so on. For addressing these applications, it is possible to extend the syntax and security definitions of deterministic MIFE to randomized MIFE (rMIFE), in the same way as in the single-input case. In fact, Goldwasser et al. [GGG+14] introduced a SIM-based security definition for rMIFE by generalizing the SIM security definition for single-input rFE [GJKS15]. This definition can be naturally enhanced to encompass decryption of correlated ciphertexts along the line of [AW17]. On the other hand, while [GGG+14] did not specifically address an IND-based security definition for rMIFE, one can similarly extend the IND security definition from [GJKS15] to the context of rMIFE. However, as in the single-input case, such an IND security definition would fail to adequately capture security against malicious encryptors. Therefore, a robust IND security definition that effectively addresses security against malicious encryptors is necessary for rMIFE as well.

Importance of IND security for rMIFE. IND security is even more significant in the context of MIFE/rMIFE. This is because Goldwasser et al. [GGG+14] demonstrated that achieving SIM security for MIFE is impossible for general deterministic functionalities, even in the secret key setting, with constant arity and at most two ciphertext queries per encryption slot. They showed that such an MIFE scheme would imply virtual black-box (VBB) obfuscation, which is known to be impossible for general functionalities [BGI+01]. Since every deterministic functionality can be viewed as a randomized functionality with a singleton output distribution, the impossibility result of [GGG+14] extends directly to SIM security for rMIFE. Consequently, it is clear that the only achievable security in any meaningful general scenario for rMIFE is IND security.

Existing rMIFE scheme and its limitations. The only known rMIFE construction is the one briefly outlined by Goldwasser et al. [GGG+14] as an extension of their deterministic MIFE scheme, based on differing-inputs obfuscation ($di\mathcal{O}$) [BGI+01, BCP14]. However, $di\mathcal{O}$ is widely regarded as implausible in general [GGHW14], rendering their proposed MIFE and rMIFE constructions generally infeasible. Moreover, Goldwasser et al. [GGG+14] did not provide any formal security proof or even a proof sketch for their proposed rMIFE construction.

In the same work, Goldwasser et al. [GGG+14] also suggested that their deterministic MIFE construction based on indistinguishability obfuscation ($i\mathcal{O}$) [BGI+01, GGH+13, JLS21, JLS22] could be generalized to an rMIFE scheme as well in a manner similar to their $di\mathcal{O}$-based construction, though no concrete construction or proof sketch was provided. The original $i\mathcal{O}$-based deterministic MIFE construction was proven to achieve IND security, and thus, the rMIFE construction extended from it is expected to achieve IND security according to the multi-input extension of the definition in [GJKS15]. However, since this definition does not fully capture security against malicious encryptors, it is unclear whether that generalized rMIFE construction would provide such guarantees.

Additionally, the rMIFE construction inherits other significant limitations from the original deterministic MIFE scheme. These include security guarantees only against a bounded number of ciphertext queries per encryption slot and selective adversaries, who must declare all challenge ciphertext queries and the encryption keys they wish to corrupt at the beginning of the security experiment. These challenges lead to the following natural question.

Open Problem 2. *Can we construct a viable* rMIFE *scheme that achieves the following security properties:*

- IND *security under an enhanced definition that effectively captures security against both malicious decryptors and encryptors.*
- *Protection against fully adaptive adversaries, who are allowed to make ciphertext, secret key, and corruption queries in any arbitrary order, at any point during the security experiment.*
- *Support for an unbounded polynomial number of secret key and ciphertext queries per encryption slot.*

Our Results

We address the above open problems in affirmative. More precisely, our contribution is threefold.

1. ***New Robust* IND *Security Definition for* rFE/rMIFE*:***

   We introduce an enhanced IND security definition for rFE, capturing security against both malicious decryptors and encryptors effectively. Instead of using a unified security experiment as in [GJKS15], we separate the two security requirements into distinct experiments. The first experiment, capturing security against malicious decryptors, resembles the existing IND security game for deterministic FE/MIFE [BSW11,O'N10,GGG$^+$14]. Here, the adversary must distinguish between encryptions of two messages, given secret keys for randomized functions, where the output distributions of the functions applied to the messages are close to each other.

   The second IND-based security experiment addresses malicious encryptors. In this experiment, the adversary is given access to a KeyStore oracle, which stores the functions, the adversary wishes to request decryption under, together with their corresponding secret keys one for each function. The adversary also interacts with a decryption oracle that operates in two modes. In the first mode, the oracle decrypts the submitted ciphertext using the secret keys stored in KeyStore. In the second mode, the oracle extracts the plaintext by querying a message extraction oracle and returns random samples from the output distributions of the functions in KeyStore, applied to that message. The adversary's goal is to distinguish between these two modes. Indistinguishability ensures that the decryption result of the rFE scheme is close to an independent random sample from the randomized function's output distribution, guaranteeing strong security. For more details, refer to 2.1.

   This IND security definition naturally extends to the multi-input setting. In fact, we formally define our IND security notion for rMIFE, with the single-input rFE definitions as a special case.

   By the way, we also observe a relatively minor gap in the existing SIM security definitions for rFE/rMIFE [GGG$^+$14, GJKS15, AW17]. Roughly speaking, existing SIM security definitions [GJKS15,AW17] guarantee security only in situation where even if the adversary repeatedly requests the decryption of the same or correlated ciphertexts under the same function, it receives independent random samples from the function's output distribution, applied to the underlying plaintexts each time.

However, in practical scenarios, a single secret key for a function is often used repeatedly to decrypt ciphertexts over time. For example, in the privacy-aware auditing application described earlier, an auditor may use the same secret key to decrypt encrypted databases from multiple financial institutions during a single audit. In this case, one institution could observe the audit results of others and exploit that information by manipulating its own database encryption to influence the audit outcome. Existing SIM security definitions [GJKS15, AW17] do not account for this type of attack, where malicious encryptors could exploit repeated use of the same decryption key. We address this issue by proposing a more advanced SIM security framework that ensures resistance against such attacks. For more details, see section 2.1.2.

2. ***Counterexample Demonstrating the Gap Between New and Existing* IND *Security Definitions:*** We present a counterexample to highlight the significance of the gap between our IND security definition and the existing definition [GJKS15]. Specifically, we construct an insecure rFE scheme that satisfies IND security criteria of the previous definition but fails to do so in our new definition. The counterexample relies solely on standard cryptographic primitives, including functional encryption (FE) for deterministic functionalities,pseudorandom function (PRF),standard public key encryption (PKE), and simulation-sound non-interactive zero-knowledge (NIZK) proof systems [Sah99]. Our construction is non-trivial, and its security analysis requires careful consideration. We elaborate more on this in the 2.2 section below.

3. ***Adaptively* IND *secure* rMIFE *scheme with unbounded message security:*** Our final contribution is an adaptively IND secure rMIFE scheme for general randomized functionalities, based on sub-exponentially secure $i\mathcal{O}$. As demonstrated by Goldwasser et al. [GGG+14], $i\mathcal{O}$ is already implied by MIFE for general deterministic functionalities, even in the secret key setting, only supporting selective IND security with a single secret key query and at most two ciphertext queries per encryption slot.

   We analyze the IND security of our construction within the new robust security model introduced in this work. Moreover, our scheme is the first plausible construction that supports an unbounded polynomial number of secret key and challenge ciphertext queries per encryption slot. The construction relies on three key components: (a) sub-exponentially secure $i\mathcal{O}$, (b) sub-exponentially secure injective one-way functions, and (c) standard public-key encryption (PKE). Here, "sub-exponential security" means that the advantage of any efficient adversary is sub-exponentially small. For the injective one-way functions, this security should additionally hold against adversaries operating in sub-exponential time.

   A few clarifications are necessary regarding these primitives. First, the required level of security varies based on the function's arity, but it does not depend on the number of challenge messages. As Goldwasser et al. noted, the selective security of their deterministic MIFE scheme based on $i\mathcal{O}$ (though not bounded-message security, which pertains to their use of statistically sound non-interactive proofs) can be overcome via standard complexity leveraging. This should similarly apply to their rMIFE construction generalized from it. However, in their case, the required security level depends on the number of challenge messages, which leads to significantly larger parameters than our scheme, especially since the number of challenge messages is typically much larger than the function's arity in practical applications.

   Secondly, although our security proof utilizes a sub-exponentially secure injective one-way function (primitive (b)), this function is not needed in the scheme itself. Therefore, the existence of such an injective one-way function is sufficient for the security of our rMIFE scheme, even without the knowledge of an explicit candidate. At a technical level, we build on the methods developed by Goyal et al. [GJO16], achieving similar results for deterministic MIFE. We give an overview of our construction in 2.3

One caveat in our adaptive rMIFE construction is that we require the output distributions of the queried functions on the queried sets of inputs to be computationally indistinguishable with a sub-exponentially small distinguishing advantage. Overcoming the sub-exponential barrier remains an interesting open problem.

Challenges in Utilizing $i\mathcal{O}$. $i\mathcal{O}$ is an exceptionally powerful cryptographic tool, yet harnessing it to develop new primitives—even when ideal obfuscation would make the task straightforward—remains highly challenging. A striking example is the long-standing effort to construct adaptively secure SNARGs for NP from $i\mathcal{O}$. Despite extensive research, this milestone was only recently reached [WW24a, WW24b], underscoring the deep complexity of using $i\mathcal{O}$ to build cryptographic primitives, even in well-established areas.

In contrast, some transformations that seem feasible under ideal obfuscation turn out to be fundamentally unattainable with $i\mathcal{O}$ alone. A notable case is the "Upgrade any PKE to FE" problem [BKSW18], which, despite being simple to achieve using ideal obfuscation, has been proven impossible using $i\mathcal{O}$ as the sole tool [BKSW18].

In fact, after more than a decade of study, many fundamental questions about $i\mathcal{O}$ remain open. One major hurdle is constructing $i\mathcal{O}$ for Turing machines capable of handling inputs of arbitrary length—an unsolved problem that continues to challenge researchers. Expanding our understanding of both the potential and the limitations of $i\mathcal{O}$ is a crucial pursuit in cryptographic foundations. Our results and techniques contribute to this long-standing effort, pushing the field closer to resolving these enduring challenges.

2 Technical Overview

This section provides a high-level overview of our three key technical contributions as outlined in Section 1.

2.1 New Security Definitions for rFE/rMIFE

We introduce a novel IND security definition of rFE/rMIFE. We also present a robust SIM security formulation for rFE/rMIFE. In this technical overview, we focus on single-input functionality for the ease of exposition. However, the formal definition in Section 4 extends to multi-input functionalities. Unlike prior works [GJKS15, GGG$^+$14], our definitions are designed for adaptive adversaries, who are not required to submit their challenge ciphertext queries upfront.

2.1.1 New IND Security Definition for rFE.

As highlighted earlier, we address the IND security of rFE through two distinct indistinguishability-based security experiments, which are formalized below.

IND Security Against Malicious Decryptors. This security experiment extends the IND security framework for deterministic FE [O'N10, BSW11] to randomized functionalities, closely following the definition in [GJKS15] (Definition 2.6), with one key distinction: we omit providing the adversary with a decryption oracle, as this experiment is not concerned with malicious encryptors.

- **Setup**: The challenger runs the Setup algorithm to generate the master public/secret key pair $(\mathsf{mpk}, \mathsf{msk})$ and provides mpk to the adversary.

- **Query Phase 1**: The adversary can adaptively request any polynomial number of secret keys for randomized functions within the function space. For each query function f, the challenger runs KeyGen with the master secret key to generate a secret key sk_f and hands it to the adversary.

- **Challenge**: The adversary submits two challenge messages x_0 and x_1. The challenger selects a random bit $b \leftarrow \{0,1\}$ and encrypts x_b under mpk to generate the ciphertext ct, which is then sent to the adversary.

- **Query Phase 2**: The adversary can continue to adaptively request additional secret keys as in **Query Phase 1**, and the challenger responds accordingly.

- **Guess**: The adversary outputs its guess $b' \in \{0,1\}$ and wins if $b' = b$.

The adversary must satisfy an admissibility condition: for any queried function f and any pair of challenge messages x_0, x_1, the output distributions $f(x_0)$ and $f(x_1)$ must be indistinguishable. This definition can readily be generalized to handle multiple challenge ciphertexts. In fact, it can be shown that security against single and multiple ciphertexts are essentially equivalent [GJKS15].

IND **Security Against Malicious Encryptors.** This security experiment models the behavior of malicious encryptors attempting to influence functional outputs by generating faulty ciphertexts. The experiment proceeds as follows:

- **Setup**: The challenger runs the Setup algorithm to generate the master public/secret key pair $(\mathsf{mpk}, \mathsf{msk})$ and provides mpk to the adversary. A random bit $b \leftarrow \{0,1\}$ is also sampled.

- **Query Phase 1**: The adversary can adaptively make the following queries:
  - *Secret Key Query*: The adversary requests secret keys for any number of randomized functions from the underlying function family. For each function f, the challenger generates the secret key sk_f by running the KeyGen algorithm and provides it to the adversary.
  - *Secret Key Store Query*: The adversary requests the challenger to store secret keys for certain randomized functions. For each function g, the challenger generates sk_g using KeyGen algorithm and stores (g, sk_g) in a register KeyReg.
  - *Decryption Query*: The adversary submits ciphertexts for decryption. If $b = 0$, the challenger decrypts the ciphertext using all stored keys in KeyReg. If $b = 1$, the challenger extracts the plaintext using msk, applies all stored functions g to it, and returns independently sampled random outputs from the resulting distributions to the adversary. Each additionally stores these outputs along with the queried ciphertext in an output register OutReg, and if the adversary requests decryption for the same ciphertext once again, it simply outputs the store values.

- **Guess**: The adversary outputs a guess $b' \in \{0,1\}$ and wins if $b' = b$.

Old vs. New IND Definition. The IND security definition in [GJKS15] closely resembles our first definition, which addresses malicious decryptors, but with a key difference: their model includes a decryption oracle, similar to IND-CCA2 security. This oracle decrypts ciphertexts using honestly generated secret keys for the functions under which decryption is sought. While [GJKS15] claims this approach ensures security against malicious encryptors, it actually falls short. Observe that, unlike the SIM security model, the IND setting lacks an ideal functionality. The decryption oracle merely runs the decryption algorithm of the underlying rFE scheme, allowing adversaries to submit malicious ciphertexts and obtain biased or correlated outputs.

Our new IND security definition, specifically designed for malicious encryptors, addresses this issue by introducing two decryption modes for the decryption oracle: one for the real decryption algorithm and the other for an ideal decryption functionality, similar to the SIM security model. Indistinguishability between these modes ensures that adversaries cannot craft ciphertexts to influence the decryption oracle's outputs in ways that deviate from the intended functionality. Additionally, note that this is true even when the same secret keys are used by the decryption oracle repeatedly for decrypting the queried ciphertext over time.

2.1.2 New SIM Security Definition for rFE.

In the existing SIM security definitions [GJKS15, AW17], the decryption oracle generates a fresh secret key for a randomized function whenever decryption is requested under that function in the real world. In the ideal world, the oracle draws fresh uniform samples from the function's output distribution each time it is queried with that function. This means that existing SIM security definitions [GJKS15,AW17] guarantee security only in situation where even if the adversary repeatedly requests the decryption of the same or correlated ciphertexts under the same function, it receives independent random samples from the function's output distribution, applied to the underlying plaintexts each time.

To address this shortcoming, we also propose an advanced SIM security definition for rFE/rMIFE that not only addresses malicious encryptors' behavior, as covered by previous definitions [GJKS15, GGG+14, AW17], but also protects against the attack scenario involving repeated usage of secret keys. Roughly, this is achieved by modifying the decryption oracle in the SIM security model as follows. We introduce a new KeyStore oracle that stores all functions the adversary wishes to query to the decryption oracle. In the real world, the KeyStore oracle generates and stores a single secret key for each submitted function. When the adversary requests decryption, the decryption oracle uses the secret keys currently stored in KeyStore to decrypt the ciphertext and return the result.

In the ideal world, the decryption oracle maintains an output register for decryption results. When the adversary queries the decryption oracle with a ciphertext, it extracts the underlying plaintext and draws random samples from the output distribution of the functions currently stored in KeyStore, applied to that plaintext. The oracle then returns these samples to the adversary and stores them, along with the ciphertext, in the output register. If the adversary later queries the same ciphertext, the oracle returns the stored samples instead of drawing fresh ones. This effectively prevents the adversary from influencing future decryption outcomes by adaptively crafting ciphertexts based on previously observed decryption results, even when the same secret keys are repeatedly used.

2.2 Counterexample.

To highlight the shortcomings of the IND security definition in [GJKS15], we present a counterexample. Specifically, we construct an rFE scheme that satisfies the IND security requirements of [GJKS15] but is, in fact, insecure. We demonstrate that this scheme fails to meet the criteria of our proposed IND security definition. An overview of the rFE scheme is provided below.

The rFE Scheme. Our counterexample rFE scheme leverages the following cryptographic tools: (a) an FE scheme for general deterministic functionalities, (b) a pseudorandom function (PRF), (c) a public key encryption (PKE) scheme, (d) a symmetric key encryption (SKE) scheme, and (e) a simulation-sound non-interactive zero-knowledge (NIZK) proof system. The scheme operates as follows:

- **Setup**: The Setup algorithm runs the Setup for the FE, PKE, and NIZK systems, generating keys: $(\mathsf{FE.mpk}, \mathsf{FE.msk})$ for FE, $(\mathsf{PKE.pk}, \mathsf{PKE.sk})$ for PKE, $\mathsf{SKE.sk}$ for SKE and a common reference string crs for NIZK. The master public key is $\mathsf{mpk} = (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \mathsf{crs})$, and the master secret key is $\mathsf{msk} = \mathsf{FE.msk}$. The algorithm outputs mpk and retains msk.
- **Enc**: Given the message x, the encryption algorithm proceeds as follows:
  - It samples a PRF key K and generates an FE ciphertext $\mathsf{FE.ct}$ encrypting $(x, K, 0, \perp)$ under $\mathsf{FE.mpk}$ where $\perp$ is special string of appropriate length.
  - It then creates a PKE ciphertext $\mathsf{PKE.ct}$ encrypting $(x, K, 0, \perp)$ along with $\mathsf{FE.ct}$ under $\mathsf{PKE.pk}$.
  - Next, it computes an NIZK proof π attesting that both ciphertexts encrypt the same values $(x, K, \alpha, \widehat{\mathsf{sk}})$, and additionally $\mathsf{PKE.ct}$ also encrypts the FE ciphertext. Here, $\alpha \in \{0, 1\}$, $\widehat{\mathsf{sk}}$ is a string of size equal to the key length of the SKE.
  - Finally, the encryption outputs the ciphertext $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
- **KeyGen**: The algorithm takes the master secret key msk and a randomized function f and proceeds as follows.
  1. It samples a seed s and a SKE ciphertext $\mathsf{SKE.ct}$ encrypting a random message with the same length as the output of f to construct a function $G[f, s, \mathsf{SKE.ct}]$.
  2. The function G takes as input $(x, K, \alpha, \widehat{\mathsf{sk}})$ and operates in two modes depending on α:
     - If $\alpha = 0$, it computes randomness $\mathsf{PRF.Eval}(K, s)$ and uses it to compute f on x;
     - If $\alpha = 1$, it decrypts the SKE ciphertext using $\widehat{\mathsf{sk}}$ and output the corresponding message.
  3. The algorithm generates an FE secret key $\mathsf{FE.sk}_{G_f}$ for $G[f, s, \mathsf{SKE.ct}]$ using $\mathsf{FE.msk}$ and outputs SK_f.
- **Dec**: The Dec algorithm takes as input the NIZK common reference string crs, a secret key $\mathsf{SK}_f = \mathsf{FE.sk}_{G_f}$ for the randomized function f and a ciphertext $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$. It verifies the NIZK proof π. If valid, it decrypts $\mathsf{FE.ct}$ using the secret key $\mathsf{FE.sk}_{G_f}$ and outputs the result; otherwise, it aborts and outputs $\perp$.

It is easy to verify that the above rFE scheme is correct.

Insecurity of the constructed rFE scheme We demonstrate that the rFE scheme described above is insecure against malicious encryptors. Specifically, consider a pseudorandom function (PRF) with a key of the form $K = (K', 0, \perp)$, where K' is the key for another pseudorandom function PRF', and $\perp$ is a special symbol. Given an input seed s, this PRF outputs $\mathsf{PRF}'(K', s)$. It is easy to verify that this PRF behaves as a valid pseudorandom function. However, suppose the evaluation algorithm of this PRF includes a trojan branch. For keys of the form $K = (K', 1, r)$, where K' is the key for PRF' and r is a fixed string (matching the length of the output of PRF'), the evaluation algorithm bypasses PRF' entirely and directly outputs the string r.

Now, consider instantiating the above rFE scheme with this specially crafted PRF and its evaluation algorithm. If the encryptor generates two ciphertexts ct_1 and ct_2 for the same message x, using PRF keys $K_1 = (K'_1, 1, r)$ and $K_2 = (K'_2, 1, r)$ with freshly sampled K'_1 and K'_2 but using the same string r, the decryption using a secret key $\mathsf{FE.sk}_{G_f}$ for a randomized function f will yield the same output $f(x; r)$ in both cases. This occurs because the trojan evaluation branch outputs the fixed string r, regardless of the seed.

In a secure rFE scheme, we expect the decryption results to be independent, uniformly sampled outputs from the distribution of $f(x)$. However, by carefully selecting PRF keys, a malicious encryptor can introduce arbitrary correlations between the decryption outputs of different ciphertexts for the same message. This attack can be further extended to enforce correlations among ciphertexts encrypting different messages, breaking the intended security guarantees of the rFE scheme.

Proving that our rFE scheme achieves [GJKS15] IND security definition. We provide an intuitive outline of why the rFE scheme constructed above satisfies the IND security definition from [GJKS15]. Observe that, this security definition is similar to the one against malicious decryptors but with an additional decryption oracle, akin to IND-CCA2 security. For simplicity in this overview, let's ignore the decryption oracle and focus on the case where the ciphertext consists solely of the underlying FE ciphertext. The other two components of the ciphertext are mostly designed to handle the decryption oracle. More precisely, those components are used to make the decryption oracle not use any secret key for FE while answering the decryption queries of the adversary during the hybrid proof. For further details, see Section 5.

At a high level, our goal is to reduce the IND security of the rFE scheme to the security of the underlying FE scheme, the SKE scheme, and the pseudorandom function (PRF). We outline the key steps in the proof through a series of hybrid arguments:

1. *Initial Setup:* We start with the security game where the FE ciphertext encrypts $(x_0, K, 0, \perp)$.
2. *Hardcoding the Output*: We now change the SKE ciphertext in the KeyGen queries from encrypting a random message to encrypting the output of $f(x_0; r)$, where r is the PRF output used as the randomness.
3. *Modifying the Ciphertext:* We update the FE ciphertext to encrypt $(\perp, \perp, 1, \mathsf{SKE.sk})$. Since $\alpha = 1$, the function $G[f, s, \mathsf{SKE.ct}]$ will output the result of decrypting $\mathsf{SKE.ct}$ using $\mathsf{SKE.sk}$, which yields exactly $f(x_0; r)$. So the output of $G[f, s, \mathsf{SKE.ct}]$ remains unchanged, and thus this transition is indistinguishable by the security of FE scheme.
4. *Switching* PRF *Output*: Next, we use the security of the PRF to replace r with a uniform random string, instead of the output of the PRF. This transition is indistinguishable due to the security of the pseudorandom function PRF, as the PRF key K is hidden and unused.
5. *Switching to $f(x_1)$:* In the next hybrid, we switch $f(x_0; r)$ to $f(x_1; r)$, which is a uniform sample from the output distribution of $f(x_1)$. This change is indistinguishable due to the

indinstinguishability of $f(x_0)$ and $f(x_1)$, as required by the IND security game restriction for rFE.

6. *Final Reversion:* Once this transition is made, we can reverse the previous steps, ultimately encrypting $(x_1, K, 0, \perp)$, completing the proof.

2.3 The sketch of proposed rMIFE scheme.

We now outline the main technical ideas behind our adaptively secure rMIFE scheme, which supports an unbounded number of challenge ciphertext queries per encryption slot. Inspired by [GJO16], we build upon the adaptively secure deterministic MIFE construction of Goldwasser et al. [GGG+14], based on $di\mathcal{O}$.

In their construction, the encryption key for each index $i \in [n]$ (where n is the function's arity) consists of a pair of public keys $(\mathsf{pk}_i^0, \mathsf{pk}_i^1)$ from an underlying public key encryption (PKE) scheme. The ciphertext for index i includes encryptions of the plaintext under both pk_i^0 and pk_i^1, along with a simulation-sound NIZK proof ensuring that both ciphertexts encrypt the same message. Additionally, the ciphertext contains a one-time signature on the two PKE ciphertexts and the NIZK proof, using a fresh one-time verification key generated at encryption time.

The secret key for a function f is an obfuscated program that processes n ciphertext pairs, each with associated proofs, one-time signatures, and verification keys. This program has the function f and a key K for a puncturable pseudorandom function (PRF) [SW14, BGI14, BW13, KPTZ13] hardwired. The program takes as input ciphertext pairs $(c_1^0, c_1^1, \pi_1, \mathsf{vk}_1, \sigma_1), \ldots, (c_n^0, c_n^1, \pi_n, \mathsf{vk}_n, \sigma_n)$ and first verifies the one-time signatures and proofs. If all checks pass, the program decrypts each ciphertext using the corresponding secret key. It then evaluates the puncturable PRF with key K on the entire ciphertext tuple to generate randomness r, which is finally used to apply f to the decrypted plaintexts.

Goldwasser et al. [GGG+14] showed that security against dishonest decryptors follows similarly to their deterministic MIFE scheme based on $di\mathcal{O}$. They further mention that, security against malicious encryptors is ensured by the use of one-time signatures, which guarantee the uniqueness of each ciphertext, preventing adversaries from modifying honestly generated ciphertexts to manipulate decryption queries.

Unfortunately, as highlighted in our previous discussion, merely preventing the adversary from modifying honestly generated ciphertexts for decryption queries is insufficient to guarantee IND security against malicious encryptors. In fact, an adversary could generate "bad" ciphertexts for slots where it has corrupted the encryption keys, then combine these faulty ciphertexts with honestly generated ones from uncorrupted slots to extract non-uniform or correlated outputs from the decryption oracle. Therefore, a more robust approach is required to construct an rMIFE scheme that meets the stronger IND security definition. Furthermore, a key assumption behind security proof of the above rMIFE scheme by [GGG+14] is the use of $di\mathcal{O}$. Specifically, when function keys are switched to decrypt the second ciphertext in each pair, an adversary capable of detecting this change could exploit it to produce a false proof.

To address these issues , we introduce modifications to the scheme, allowing us to leverage a result from [BCP14], which shows that any indistinguishability obfuscator is, in fact, a $di\mathcal{O}$ for circuits that differ on polynomially many points. Fortunately, [CGJS15] recently demonstrated that the result of [BCP14] extends to our setting. Thus, we begin with a sub-exponentially secure indistinguishability obfuscator, which forms the basis of our enhanced approach to achieving adaptive IND security in the presence of malicious encryptors.

Specifically, to ensure that the proofs of well-formedness are unique for each ciphertext pair—and to limit the number of differing input points in the corresponding hybrids of our security proofs to a polynomial amount-we design novel proof strategy using $i\mathcal{O}$ and puncturable PRFs. Here's how it works:

We include in the public parameters an obfuscated program that takes two ciphertexts and a witness proving they are well-formed (i.e., generated using the same message and randomness). If this check succeeds, the program outputs a puncturable PRF evaluation on those ciphertexts. The secret key for a function f is then an obfuscated program that has hardwired the PRF keys and verifies the proofs of well-formedness by checking the correctness of the PRF evaluations. As in the construction by [GGG+14], the program also contains an additional puncturable PRF key K, which is sampled during key generation. Once all PRF evaluations are verified, this puncturable PRF with key K is applied to the entire collection of ciphertexts to generate the randomness needed for the functional output.

In the security proof, we introduce a key enhancement by performing the verification through an injective one-way function applied to the PRF values, rather than directly comparing the PRF outputs. This approach ensures that extracting a differing input at this stage of the proof corresponds to inverting the injective one-way function. Without this mechanism, we would need to hardcode the correct PRF evaluation into the obfuscated secret key, making it difficult to argue security effectively.

Security against malicious decryptors. We now sketch the sequence of hybrids in our IND security proof against malicious decryptors. The proof starts from a hybrid where each challenge ciphertext encrypts x_i^0 for $i \in [n]$. Then we switch to a hybrid where each c_i^1 is an encryption of x_i^1 instead. These two hybrids are indistinguishable due to security of the PKE scheme. Let s denote the length of a ciphertext. For each index $i \in [n]$ we define hybrids indexed by w, for all $w \in [2^{2sn}]$, in which function key SK_f decrypts the first ciphertext in the pair using SK_i^0 when $(c_1^0, c_1^1, \ldots, c_n^0, c_n^1) < w$ and decrypts the second ciphertext in the pair using SK_i^1 otherwise. Parse $w = (w_1^0, w_1^1, \ldots, w_n^0, w_n^1)$.

Hybrids indexed by w and $w + 1$ can be proven indistinguishable as follows: We first switch to sub-hybrids that puncture the PRF key at w_i^0, w_i^1, changes a function key SK_f to check correctness of an PRF value by applying an injective one-way function as described above, and hardcoded the output of the injective one-way function as the PRF evaluation at the punctured point. We also puncture the hardwired puncturable PRF key K used for generating output randomness as the point $(w_1^0, w_1^1, \ldots, w_n^0, w_n^1)$. Roughly speaking, if the two hybrids differ at an input of the form $(w_1^0, w_1^1, u_1, \ldots, w_n^0, w_n^1, u_n)$ where u_i is some fixed value (a PRF evaluation of (w_i^0, w_i^1)), extracting the differing input can be used to invert the injective one-way function on random input (namely the u_i). The formal security argument is a bit subtle at this point since unlike deterministic MIFE, we do not have exactly quality of the functional values corresponding to 0 and 1 cases. Instead, we carefully leverage the sub-exponentially small computational distance between the functional output distribution corresponding to the 0 and 1 cases. Please refer to our formal proof in the sequel. Finally, we note that exponentially many hybrids are indexed by all possible ciphertext vectors that could be input to decryption (i.e., vectors of length the arity of the functionality) and not all possible challenge ciphertext vectors. This allows us to handle any unbounded (polynomial) number of ciphertexts for every index. Also, by deterministically traversing over all possible ciphertexts, we are able to support adaptive adversary since the deduction does not need to know what challenge ciphertexts the adversary will be querying during the hardwiring at different stages of the security proof.

– We now outline the sequence of hybrids in our IND security proof against malicious decryptors.

- The proof begins with a hybrid where each challenge ciphertext encrypts x_i^0 for $i \in [n]$.
- We then switch to a hybrid where each c_i^1 encrypts x_i^1 instead. These two hybrids are indistinguishable due to the security of the underlying PKE scheme.
- Let s denote the length of a ciphertext. For each index $i \in [n]$, we define hybrids indexed by w, for all $w \in [2^{2sn}]$. In these hybrids, the function key SK_f decrypts the first ciphertext in each pair using SK_i^0 when $(c_1^0, c_1^1, \ldots, c_n^0, c_n^1) < w$, and decrypts the second ciphertext in each pair using SK_i^1 otherwise. The index w is parsed as $(w_1^0, w_1^1, \ldots, w_n^0, w_n^1)$.
- Hybrids indexed by w and $w+1$ can be proven indistinguishable as follows:
  - First, we introduce sub-hybrids that puncture the PRF key at the points w_i^0, w_i^1.
  - We then modify the function key SK_f to check the correctness of a PRF value by applying an injective one-way function, as described earlier, and hardcode the output of this function at the punctured points. Additionally, we puncture the hardcoded PRF key K, used for generating output randomness, at the point $(w_1^0, w_1^1, \ldots, w_n^0, w_n^1)$.
  - Roughly speaking, if the two hybrids differ on an input of the form $(w_1^0, w_1^1, u_1, \ldots, w_n^0, w_n^1, u_n)$ —where u_i is the result of a PRF evaluation on (w_i^0, w_i^1)—extracting the differing input would allow us to invert the injective one-way function on random input, i.e., the u_i. The formal security argument is subtle here, as unlike in deterministic MIFE, we do not have exact equality between functional values in the 0 and 1 cases. Instead, we carefully exploit the sub-exponentially small computational distance between the output distributions of the function in these two cases. For full details, please refer to our formal proof.

Lastly, we note that the exponentially many hybrids are indexed by all possible ciphertext vectors that could be input to decryption (i.e., vectors of length equal to the function's arity), rather than just the challenge ciphertext vectors. This allows us to handle any unbounded (polynomial) number of ciphertexts per index. By deterministically traversing all possible ciphertexts, we support adaptive adversaries without needing to know in advance which challenge ciphertexts the adversary will query at different stages of the security proof.

Security against malicious encryptors Finally, to argue security against malicious encryptors, we observe that the randomness used to evaluate the function output within the secret key program is derived from the hardwired PRF key, which is applied to the entire tuple of input ciphertexts. As a result, the encryptor has no control over the randomness used to generate the function output. This is because due to the pseudorandom properties of the PRF, whenever a different collection of ciphertexts is provided to the secret key program, the program generates a fresh random string for evaluating the output. This ensures that the encryptors cannot influence the randomness, maintaining the integrity of the function evaluation.

3 Preliminaries

Throughout, we will use λ to denote the security parameter.

Notation

- We say that a function $f(\lambda)$ is negligible in λ if $f(\lambda) = \lambda^{-\omega(1)}$, and we denote it by $f(\lambda) = \mathsf{negl}(\lambda)$.

- We say that a function $g(\lambda)$ is polynomial in λ if $g(\lambda) = p(\lambda)$ for some fixed polynomial p, and we denote it by $g(\lambda) = \mathsf{poly}(\lambda)$.
- For $n \in \mathbb{N}$, we use $[n]$ to denote $\{1, \ldots, n\}$.
- If R is a random variable, then $r \leftarrow R$ denotes sampling r from R. If T is a set, then $i \leftarrow T$ denotes sampling i uniformly at random from T.

Definition 3.1 (Statistical Distance). *Let D_1 and D_2 be two distributions with support in X. The statistical distance between D_1 and D_2 is*

$$\Delta(D_1, D_2) = \frac{1}{2} \sum_{x \in X} \Big| \Pr[D_1 = x] - \Pr[D_2 = x] \Big|$$

Notation Let A and B be two random variables with support in X. We use $\Delta(A, B)$ to denote the statistical distance $\Delta(P_A, P_B)$ between the underlying distributions of the random variables.

Definition 3.2 (Pseudorandom Function (PRF)). *A pseudorandom function family (PRF) with key space $\mathcal{K} = \{\mathcal{K}_{\lambda,n,m}\}_{\lambda,n,m \in \mathbb{N}}$ is a tuple of PPT algorithms $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$ where*

- $\mathsf{PRF.Setup}(1^\lambda, 1^n, 1^m)$ *is a randomized algorithm that takes as input the security parameter λ, an input length n, and an output length m, and outputs a key $K \in \mathcal{K}_{\lambda,n,m}$.*
- $\mathsf{PRF.Eval}(K, x)$ *is a deterministic algorithm that takes as input a key $K \in \mathcal{K}_{\lambda,n,m}$ and an input $x \in \{0,1\}^n$, and outputs a value $y \in \{0,1\}^m$.*

Security requires that there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and all PPT *adversaries $\mathcal{A}$,*

$$\Big| \Pr[\mathsf{Expt}^{\mathsf{PRF}}_{\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{PRF}}_{\mathcal{A}}(1^\lambda, 1) = 1] \Big| \le \mu(\lambda)$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, we define

$\mathsf{Expt}^{\mathsf{PRF}}_{\mathcal{A}}(1^\lambda, b)$

1. ***Parameters**: $\mathcal{A}$ takes as input 1^λ and outputs an input size 1^n and an output size 1^m.*
2. ***Setup**:*
   (a) *If $b = 0$, sample $K \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^n, 1^m)$.*
   (b) *If $b = 1$, sample $R \leftarrow \mathcal{R}_{n,m}$ where $\mathcal{R}_{n,m}$ is the set of all functions from $\{0,1\}^n$ to $\{0,1\}^m$.*
3. ***PRF Queries**: The following can be repeated any polynomial number of times:*
   (a) *$\mathcal{A}$ outputs a value $x \in \{0,1\}^n$.*
   (b) *If $b = 0$, send $y = \mathsf{PRF.Eval}(K, x)$ to $\mathcal{A}$.*
   (c) *If $b = 1$, send $y = R(x)$ to $\mathcal{A}$.*
4. ***Experiment Outcome**: $\mathcal{A}$ outputs a bit b' which is the output of the experiment.*

Definition 3.3 (Puncturable Pseudorandom Function (PPRF)). *A puncturable pseudorandom function family with key space $\mathcal{K} = \{\mathcal{K}_{\lambda,n,m}\}_{\lambda,n,m\in\mathbb{N}}$ is a tuple of PPT algorithms* $\mathsf{PPRF} = (\mathsf{PPRF.Setup}, \mathsf{PPRF.Eval}, \mathsf{PPRF.Punc}, \mathsf{PPRF.EvalPunc})$ *where*

- $\mathsf{PPRF.Setup}(1^\lambda, 1^n, 1^m)$ *is a randomized algorithm that takes as input the security parameter λ, an input length n, and an output length m, and outputs a key $K \in \mathcal{K}_{\lambda,n,m}$.*
- $\mathsf{PPRF.Eval}(K, x)$ *is a deterministic algorithm that takes as input a key $K \in \mathcal{K}_{\lambda,n,m}$ and an input $x \in \{0,1\}^n$, and outputs a value $y \in \{0,1\}^m$.*
- $\mathsf{PPRF.Punc}(K, x^*)$ *is a randomized algorithm that takes as input a key $K \in \mathcal{K}_{\lambda,n,m}$ and an input $x^* \in \{0,1\}^n$, and outputs a punctured key $K[x^*]$.*
- $\mathsf{PPRF.EvalPunc}(K[x^*], x)$ *is a deterministic algorithm that takes as input a punctured key $K[x^*]$ and an input $x \in \{0,1\}^n$, and outputs either a value $y \in \{0,1\}^m$ or $\perp$.*

We require correctness under puncturing, and selective pseudorandomness at punctured points.

Remark 3.4. For convenience, we will sometimes combine $\mathsf{PPRF.Eval}$ and $\mathsf{PPRF.EvalPunc}$ into one algorithm. This can be done by having the combined algorithm run $\mathsf{PPRF.Eval}$ if it receives a regular key K and run $\mathsf{PPRF.EvalPunc}$ if it receives a punctured key $K[x^*]$ since the two types of keys are easily distinguishable in the construction from [SW14]. When using the combined algorithm, we will overload notation and refer to it simply by $\mathsf{PPRF.Eval}$.

Definition 3.5 (Correctness under Puncturing). *For all $\lambda, n, m \in \mathbb{N}$ and all $x^* \in \{0,1\}^n$, if $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^n, 1^m)$ and $K[x^*] \leftarrow \mathsf{PPRF.Punc}(K, x^*)$, then*

$$\mathsf{PPRF.EvalPunc}(K[x^*], x) = \begin{cases} \mathsf{PPRF.Eval}(K, x) & \text{if } x \neq x^* \\ \perp & \text{else} \end{cases}$$

Definition 3.6 (Selective Pseudorandomness at Punctured Points). *There exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and all* PPT *adversaries $\mathcal{A}$,*

$$\left| \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PPRF}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PPRF}}(1^\lambda, 1) = 1] \right| \leq \mu(\lambda)$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, we define

$\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PPRF}}(1^\lambda, b)$

1. ***Parameters**: $\mathcal{A}$ takes as input 1^λ, and outputs an input size 1^n, an output size 1^m, and a message $x^* \in \{0,1\}^n$.*
2. ***Compute Values**:*
   (a) $K \leftarrow \mathsf{PPRF.Setup}(1^\lambda, 1^n, 1^m)$.
   (b) $K[x^*] \leftarrow \mathsf{PPRF.Punc}(K, x^*)$.
   (c) *If $b = 0$, send $(y, K[x^*])$ to $\mathcal{A}$ where $y = \mathsf{PPRF.Eval}(K, x^*)$.*
   (d) *If $b = 1$, send $(r, K[x^*])$ to $\mathcal{A}$ where $r \leftarrow \{0,1\}^m$.*
3. ***Experiment Outcome**: $\mathcal{A}$ outputs a bit b' which is the output of the experiment.*

Definition 3.7 (Symmetric Key Encryption (SKE)). *A symmetric key encryption scheme with key space $\mathcal{K} = \{\mathcal{K}_{\lambda,n}\}_{\lambda,n\in\mathbb{N}}$ and ciphertext size $m(\cdot)$ is a tuple of PPT algorithms* $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *where*

- $\mathsf{SKE.Setup}(1^\lambda, 1^n)$ *is a randomized algorithm that takes as input the security parameter λ and an input length n and outputs a secret key $k \in \mathcal{K}_{\lambda,n}$*
- $\mathsf{SKE.Enc}(k, x)$ *is a randomized algorithm that takes as input a secret key $k \in \mathcal{K}_{\lambda,n}$ and a message $x \in \{0,1\}^n$ and outputs an encryption $\mathsf{ct} \in \{0,1\}^{m(\lambda,n)}$ of x.*
- $\mathsf{SKE.Dec}(k, \mathsf{ct})$ *is a deterministic algorithm that takes as input a secret key $k \in \mathcal{K}_{\lambda,n}$ and a ciphertext $\mathsf{ct} \in \{0,1\}^{m(\lambda,n)}$ and outputs a value $y \in \{0,1\}^n$.*

Correctness requires that for all $\lambda, n \in \mathbb{N}$ and every $x \in \{0,1\}^n$,

$$\Pr\left[\mathsf{SKE.Dec}(k, \mathsf{SKE.Enc}(k, x)) = x : k \leftarrow \mathsf{SKE.Setup}(1^\lambda, 1^n)\right] = 1$$

Security requires that there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and all PPT *adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathsf{Expt}^{\mathsf{SKE}}_{\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{SKE}}_{\mathcal{A}}(1^\lambda, 1) = 1]\right| \le \mu(\lambda)$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, we define

$\mathsf{Expt}^{\mathsf{SKE}}_{\mathcal{A}}(1^\lambda, b)$

1. ***Parameters**: $\mathcal{A}$ takes as input 1^λ and outputs an input size 1^n.*
2. ***Setup**: $k \leftarrow \mathsf{SKE.Setup}(1^\lambda, 1^n)$*
3. ***Challenge Message Queries**: The following can be repeated any polynomial number of times:*
   (a) *$\mathcal{A}$ outputs a challenge message pair (x_0, x_1) where $x_0, x_1 \in \{0,1\}^n$.*
   (b) $\mathsf{ct}_b \leftarrow \mathsf{SKE.Enc}(k, x_b)$
   (c) *Sent ct_b to $\mathcal{A}$.*
4. ***Experiment Outcome**: $\mathcal{A}$ outputs a bit b' which is the output of the experiment.*

Definition 3.8 (Public Key Encryption (PKE)). *A public-key encryption (PKE) scheme is a tuple of PPT algorithms* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *with the following syntax:*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$*: Takes as input the security parameter and samples a public/private key pair.*
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$*: Takes as input the public key and a message, and outputs a ciphertext encrypting the corresponding message.*
- $m \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$*: Takes as input the private key and a ciphertext, and outputs a decrypted message.*

Correctness requires that for all $\lambda \in \mathbb{N}$ and every m,

$$\Pr\left[\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{PKE.Enc}(\mathsf{pk}, m)) = m : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)\right] = 1.$$

A PKE scheme PKE *is IND-CPA secure if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we have*

$$\Pr\left[b' = b : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda); \\ (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pk}); \\ b \leftarrow \{0,1\}, \mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b); \\ b' \leftarrow \mathcal{A}_2(\mathsf{st}, \mathsf{ct}^*) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

3.1 Indistinguishability Obfuscation

The recent work of [JLS22] shows how to construct $i\mathcal{O}$ for P/Poly from well-established computational assumptions.

Definition 3.9 (Indistinguishability Obfuscation ($i\mathcal{O}$) for Circuits [JLS21])**.** *A uniform PPT algorithm $i\mathcal{O}$ is an indistinguishability obfuscator for polynomial-sized circuits if the following holds:*

- ***Completeness**: For every $\lambda \in \mathbb{N}$, every circuit C with input length n, and every input $x \in \{0,1\}^n$,*
$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(1^\lambda, C)] = 1$$
- ***Indistinguishability**: For every two ensembles $\{C_{0,\lambda}\}, \{C_{1,\lambda}\}$ of polynomial-sized circuits that have the same size, input length, and output length, and are functionally equivalent, that is, $\forall \lambda \in \mathbb{N}$, $C_{0,\lambda}(x) = C_{1,\lambda}(x)$ for every input x, then for all polynomial-time, non-uniform adversaries $\mathcal{A}$, there exists a negligible function μ, such that for all λ,*
$$\left|\Pr[\mathcal{A}(1^\lambda, i\mathcal{O}(1^\lambda, C_{0,\lambda}))] = 1 - \Pr[\mathcal{A}(1^\lambda, i\mathcal{O}(1^\lambda, C_{1,\lambda}))] = 1\right| \leq \mu(\lambda)$$

3.2 Functional Encryption

Here we give some fundamental definitions for functional encryption (FE) schemes. First, we define a class of functions parameterized by function size, input length, and output length.

Definition 3.10 (Function Class)**.** *The function class $\mathcal{F}[n, \ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{R}, \ell_\mathcal{Y}]$ is the set of all functions f that have a description $\widehat{f} \in \{0,1\}^{\ell_\mathcal{F}}$, take inputs in $\{0,1\}^{\ell_\mathcal{X}}$, and output values in $\{0,1\}^{\ell_\mathcal{Y}}$.*

Definition 3.11 (Public-Key Functional Encryption)**.** *A public-key functional encryption scheme for* P/Poly *is a tuple of PPT algorithms* FE = (Setup, Enc, KeyGen, Dec) *defined as follows:*[3]

- $\mathsf{Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$*: takes the security parameter λ, a function size $\ell_\mathcal{F}$, an input size $\ell_\mathcal{X}$, and an output size $\ell_\mathcal{Y}$, and outputs the master public key* mpk *and the master secret key* msk.
- $\mathsf{Enc}(\mathsf{mpk}, x)$*: takes as input the master public key* mpk *and a message $x \in \{0,1\}^{\ell_\mathcal{X}}$, and outputs an encryption* ct *of x.*
- $\mathsf{KeyGen}(\mathsf{msk}, f)$*: takes as input the master secret key* msk *and a function $f \in \mathcal{F}[n, \ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{R}, \ell_\mathcal{Y}]$, and outputs a function key sk_f.*

[3]We also allow Enc, KeyGen, and Dec to additionally receive parameters $1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$ as input, but omit them from our notation for convenience.

- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct})$: *takes a function key* sk_f *and a ciphertext* ct*, and outputs a value* $y \in \{0,1\}^{\ell_\mathcal{Y}}$.

FE *satisfies **correctness** if for all polynomials* p*, there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$*, all* $\ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{Y} \le p(\lambda)$*, all* $x \in \{0,1\}^{\ell_\mathcal{X}}$*, and all* $f \in \mathcal{F}[n, \ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{R}, \ell_\mathcal{Y}]$,

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_x) = f(x) : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}) \\ \mathsf{ct}_x \leftarrow \mathsf{Enc}(\mathsf{mpk}, x) \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \end{array}\right] \ge 1 - \mu(\lambda).$$

We now define adaptive security.

Definition 3.12 (Adaptive Security for Public-Key FE)**.** *A public-key functional encryption scheme* FE *for* $\mathsf{P/Poly}$ *is adaptively secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and every PPT adversary* $\mathcal{A}$,

$$\left|\Pr[\mathsf{Expt}_\mathcal{A}^{\mathsf{FE\text{-}Adaptive}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}_\mathcal{A}^{\mathsf{FE\text{-}Adaptive}}(1^\lambda, 1) = 1]\right| \le \mu(\lambda)$$

where for each $b \in \{0,1\}$ *and* $\lambda \in \mathbb{N}$*, we define*

$\mathsf{Expt}_\mathcal{A}^{\mathsf{FE\text{-}Adaptive}}(1^\lambda, b)$

1. ***Parameters***: $\mathcal{A}$ *takes as input* 1^λ*, and outputs a function size* $1^{\ell_\mathcal{F}}$*, an input size* $1^{\ell_\mathcal{X}}$*, and an output size* $1^{\ell_\mathcal{Y}}$.
2. ***Setup***: *Compute* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$.
3. ***Public Key***: *Send* mpk *to* $\mathcal{A}$.
4. ***Function Queries Phase 1***: *The following is repeated any polynomial number of times:*
   (a) $\mathcal{A}$ *outputs a function query* $f \in \mathcal{F}[n, \ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{R}, \ell_\mathcal{Y}]$
   (b) $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f)$
   (c) *Send* sk_f *to* $\mathcal{A}$
5. ***Challenge Message Query***:
   (a) $\mathcal{A}$ *outputs a challenge message pair* (x_0, x_1) *where* $x_0, x_1 \in \{0,1\}^{\ell_\mathcal{X}}$.
   (b) $\mathsf{ct} \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, x_b)$
   (c) *Send* ct *to* $\mathcal{A}$.
6. ***Function Queries Phase 2***: *This is identical to Function Queries Phase 1.*
7. ***Experiment Outcome***: $\mathcal{A}$ *outputs a bit* b' *which is the output of the experiment.*

Additionally, when running the experiment, we immediately halt and output 0 *if the adversary ever aborts or if it at any point* $f(x_0) \ne f(x_1)$ *for some message query* (x_0, x_1) *and function query* f *submitted by the adversary.*

Definition 3.13 (Other Public-Key FE Security Definitions)**.** *There are many variations of the security definition. We list a few below:*

- ***Semi-Adaptive Security***: *The adversary is required to make the message query before the function queries. This is identical to adaptive security, except that we remove* Function Queries Phase 1 *from the security game.*

- ***Function-Selective Semi-Adaptive Security:*** *The adversary is required to make all function queries before the message query. This is identical to adaptive security, except that we remove* Function Queries Phase 2 *from the security game.*

- ***Selective Security:*** *The adversary is required to make the message query at the beginning of the experiment before receiving the master public key. This is similar to adaptive security, except that in the security game, we move the* Challenge Message Query *step so that it now lies between the* Setup *step and the* Public Key *step. Note that the two function query phases are now adjacent and can thus be merged into one step.*

- ***Function-Selective Security:*** *The adversary is required to make the function queries at the beginning of the experiment before receiving the master public key. This is similar to adaptive security, except that in the security game, we move the two function query steps so that they now lie between the* Setup *step and the* Public Key *step. Note that the two function query phases are now adjacent and can thus be merged into one step.*

3.3 Non-Interactive Zero Knowledge Proof Systems

Definition 3.14 (Non-Interactive Zero Knowledge Proof)**.** *Let $L \in \mathsf{NP}$ and R_L be the corresponding* NP *relation. Let $\lambda \in \mathbb{N}$ be the security parameter. A* Non-Interactive Zero Knowledge *(NIZK) Proof is a tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:*

- $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$*: Takes as input the security parameter 1^λ and outputs the common reference string* crs.

- $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)$*: Takes as input the common reference string* crs*, a statement x, and a witness w, and outputs a proof π.*

- $1/0 \leftarrow \mathsf{Verify}(\mathsf{crs}, x, \pi)$*: Takes as input the common reference string* crs*, a statement x and a proof π, outputs a single bit $1/0$ signaling whether the proof π is valid for the statement x.*

We require the following properties of a NIZK *scheme:*

- **Perfect Completeness***: For all security parameters $\lambda \in \mathbb{N}$, and all $(x, w) \in R_L$, we have*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 : \begin{matrix} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda); \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{matrix}\right] = 1.$$

- **Computational Adaptive Soundness***: For all PPT adversaries $\mathcal{A}$, we have*

$$\Pr[\mathsf{ExptSound}_{\Pi,\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{ExptSound}$ *is defined as below:*

$\mathsf{ExptSound}_{\Pi,\mathcal{A}}(\lambda)$*:*
1. $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$*;*
2. $(x, \pi) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs})$*;*
3. The experiment outputs 1 if and only if $x \notin L \wedge \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$.

- **Computational Zero-Knowledge**: *There exists a pair of PPT simulators* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *where* $\mathsf{Sim}_1(1^\lambda)$ *outputs* $(\widetilde{\mathsf{crs}}, \tau)$ *and* $\mathsf{Sim}_2(\widetilde{\mathsf{crs}}, \tau, x)$ *outputs a simulated proof* $\widetilde{\pi}$ *such that for all non-uniform PPT adversaries* $\mathcal{A}$*:*

$$\Big| \Pr\left[\mathcal{A}^{\mathcal{O}_1(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) = 1 : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)\right] - \Pr\left[\mathcal{A}^{\mathcal{O}_2(\mathsf{crs},\tau,\cdot,\cdot)}(\widetilde{\mathsf{crs}}) = 1 : (\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{Sim}_1(1^\lambda)\right] \Big| \leq \mathsf{negl}(\lambda),$$

where $\mathcal{O}_1, \mathcal{O}_2$ *on input* (x, w) *returns* $\perp$ *if* $(x, w) \notin R_L$*. Otherwise,* $\mathcal{O}_1$ *returns* $\mathsf{Prove}(\mathsf{crs}, x, w)$ *and* $\mathcal{O}_2$ *returns* $\mathsf{Sim}_2(\widetilde{\mathsf{crs}}, \tau, x)$*.*

We can also require a NIZK to have *Simulation Soundness*, saying that the protocol still has soundness after the adversary sees *simulated* proofs.

Definition 3.15 (Unbounded Simulation Soundness)**.** *Let* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *be the simulators for a NIZK protocol* $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *as defined in Computational Zero-Knowledge. We say* Π *has* (unbounded) simulation soundness *if for all PPT adversaries* $\mathcal{A}$*, we have*

$$\Pr\left[(x, \pi) \notin Q \wedge x \notin \mathcal{L} \wedge \mathsf{Verify}(\widetilde{\mathsf{crs}}, x, \pi) = 1 : \begin{array}{c} (\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{Sim}_1(1^\lambda); \\ (x, \pi) \leftarrow \mathcal{A}^{\mathsf{Sim}_2(\widetilde{\mathsf{crs}}, \tau, \cdot)}(\widetilde{\mathsf{crs}}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where Q *denotes the set of all* Sim_2 *queries by* $\mathcal{A}$ *and their corresponding responses* $(x_i, \widetilde{\pi}_i)$*.*

4 Improved Security Definitions for Randomized (Multi-Input) Functional Encryption (rMIFE)

The syntax of rMIFE follows naturally from that of MIFE and FE for randomized functionalities.

Definition 4.1 (Randomized Function Class)**.** *The randomized function class* $\mathcal{F}[n, \ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{R}, \ell_\mathcal{Y}]$ *is the set of all functions* $f : (\{0,1\}^{\ell_\mathcal{X}})^n \times \{0,1\}^{\ell_\mathcal{R}} \to \{0,1\}^{\ell_\mathcal{Y}}$ *that have a description* $\widehat{f} \in \{0,1\}^{\ell_\mathcal{F}}$*. We interpret each function* f *as a randomized function with arity* n *that takes as input values* $x_1, \ldots, x_n$ *where each* $x_i \in \{0,1\}^{\ell_\mathcal{X}}$ *and randomness* $r \in \{0,1\}^{\ell_\mathcal{R}}$ *out outputs a value* $y \in \{0,1\}^{\ell_\mathcal{Y}}$*.*

Definition 4.2 (rMIFE)**.** *A randomized multi-input functional encryption (rMIFE) scheme for* $\mathsf{P/Poly}$ *is a tuple of PPT algorithms* $\mathsf{rMIFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *defined as follows:*

- $\mathsf{Setup}(1^\lambda, 1^n, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{R}}, 1^{\ell_\mathcal{Y}})$*: takes as input the security parameter* λ*, a function arity* n*, a function size* $\ell_\mathcal{F}$*, an input size* $\ell_\mathcal{X}$*, a randomness size* $\ell_\mathcal{R}$*, and an output size* $\ell_\mathcal{Y}$*, out outputs* n *encryption keys* $\mathsf{EK}_1, \ldots, \mathsf{EK}_n$ *and the master secret key* MSK*.*

- $\mathsf{KeyGen}(\mathsf{MSK}, f)$*: takes as input the master secret key* MSK *and a function* $f \in \mathcal{F}[n, \ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{R}, \ell_\mathcal{Y}]$ *and outputs a function key* sk_f*.*

- $\mathsf{Enc}(\mathsf{EK}_j, x_j)$*: takes as input an encryption key* EK_j *and an input* $x_j \in \{0,1\}^{\ell_\mathcal{X}}$ *and outputs an encryption* ct_j *of* x_j*.*

- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$*: takes as input a function key* sk_f *and ciphertexts* $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ *and outputs a value* $y \in \{0,1\}^{\ell_\mathcal{Y}}$*.*

rMIFE *satisfies **correctness** if for all polynomials p, there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$*, all* $n, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{R}}, \ell_{\mathcal{Y}}, q \leq p(\lambda)$*, all* $\{f_k\}_{k\in[q]}$ *where each* $f_k \in \mathcal{F}[n, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{R}}, \ell_{\mathcal{Y}}]$*, all* $\{x_{i,1}, \ldots, x_{i,n}\}_{i\in[q]}$ *where each* $x_{i,j} \in \{0,1\}^{\ell_{\mathcal{X}}}$*, and all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{Real}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{A}}(1^\lambda) = 1\right| \leq \mu(1^\lambda)$$

where we define

- $\mathsf{Real}_{\mathcal{A}}(1^\lambda)$
  1. $(\mathsf{EK}_1, \ldots, \mathsf{EK}_n, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$.
  2. *For* $k \in [q]$, $\mathsf{sk}_k \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f_k)$.
  3. *For* $i \in [q], j \in [n]$, $\mathsf{ct}_{i,j} \leftarrow \mathsf{Enc}(\mathsf{EK}_j, x_{i,j})$.
  4. *For* $k, i_1, \ldots, i_n \in [q]$, $y_{k,i_1,\ldots,i_n} = \mathsf{Dec}(\mathsf{sk}_k, \mathsf{ct}_{i_1,1}, \ldots, \mathsf{ct}_{i_n,n})$.
  5. *Output* $\mathcal{A}^{\mathcal{O}(\cdot)}(1^\lambda)$ *where* $\mathcal{O}(k, i_1 \ldots, i_n) = y_{k,i_1,\ldots,i_n}$.
- $\mathsf{Ideal}_{\mathcal{A}}(1^\lambda)$
  1. *For* $k, i_1, \ldots, i_n \in [q]$,
     (a) $r_{k,i_1,\ldots,i_n} \leftarrow \{0,1\}^{\ell_{\mathcal{R}}}$
     (b) $y_{k,i_1,\ldots,i_n} = f_k(x_{i_1,1}, \ldots, x_{i_n,n}; r_{k,i_1,\ldots,i_n})$
  2. *Output* $\mathcal{A}^{\mathcal{O}(\cdot)}(1^\lambda)$ *where* $\mathcal{O}(k, i_1 \ldots, i_n) = y_{k,i_1,\ldots,i_n}$.

Definition 4.3 (Simulation-based Security for rMIFE)**.** *A randomized multi-input functional encryption scheme* rMIFE = (Setup, KeyGen, Enc, Dec) *for* P/Poly *is* (q_1, q_c, q_2) *simulation-secure against both malicious encryptors and decryptors if there exists a* PPT *simulator* $S = (S_1, S_2, \ldots, S_5)$ *such that for all sufficiently large* λ*, and all* PPT *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, we have*

$$\left|\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rMIFE,Real}}(1^\lambda) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A},\mathcal{S}}^{\mathsf{rMIFE,Ideal}}(1^\lambda) = 1\right| \leq \mathsf{negl}(\lambda)$$

where we define,

- $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rMIFE,Real}}(1^\lambda)$
  1. $(\mathsf{EK}_1, \ldots, \mathsf{EK}_n, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{R}}}, 1^{\ell_{\mathcal{Y}}})$.
     We define $\overrightarrow{\mathsf{EK}} = (\mathsf{EK}_1, \ldots, \mathsf{EK}_n)$
  2. $(\{(x_{i,j})\}_{i\in[n],j\in[q_c]}, st) \leftarrow \mathcal{A}_1^{\mathsf{OGetEK}(\overrightarrow{\mathsf{EK}},\cdot),\mathsf{KeyGen}(\mathsf{MSK},\cdot),\mathsf{OKeyStore}(\mathsf{MSK},\cdot),\mathsf{ODec}(\mathsf{MSK},\cdot)}(1^\lambda)$.
  3. $ct^*_{i,j} \leftarrow \mathsf{Enc}(\mathsf{EK}_i, x_{i,j})$ *for all* $i \in [n], j \in [q_c]$.
  4. $\alpha \leftarrow \mathcal{A}_2^{\mathsf{OGetEK}(\overrightarrow{\mathsf{EK}},\cdot),\mathsf{KeyGen}(\mathsf{MSK},\cdot),\mathsf{OKeyStore}(\mathsf{MSK},\cdot),\mathsf{ODec}(\mathsf{MSK},\cdot)}(\{\mathsf{ct}^*_{i,j}\}_{i\in[n],j\in[q_c]}, \mathsf{st})$.
  5. ***Output*** $(\{(x_{i,j})\}_{i\in[n],j\in[q_c]}, \{f_k\}_{k\in[q_1+q_2]}, \{g\}, \{y\}, \alpha)$.
- $\mathsf{Expt}_{\mathcal{A},\mathcal{S}}^{\mathsf{rMIFE,Ideal}}(1^\lambda)$
  1. $\mathsf{st}' \leftarrow S_1(1^\lambda, 1^n, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{R}}}, 1^{\ell_{\mathcal{Y}}})$.
  2. $(\{(x_{i,j})\}_{i\in[n],j\in[q_c]}, st) \leftarrow \mathcal{A}_1^{\mathsf{O'GetEK}(\mathsf{st}',\cdot),\mathsf{O'KeyGen}_1(\mathsf{st}',\cdot),\mathsf{O'KeyStore}(\mathsf{st}',\cdot),\mathsf{ODec}'(\mathsf{st}',\cdot)}(1^\lambda)$.

– *Let* $\{f_k\}_{k\in[q_1]}$ *be* $\mathcal{A}_1$*'s oracle queries to* $\mathsf{O'KeyGen}_1(\mathsf{st}', \cdot)$.
– *Pick* $r_{j_1,\ldots,j_n,k} \leftarrow \{0,1\}^{\ell_{\mathcal{R}}}$ *and let* $y_{j_1,\ldots,j_n,k} = f_k(x_{1,j_1}, \ldots, x_{n,j_n}; r_{j_1,\ldots,j_n,k})$ *for all* $j_1, \ldots, j_n \in [q_c], k \in [q_1]$.

3. $(\{ct^*_{i,j}\}_{i\in[n],j\in[q_c]}, \mathsf{st}') \leftarrow S_3(\mathsf{st}', \{y_{j_1,\ldots,j_n,k}\}_{j_1,\ldots,j_n\in[q_c],k\in[q_1]})$.
4. $\alpha \leftarrow \mathcal{A}_2^{\mathsf{O'GetEK}(\mathsf{st}',\cdot),\mathsf{O'KeyGen}_2(\mathsf{st}',\cdot),\mathsf{O'KeyStore}(\mathsf{st}',\cdot),\mathsf{O'Dec}(\mathsf{st}',\cdot)}(\{\mathsf{ct}^*_{i,j}\}_{i\in[n],j\in q_c}, st)$.

- ***Output*** $(\{(x_{i,j})\}_{i\in[n],j\in[q_c]}, \{f_k\}_{k\in[q_1+q_2]}, \{g\}, \{y'\}, \alpha)$.

and where we define

In Real Experiment:

- $\mathsf{OGetEK}(\overrightarrow{\mathsf{EK}}, i)$*: Output* EK_i.
- $\mathsf{OKeyStore}(\mathsf{MSK}, g)$*:*
  1. $\mathsf{sk}_g \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, g)$.
  2. *Store* (g, sk_g) *in register* KeyReg.
  3. *Output* $\perp$.
- $\mathsf{ODec}(\{\mathsf{ct}^*_{i,j}\}_{i\in[n],j\in[q_c]}, \{\mathsf{ct}_i\}_{i\in[n]})$*:*
  1. *For each* (g_l, sk_{g_l}) *stored in register* KeyReg*:* $y_l = \mathsf{Dec}(\mathsf{sk}_{g_l}, \{\mathsf{ct}_i\}_{i\in[n]})$.
  2. *Output* $\{y_l\}$.

In Ideal Experiment:

- $\mathsf{O'GetEK}(st', i)$*: Output* EK'_i
- $\mathsf{O'KeyGen}_1(\mathsf{st}', f_k)$
  1. $sk'_{f_k} \leftarrow S_2(\mathsf{st}', f_k)$
  2. *Output* $\{sk'_{f_k}\}$
- $\mathsf{O'KeyGen}_2(\mathsf{st}', f_k)$
  1. $sk'_{f_k} \leftarrow S_4(\mathsf{st}', f_k, KeyIdeal(\{x_{i,j}\}_{i\in[n],j\in q_c}))$
  2. *Output* sk'_{f_k}
  3. $KeyIdeal(\{x_{i,j}\}_{i\in[n],j\in q_c})$
     (a) $r_{j_1,\ldots,j_n,k} \leftarrow \{0,1\}^{\ell_{\mathcal{R}}}, j_n \in [q_c]$
     (b) $y_{j_1,\ldots,j_n,k} = f_k(x_{1,j_1}, \ldots, x_{n,j_n}; r_{j_1,\ldots,j_n,k})$ *for all* $j_1, \ldots, j_n \in [q_c]$
     (c) *Output* $\{y_{j_1,\ldots,j_n,k}\}$ *for all* $j_1, \ldots, j_n \in [q_c]$.
- $\mathsf{O'KeyStore}(\mathsf{st}', g)$*:*
  1. *Store* $(g, \perp)$ *in register* KeyReg.
  2. *Output* $\perp$.

- $\mathsf{O'Dec}(\mathsf{st}', \{\mathsf{ct}^*_{i,j}\}_{i\in[n],j\in q[c]}, \{ct_i\}_{i\in[n]})$:

   1. *For each* $(g_l, \perp)$ *stored in register* KeyReg*:*
      (a) *If* $(l, \{\mathsf{ct}_i\}_{i\in[n]}, y)$ *is stored in register* OutReg *for some* y*, output* y*.*
      (b) *For* $i \in [n]$*,* $x_i = S_5(\mathsf{st}', \mathsf{ct}_i)$*.*
      (c) $r_l \leftarrow \{0,1\}^{\ell_{\mathcal{R}}}$*.*
      (d) $y_l \leftarrow g_l(\{x_i\}_{i\in[n]}; r_l)$*.*
      (e) *Store* $(l, \{\mathsf{ct}_i\}_{i\in[n]}, y_l)$ *in* OutReg*.*
   2. *Output* $\{y_l\}$*.*

Definition 4.4 ($(\mathcal{A}, \epsilon)$-$\mathcal{I}$-randomized-compatible)**.** *Let* $n, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{R}}, \ell_{\mathcal{Y}}, q_1, q_2 \in \mathbb{N}$*.*

- *Let* $\mathcal{I} \subseteq [n]$*.*
- *Let* $\{(x^{(0)}_{i,j}, x^{(1)}_{i,j})\}_{i\in[n],j\in[q_1]}$ *be a set of inputs where each* $x^{(b)}_{i,j} \in \{0,1\}^{\ell_{\mathcal{X}}}$*.*
- *Let* $\{f_k\}_{k\in[q_2]}$ *be a set of functions where each* $f_k \in \mathcal{F}[n, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{R}}, \ell_{\mathcal{Y}}]$*.*

We say that $\{x^{(0)}_{i,j}, x^{(1)}_{i,j}\}_{i\in[n],j\in[q_1]}$ *is* ϵ*-*$\mathcal{I}$*-randomized-compatible with* $\{f_k\}_{k\in[q_2]}$ *if*

- *For all* $U \subseteq I$*, all* $\{x'_u\}_{u\in U}$ *where each* $x'_u \in \{0,1\}^{\ell_{\mathcal{X}}}$*, all* $\{j_t\}_{t\in[n]\setminus U}$ *where each* $j_t \in [q_1]$*, and all* $k \in [q_2]$*,*

$$\Big|\Pr[\mathcal{A}(f_k, U, \{x'_u\}_{u\in U}, \{x^{(0)}_{t,j_t}\}_{t\in[n]\setminus U}, f_k(\langle\{x'_u\}_{u\in U}, \{x^{(0)}_{t,j_t}\}_{t\in[n]\setminus U}\rangle)) = 1]$$
$$- \Pr[\mathcal{A}(f_k, U, \{x'_u\}_{u\in U}, \{x^{(0)}_{t,j_t}\}_{t\in[n]\setminus U}, f_k(\langle\{x'_u\}_{u\in U}, \{x^{(1)}_{t,j_t}\}_{t\in[n]\setminus U}\rangle)) = 1]\Big| \leq \epsilon$$

where $\langle\{x'_u\}_{u\in U}, \{x^{(b)}_{t,j_t}\}_{t\in[n]\setminus U}\rangle$ *is a permutation* $(x_1, \ldots, x_n)$ *such that*

$$x_i = \begin{cases} x'_i & \text{if } i \in U \\ x^{(b)}_{i,j_i} & \text{if } i \in [n]\setminus U \end{cases}$$

Definition 4.5 (Indistinguishability Based Security Against Malicious decryptors for rMIFE)**.** *A randomized multi-input functional encryption scheme* $\mathsf{rMIFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *for* $\mathsf{P/Poly}$ *is IND secure against malicious receivers for* $\epsilon = \epsilon(\lambda)$*-distinguishable distributions if for all sufficiently large* λ*, and all* PPT *adversaries* $\mathcal{A}$*,*

$$\Pr[\mathsf{Expt}^{\mathsf{rMIFE\text{-}Decryptors}}_{\mathcal{A}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where we define

$\mathsf{Expt}^{\mathsf{rMIFE\text{-}Decryptors}}_{\mathcal{A}}(1^\lambda)$:

1. ***Parameters****:* $\mathcal{A}$ *takes as input* 1^λ*, and outputs an arity* 1^n*, a function size* $1^{\ell_{\mathcal{F}}}$*, an input size* $1^{\ell_{\mathcal{X}}}$*, a randomness size* $1^{\ell_{\mathcal{R}}}$*, and an output size* $1^{\ell_{\mathcal{Y}}}$*.*
2. ***Setup****:* $(\mathsf{EK}_1, \ldots, \mathsf{EK}_n, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{R}}}, 1^{\ell_{\mathcal{Y}}})$*.*
   We define $\overrightarrow{\mathsf{EK}} = (\mathsf{EK}_1, \ldots, \mathsf{EK}_n)$*.*

3. ***Challenge Bit:*** $b \leftarrow \{0,1\}$.

4. ***Adversary's Output:*** $b' \leftarrow \mathcal{A}^{\mathsf{OGetEK}(\overrightarrow{\mathsf{EK}},\cdot),\mathsf{OEncLR}(\overrightarrow{\mathsf{EK}},b,\cdot),\mathsf{KeyGen}(\mathsf{MSK},\cdot)}$.

5. ***Experiment Output:*** *Output* 1 *if* $b = b'$ *and if* $\{(x_{i,j}^{(0)}, x_{i,j}^{(1)})\}_{i\in[n],j\in[q_1]}$ *are* $(\mathcal{A},\epsilon) - \mathcal{I}$*-randomized-compatible with* $\{f_k\}_{k\in[q_2]}$ *where*
   - $\mathcal{I}$ *are the set of queries made to* OGetEK *by* $\mathcal{A}$.
   - $\{(x_{i,j}^{(0)}, x_{i,j}^{(1)})\}_{i\in[n],j\in[q_1]}$ *are the set of queries made to* OEncLR *by* $\mathcal{A}$.
   - $\{f_k\}_{k\in q_2}$ *are the set of queries made to* KeyGen *by* $\mathcal{A}$.

and where we define

$\mathsf{OGetEK}(\overrightarrow{\mathsf{EK}}, j)$: *Output* EK_j.

$\mathsf{OEncLR}(\overrightarrow{\mathsf{EK}}, b, \{(x_i^{(0)}, x_i^{(1)})\}_{i\in[n]})$

1. *For* $i \in [n]$,

   (a) $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{MIFE.Enc}(\mathsf{EK}_i, x_i^{(b)})$.

2. *Output* $\mathsf{CT}_b = \{\mathsf{ct}_i^{(b)}\}_{i\in[n]}$.

Definition 4.6 (Indistinguishability Based Security Against Malicious Encryptors for rMIFE)**.** *A randomized multi-input functional encryption scheme* rMIFE = (Setup, KeyGen, Enc, Dec) *for* P/Poly *is IND secure against malicious encryptors if there exists a* PPT *extractor* Extr *such that for all* $\lambda \in \mathbb{N}$, *and all* PPT *adversaries* $\mathcal{A}$,

$$\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rMIFE\text{-}Encryptors}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where we define

$\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rMIFE\text{-}Encryptors}}(1^\lambda)$:

1. ***Parameters:*** $\mathcal{A}$ *takes as input* 1^λ, *and outputs an arity* 1^n, *a function size* $1^{\ell_{\mathcal{F}}}$, *an input size* $1^{\ell_{\mathcal{X}}}$, *a randomness size* $1^{\ell_{\mathcal{R}}}$, *and an output size* $1^{\ell_{\mathcal{Y}}}$.

2. ***Setup:*** $(\mathsf{EK}_1, \ldots, \mathsf{EK}_n, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{R}}}, 1^{\ell_{\mathcal{Y}}})$.
   We define $\overrightarrow{\mathsf{EK}} = (\mathsf{EK}_1, \ldots, \mathsf{EK}_n)$.

3. ***Challenge Bit:*** $b \leftarrow \{0,1\}$.

4. ***Adversary's Output:***
   $b' \leftarrow \mathcal{A}^{\mathsf{OGetEK}(\overrightarrow{\mathsf{EK}},\cdot),\mathsf{OEnc}(\overrightarrow{\mathsf{EK}},\cdot),\mathsf{KeyGen}(\mathsf{MSK},\cdot),\mathsf{OKeyStore}(\mathsf{MSK},\cdot),\mathsf{ODec}(\mathsf{MSK},b,\cdot)}$.

5. ***Experiment Output:*** *Output* 1 *if* $b = b'$.

and where we define

$\mathsf{OGetEK}(\overrightarrow{\mathsf{EK}}, j)$: *Output* EK_j.

$\mathsf{OEnc}(\overrightarrow{\mathsf{EK}}, \{x_i\}_{i\in[n]})$

1. *For* $i \in [n]$,

   (a) $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{EK}_i, x_i)$.

2. *Output* $\mathsf{CT} = \{\mathsf{ct}_i\}$.

$\mathsf{OKeyStore}(\mathsf{MSK}, g)$:

1. $\mathsf{sk}_g \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, g)$.

2. *Store* (g, sk_g) *in register* KeyReg.

3. *Output* $\perp$.

$\mathsf{ODec}(\mathsf{MSK}, b, \{\mathsf{ct}_i\}_{i\in[n]})$:

1. *For each* (g_j, sk_{g_j}) *stored in register* KeyReg:

   (a) *If* $(j, \{\mathsf{ct}_i\}_{i\in[n]}, y)$ *is stored in register* OutReg *for some* y, *output* y.

   (b) *Else if* $b = 0$,

      i. $y_j = \mathsf{Dec}(\mathsf{sk}_{g_j}, \{\mathsf{ct}_i\}_{i\in[n]})$.

      ii. *Store* $(j, \{\mathsf{ct}_i\}_{i\in[n]}, y_j)$ *in* OutReg.

   (c) *Else if* $b = 1$,

      i. *For* $i \in [n]$, $x_i \leftarrow \mathsf{Extr}(\mathsf{MSK}, \mathsf{ct}_i)$.

      ii. $r_j \leftarrow \{0,1\}^{\ell_{\mathcal{R}}}$.

      iii. $y_j \leftarrow g_j(\{x_i\}_{i\in[n]}; r_j)$.

      iv. *Store* $(j, \{\mathsf{ct}_i\}_{i\in[n]}, y_j)$ *in* OutReg.

2. *Output* $\{y_j\}$.

Remark 4.7 (On Imposing Equivalence Relations over Ciphertexts.)**.** The work of [AW17] define randomized (single-input) functional encryption with respect to "admissible ciphertext equivalence relations."

In more detail, in their scheme (and some prior schemes such as [GJKS15]), it may be the case that $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}) = \mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}')$ even though $\mathsf{ct} \neq \mathsf{ct}'$. This could occur for example if the ciphertext $\mathsf{ct} = (c, \pi)$ where π only serves to prove the validity of c, but does not otherwise contribute to the decryption output. Thus, if we were to generate a different proof π' attesting to the validity of c, then we could have two different valid ciphertexts $\mathsf{ct} = (c, \pi)$ and $\mathsf{ct}' = (c, \pi')$ that decrypt to the same value on every function key.

However, this means that their scheme would be insecure if we define security against malicious encryptors in the manner which we have done. This is because the decryption oracle ODec' in the ideal world will output independently generated values whenever it decrypts two different ciphertexts ct and ct'. However, in the real world, the decryption of these two ciphertexts may be the same value, leading to a trivial distinguisher between the real and ideal worlds.

To handle this, [AW17] propose the notion of an admissable ciphertext equivalence relation which is an efficiently checkable relation where two ciphertexts are considered equivalent if they decrypt to the same value.[4][5] Then, in their security game, they require that the adversary does not query the decryption oracle on two different ciphertexts from the same equivalence class. This restriction allows them to prove the security of their scheme under some reasonable notion of security.

Although our definition of security is different from that of [AW17], we believe that both [AW17] and [GJKS15] will be secure under our definitions if we additionally add the restriction that an adversary cannot query the decryption oracle on two different ciphertexts from the same admissable equivalence class.

We further remark that the rMIFE scheme we construct in this paper does not require this restriction or any notion of ciphertext equivalence relations and can be proven secure as per our main definitions given above.

Remark 4.8 (On SIM Security of [GGG+14, GJKS15, AW17] under our new definition.)**.** As mentioned above, the existing $\mathsf{rFE}/\mathsf{rMIFE}$ constructions from [GGG+14, GJKS15, AW17], which were proven secure under the previous SIM-security definition, also achieve SIM-security under our new definition (of course with the equivalence relations restriction mentioned in 4.7). Observe that the key distinction between our new definition and the prior one lies in the handling of secret keys within the decryption oracle. In previous definitions, the decryption oracle generates a fresh secret key for the function each time a decryption is performed involving that function. In contrast, under our new definition, the secret key for the function is generated once and reused for subsequent decryptions involving that function. Importantly, the security proofs in the prior works [GGG+14, GJKS15, AW17] do not rely on the generation of fresh secret keys for each decryption by the decryption oracle. As a result, their security proofs should naturally extend to our new definition without modification.

Remark 4.9 (On submitting multiple ciphertexts to the decryption oracle.)**.** Recall that [AW17] extends the original SIM security definition from [GJKS15] by allowing the adversary to submit multiple ciphertexts to the decryption oracle simultaneously, rather than one at a time. As noted in [AW17], this modification captures security against adversarially generated correlated ciphertexts, which could potentially influence the decryption outputs.

In contrast, our definition, similar to [GJKS15], allows the adversary to submit a single ciphertext (or a single tuple of ciphertexts in the multi-input case) to the decryption oracle at a time. However, this does not make our model more restrictive compared to [AW17]. In their model, the decryption oracle generates a fresh secret key for each randomized function every time a decryption query is made, and this key is used to decrypt the set of ciphertexts submitted along with the function. In our definition, the same secret keys are used consistently for decrypting all ciphertexts submitted over time.

Therefore, our model naturally addresses attacks involving adversarially generated correlated ciphertexts, even though we submit one ciphertext at a time to the decryption oracle.

[4]In more detail, consider an equivalence relation $\sim$ on the ciphertext space. We say that $\sim$ is admissible if it is efficiently checkable and if for every two ciphertexts $\mathsf{ct}^{(0)}$ and $\mathsf{ct}^{(1)}$, then $\mathsf{ct}^{(0)} \sim \mathsf{ct}^{(1)}$ if and only if for any honestly generated function key sk_f, one of the following holds: (1) $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}^{(0)}) = \perp$ or $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}^{(1)}) = \perp$, or (2) $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}^{(0)}) = \mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}^{(1)})$.

[5]This notion can also be extended to the multi-input setting by defining an equivalence relation on sets $\{\mathsf{ct}_i\}_{i\in[n]}$ of ciphertext queries where n is the arity of the function.

Lemma 4.10 (SIM-security implies IND-security for rFE/rMIFE.)**.** *Let* $\mathsf{rMIFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be* SIM*-secure as per Definition 4.3. Then,* rMIFE *is* IND*-secure against both malicious encryptors and malicious decryptors as per Definitions 4.5 and 4.6, respectively.*

The proof that SIM-security of rMIFE as per definition 4.3 implies IND-security against malicious decryptors (definition 4.5) is essentially the same as the one in [GJKS15]. Please refer to the proof of Lemma 2.9, Appendix C in [GJKS15] for details.

To show that SIM-security implies IND-security against malicious encryptors (Definition 4.6), we proceed as follows. Observe that $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rMIFE\text{-}Encryptors}}(1^\lambda)$ with challenge bit $b = 0$ is equivalent to $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rMIFE,Real}}(1^\lambda)$ where the adversary issues a zero-challenge ciphertext query. Similarly, $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rMIFE\text{-}Encryptors}}(1^\lambda)$ with challenge bit $b = 1$ is identical to $\mathsf{Expt}_{\mathcal{A},\mathcal{S}}^{\mathsf{rMIFE,Ideal}}(1^\lambda)$ under the same zero-challenge query.

Since $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rMIFE,Real}}(1^\lambda)$ and $\mathsf{Expt}_{\mathcal{A},\mathcal{S}}^{\mathsf{rMIFE,Ideal}}(1^\lambda)$ are computationally indistinguishable under the SIM-security guarantee, it follows that $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rMIFE\text{-}Encryptors}}(1^\lambda)$ with $b = 0$ and $b = 1$ are also computationally indistinguishable. Hence, SIM-security implies IND-security against malicious encryptors.

5 Counterexample

In this section, we provide a counterexample showcasing the issue with the *indistinguishability-based* definition of Functional Encryption for Randomized Functionalities as presented in the work by Goyal, Jain, Koppula, and Sahai [GJKS15]. This definition has been used in a line of follow-up works, and therefore we think it is crucial that the insufficiency of the definition be pointed out.

5.1 Definition in [GJKS15]

First, we recall the indistinguishability-based definition in [GJKS15]. [GJKS15] provides two IND definitions, corresponding to whether the adversary needs to submit the function queries before or after receiving the master public key mpk. Here we focus on the more adaptive $\mathsf{IND}_{\mathsf{post}}$ security, where the adversary sends function queries after receiving the master public key.

Definition 5.1 (IND-based Definition of rFE as in [GJKS15])**.** *A functional encryptions scheme for randomized functionalities* $\mathsf{rFE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *is IND-secure if for all PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$[6]*, we have*

$$\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rFE}}(1^\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rFE}}$ *is defined as below.*

$\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rFE}}(1^\lambda)$:

1. $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.
2. *Sample* $b \leftarrow \{0,1\}$ *and* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$*, and compute* $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_b)$.
3. $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$*, where* KeyGen *is the key generation oracle, and* $\mathcal{O}$ *is defined as below.*

[6] The original definition in [GJKS15] considers non-uniform attackers to allow for a more general definition, but the non-uniformity is never utilized anywhere in the proofs. So here we present the definition and counterexample against uniform attackers, but bear in mind that it can be easily adapted to allow for non-uniform attackers if one desires so.

4. *Let $\{f\}$ be the set of function queries by $\mathcal{A}_2$ to the key generation oracle. We require that the distributions $(\mathsf{mpk}, \mathsf{st}, \{f(x_0)\})$ and $(\mathsf{mpk}, \mathsf{st}, \{f(x_1)\})$ are statistically indistinguishable*[a]*. If not, the experiment aborts and outputs* 0.

5. *If $b' = b$, the adversary wins and the game outputs 1. Otherwise, the adversary loses and the game outputs 0.*

[a] Statistical indistinguishability is necessary here to prevent circularity. If we only require *computational* indistinguishability, $\mathcal{A}_2$ can submit a query $f = \mathsf{Enc}(\mathsf{mpk}, \cdot)$. Then, the indistinguishably requirement on these two distribution will be the same as the desired security requirement for the challenge ciphertext, making this a vacuous definition.

$\mathcal{O}_{\mathsf{CT}^*}(\mathsf{msk}, \mathsf{CT}, g)$:

1. *If* $\mathsf{CT} = \mathsf{CT}^*$*, return* $\perp$.
2. *Compute* $\mathsf{SK}_g \leftarrow \mathsf{KeyGen}(\mathsf{msk}, g)$.
3. *Return* $\mathsf{Dec}(\mathsf{SK}_g, \mathsf{CT})$.

5.2 Construction of Counterexample

Now, we present our construction of a rFE scheme that satisfies Definition 5.1, but is insecure. We use a number of tools, the definitions of which can be found in Section 3 of the supplementary material.

Construction 1 (Counterexample rFE)**.** Let $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure PKE scheme, $\mathsf{SKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure symmetric key encryption, $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a NIZK with simulation soundness, $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ be a plain Functional Encryption scheme with selective security, and PRF be a secure PRF with output length $\ell_{\mathsf{PRF}} = \mathsf{poly}(\lambda)$. We construct our randomized FE scheme $\mathsf{rFE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ as follows:

- $\mathsf{Setup}(1^\lambda)$:
  1. $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
  2. $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
  3. $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
  4. $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$.
  5. Output $(\mathsf{mpk}, \mathsf{msk}) = ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \mathsf{crs}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \mathsf{crs}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.
- $\mathsf{Enc}(\mathsf{mpk}, x)$:
  1. Parse $\mathsf{mpk} = (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \mathsf{crs})$.
  2. $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
  3. $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (x, K, 0, \perp))$.
  4. $\mathsf{PKE.ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, (x, K, 0, \perp, \mathsf{FE.ct}))$.
  5. $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, z, w)$ where z is the following statement on $\mathsf{FE.ct}$ and $\mathsf{PKE.ct}$:
     "$\mathsf{PKE.ct}$ *correctly encrypts* $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct})$*, where* $(x, K, \alpha, \widehat{\mathsf{sk}})$ *is encrypted in* $\mathsf{FE.ct}$"
     The witness w for the statement is the tuple $(x, K, \alpha = 0, \widehat{\mathsf{sk}} = \perp)$.

6. Output $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.

- $\mathsf{KeyGen}(\mathsf{msk}, f)$:
  1. Parse $\mathsf{msk} = (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \mathsf{crs}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk})$.
  2. $s \leftarrow \{0,1\}^\lambda$.
  3. $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|f(\cdot)|}$ is randomly sampled.
  4. $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.
  5. Output $\mathsf{SK}_f = (\mathsf{crs}, \mathsf{FE.sk}_{G_f})$.

$G[f, s, \mathsf{SKE.ct}](x, K, \alpha, \widehat{\mathsf{sk}})$:

1. If $\alpha = 0$:
   (a) $r \leftarrow \mathsf{PRF.Eval}(K, s)$.
   (b) Output $f(x; r)$.
2. If $\alpha = 1$:
   (a) Output $\mathsf{SKE.Dec}(\widehat{\mathsf{sk}}, \mathsf{SKE.ct})$.

- $\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT})$:
  1. Parse $\mathsf{SK}_f = (\mathsf{crs}, \mathsf{FE.sk}_{G_f})$ and $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
  2. If $\mathsf{NIZK.Verify}(\mathsf{crs}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, abort and output $\perp$.
  3. Output $y = \mathsf{FE.Dec}(\mathsf{FE.sk}_{G_f}, \mathsf{FE.ct})$.

Correctness follows from the correctness of the underlying FE scheme.

5.3 Proof of (Insufficient) Security

In this section, we prove (insufficient) security of Construction 1 according to Definition 5.1.

Theorem 5.2. *If* PKE *is a CPA-secure PKE,* SKE *is a CPA-secure SKE,* NIZK *is a NIZK with simulation soundness,* FE *is a selectively-secure FE scheme, and* PRF *is a secure PRF, then Construction 1 is IND-secure per Definition 5.1.*

We prove this through a hybrid proof. We first lay out the sequence of hybrids, followed by proofs of each hybrid argument.

Sequence of Hybrids

$\mathbf{Hybrid}_0^{\mathcal{A}}$: The same as $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rFE}}$ where the challenge bit b is fixed to be 0. More specifically:

1. $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.
2. Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ as follows:
   (a) $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
   (b) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
   (c) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.

(d) $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$.

(e) $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \mathsf{crs}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \mathsf{crs}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.

3. Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0)$ as follows:

   (a) $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.

   (b) $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (x_0, K, 0, \perp))$.

   (c) $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, (x_0, K, 0, \perp, \mathsf{FE.ct}^*))$.

   (d) $\pi^* \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, z, w)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$:

   "$\mathsf{PKE.ct}^*$ correctly encrypts $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$, where $(x, K, \alpha, \widehat{\mathsf{sk}})$ is encrypted in $\mathsf{FE.ct}^*$"

   The witness w is the tuple $(x_0, K, 0, \perp)$.

   (e) Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.

4. $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$, where KeyGen and $\mathcal{O}$ are oracles defined as below:

   - $\mathsf{KeyGen}(\mathsf{msk}, f)$:

     (a) $s \leftarrow \{0,1\}^\lambda$.

     (b) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|f(\cdot)|}$ is randomly sampled.

     (c) $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.

     (d) Output $\mathsf{SK}_f = (\mathsf{crs}, \mathsf{FE.sk}_{G_f})$.

   - $\mathcal{O}(\mathsf{msk}, \mathsf{CT}, g)$:

     (a) If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.

     (b) $s \leftarrow \{0,1\}^\lambda$.

     (c) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.

     (d) $\mathsf{FE.sk}_{G_g} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[g, s, \mathsf{SKE.ct}])$.

     (e) Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.

     (f) If $\mathsf{NIZK.Verify}(\mathsf{crs}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.

     (g) Return $\mathsf{FE.Dec}(\mathsf{SK}_{G_g}, \mathsf{FE.ct})$.

5. Let $\{f\}$ be the set of function queries by $\mathcal{A}_2$ to the KeyGen oracle. We require that the distributions $(\mathsf{mpk}, \mathsf{st}, \{f(x_0)\})$ and $(\mathsf{mpk}, \mathsf{st}, \{f(x_1)\})$ are statistically indistinguishable. If not, the experiment aborts and outputs 0.

6. If $b' = b$, the adversary wins and the game outputs 1. Otherwise, the adversary loses and the game outputs 0.

$\mathbf{Hybrid}_1^{\mathcal{A}}$: Instead of sampling crs using $\mathsf{NIZK.Setup}$, now sample a simulated $\widetilde{\mathsf{crs}}$ using $\mathsf{NIZK.Sim}_1(1^\lambda)$. And later replace $\mathsf{NIZK.Prove}$ with $\mathsf{NIZK.Sim}_2$. This step follows from the computational zero-knowledge of the underlying NIZK.

1. $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.

2. Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ as follows:

   (a) $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.

(b) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.

(c) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.

(d) $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$.

(e) $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.

3. Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0)$ as follows:

   (a) $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.

   (b) $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (x_0, K, 0, \perp))$.

   (c) $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, (x_0, K, 0, \perp, \mathsf{FE.ct}^*))$.

   (d) $\pi^* \leftarrow \mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, z)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$:

   "$\mathsf{PKE.ct}^*$ *correctly encrypts* $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$, *where* $(x, K, \alpha, \widehat{\mathsf{sk}})$ *is encrypted in* $\mathsf{FE.ct}^*$"

   ~~The witness w is the tuple $(x_0, K, 0, \perp)$.~~

   (e) Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.

4. $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$, where KeyGen and $\mathcal{O}$ are oracles defined as below:

   - $\mathsf{KeyGen}(\mathsf{msk}, f)$:

     (a) $s \leftarrow \{0,1\}^\lambda$.

     (b) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|f(\cdot)|}$ is randomly sampled.

     (c) $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.

     (d) Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.

   - $\mathcal{O}(\mathsf{msk}, \mathsf{CT}, g)$:

     (a) If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.

     (b) $s \leftarrow \{0,1\}^\lambda$.

     (c) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.

     (d) $\mathsf{FE.sk}_{G_g} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[g, s, \mathsf{SKE.ct}])$.

     (e) Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.

     (f) If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.

     (g) Return $\mathsf{FE.Dec}(\mathsf{SK}_{G_g}, \mathsf{FE.ct})$.

5. Let $\{f\}$ be the set of function queries by $\mathcal{A}_2$ to the KeyGen oracle. We require that the distributions $(\mathsf{mpk}, \mathsf{st}, \{f(x_0)\})$ and $(\mathsf{mpk}, \mathsf{st}, \{f(x_1)\})$ are statistically indistinguishable. If not, the experiment aborts and outputs 0.

6. If $b' = b$, the adversary wins and the game outputs 1. Otherwise, the adversary loses and the game outputs 0.

Hybrid$_2^{\mathcal{A}}$: When generating the challenge ciphertext, change $\mathsf{PKE.ct}^*$ into an encryption of $\perp$. This step follows from CPA security of the underlying PKE scheme.

1. $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.

2. Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ as follows:

(a) $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
(b) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
(c) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
(d) $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$.
(e) $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.

3. Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0)$ as follows:
   (a) $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
   (b) $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (x_0, K, 0, \perp))$.
   (c) $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \perp)$.
   (d) $\pi^* \leftarrow \mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, z)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$:
   "$\mathsf{PKE.ct}^*$ *correctly encrypts* $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$*, where* $(x, K, \alpha, \widehat{\mathsf{sk}})$ *is encrypted in* $\mathsf{FE.ct}^*$"
   (e) Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.

4. $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$, where KeyGen and $\mathcal{O}$ are oracles defined as below:
   - $\mathsf{KeyGen}(\mathsf{msk}, f)$:
     (a) $s \leftarrow \{0,1\}^\lambda$.
     (b) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|f(\cdot)|}$ is randomly sampled.
     (c) $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.
     (d) Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.
   - $\mathcal{O}(\mathsf{msk}, \mathsf{CT}, g)$:
     (a) If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.
     (b) $s \leftarrow \{0,1\}^\lambda$.
     (c) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.
     (d) $\mathsf{FE.sk}_{G_g} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[g, s, \mathsf{SKE.ct}])$.
     (e) Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
     (f) If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.
     (g) Return $\mathsf{FE.Dec}(\mathsf{SK}_{G_g}, \mathsf{FE.ct})$.

5. Let $\{f\}$ be the set of function queries by $\mathcal{A}_2$ to the KeyGen oracle. We require that the distributions $(\mathsf{mpk}, \mathsf{st}, \{f(x_0)\})$ and $(\mathsf{mpk}, \mathsf{st}, \{f(x_1)\})$ are statistically indistinguishable. If not, the experiment aborts and outputs 0.

6. If $b' = b$, the adversary wins and the game outputs 1. Otherwise, the adversary loses and the game outputs 0.

Hybrid$_3^{\mathcal{A}}$: In $\mathcal{O}$, now instead of returning FE decryption of $\mathsf{FE.ct}$ (which results in evaluating $G[g, s, \mathsf{SKE.ct}, \mathsf{pad}]$ on the tuple encrypted under FE), use the PKE decryption of $\mathsf{PKE.ct}$ to help compute $G[g, s, \mathsf{SKE.ct}, \mathsf{pad}]$ directly. This step follows from the simulation soundness of the underlying NIZK.

1. $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.

2. Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ as follows:
   (a) $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
   (b) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
   (c) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (d) $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$.
   (e) $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.

3. Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0)$ as follows:
   (a) $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
   (b) $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (x_0, K, 0, \perp))$.
   (c) $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \perp)$.
   (d) $\pi^* \leftarrow \mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, z)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$:
   "$\mathsf{PKE.ct}^*$ *correctly encrypts* $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$*, where* $(x, K, \alpha, \widehat{\mathsf{sk}})$ *is encrypted in* $\mathsf{FE.ct}^*$"
   (e) Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.

4. $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$, where KeyGen and $\mathcal{O}$ are oracles defined as below:
   - $\mathsf{KeyGen}(\mathsf{msk}, f)$:
     (a) $s \leftarrow \{0,1\}^\lambda$.
     (b) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|f(\cdot)|}$ is randomly sampled.
     (c) $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.
     (d) Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.
   - $\mathcal{O}(\mathsf{msk}, \mathsf{CT}, g)$:
     (a) If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.
     (b) $s \leftarrow \{0,1\}^\lambda$.
     (c) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.
     (d) $\cancel{\mathsf{FE.sk}_{G_g} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[g, s, \mathsf{SKE.ct}])}$.
     (e) Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
     (f) If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.
     (g) Compute $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}') \leftarrow \mathsf{PKE.Dec}(\mathsf{PKE.sk}, \mathsf{PKE.ct})$, and return $G[g, s, \mathsf{SKE.ct}]$ $(x, K, \alpha, \widehat{\mathsf{sk}})$.

5. Let $\{f\}$ be the set of function queries by $\mathcal{A}_2$ to the KeyGen oracle. We require that the distributions $(\mathsf{mpk}, \mathsf{st}, \{f(x_0)\})$ and $(\mathsf{mpk}, \mathsf{st}, \{f(x_1)\})$ are statistically indistinguishable. If not, the experiment aborts and outputs 0.

6. If $b' = b$, the adversary wins and the game outputs 1. Otherwise, the adversary loses and the game outputs 0.

Hybrid$_4^{\mathcal{A}}$: When answering KeyGen queries, we now compute $r \leftarrow \mathsf{PRF.Eval}(K, s)$, and set $\mathsf{SKE.ct}$ to encrypt $\hat{y} \leftarrow f(x_0; r)$ instead of a randomly sampled y. This step follows from SKE security.

1. $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.

2. Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ as follows:

   (a) $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.

   (b) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.

   (c) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.

   (d) $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$.

   (e) $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.

3. Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0)$ as follows:

   (a) $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.

   (b) $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (x_0, K, 0, \perp))$.

   (c) $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \perp)$.

   (d) $\pi^* \leftarrow \mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, z)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$:

   "$\mathsf{PKE.ct}^*$ correctly encrypts $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$, where $(x, K, \alpha, \widehat{\mathsf{sk}})$ is encrypted in $\mathsf{FE.ct}^*$"

   (e) Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.

4. $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot), \mathcal{O}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$, where KeyGen and $\mathcal{O}$ are oracles defined as below:

   - $\mathsf{KeyGen}(\mathsf{msk}, f)$:

     (a) $s \leftarrow \{0,1\}^\lambda$.

     (b) $r \leftarrow \mathsf{PRF.Eval}(K, s)$.

     (c) $\hat{y} \leftarrow f(x_0; r)$.

     (d) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, \hat{y})$.

     (e) $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.

     (f) Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.

   - $\mathcal{O}(\mathsf{msk}, \mathsf{CT}, g)$:

     (a) If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.

     (b) $s \leftarrow \{0,1\}^\lambda$.

     (c) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.

     (d) Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.

     (e) If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.

     (f) Compute $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}') \leftarrow \mathsf{PKE.Dec}(\mathsf{PKE.sk}, \mathsf{PKE.ct})$, and return $G[g, s, \mathsf{SKE.ct}]$ $(x, K, \alpha, \widehat{\mathsf{sk}})$.

5. Let $\{f\}$ be the set of function queries by $\mathcal{A}_2$ to the KeyGen oracle. We require that the distributions $(\mathsf{mpk}, \mathsf{st}, \{f(x_0)\})$ and $(\mathsf{mpk}, \mathsf{st}, \{f(x_1)\})$ are statistically indistinguishable. If not, the experiment aborts and outputs 0.

6. If $b' = b$, the adversary wins and the game outputs 1. Otherwise, the adversary loses and the game outputs 0.

Hybrid$_5^{\mathcal{A}}$: Now we invoke FE selective security to encrypt $(\bot, \bot, 1, \mathsf{SKE.sk})$ instead in $\mathsf{FE.ct}^*$.

1. $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.
2. Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ as follows:
   (a) $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
   (b) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
   (c) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (d) $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$.
   (e) $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.
3. Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0)$ as follows:
   (a) $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
   (b) $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\bot, \bot, 1, \mathsf{SKE.sk}))$.
   (c) $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \bot)$.
   (d) $\pi^* \leftarrow \mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, z)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$:
   "$\mathsf{PKE.ct}^*$ correctly encrypts $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$, where $(x, K, \alpha, \widehat{\mathsf{sk}})$ is encrypted in $\mathsf{FE.ct}^*$"
   (e) Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.
4. $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$, where KeyGen and $\mathcal{O}$ are oracles defined as below:
   - $\mathsf{KeyGen}(\mathsf{msk}, f)$:
     (a) $s \leftarrow \{0,1\}^\lambda$.
     (b) $r \leftarrow \mathsf{PRF.Eval}(K, s)$.
     (c) $\hat{y} \leftarrow f(x_0; r)$.
     (d) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, \hat{y})$.
     (e) $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.
     (f) Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.
   - $\mathcal{O}(\mathsf{msk}, \mathsf{CT}, g)$:
     (a) If $\mathsf{CT} = \mathsf{CT}^*$, return $\bot$.
     (b) $s \leftarrow \{0,1\}^\lambda$.
     (c) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.
     (d) Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
     (e) If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\bot$.
     (f) Compute $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}') \leftarrow \mathsf{PKE.Dec}(\mathsf{PKE.sk}, \mathsf{PKE.ct})$, and return $G[g, s, \mathsf{SKE.ct}]$ $(x, K, \alpha, \widehat{\mathsf{sk}})$.
5. Let $\{f\}$ be the set of function queries by $\mathcal{A}_2$ to the KeyGen oracle. We require that the distributions $(\mathsf{mpk}, \mathsf{st}, \{f(x_0)\})$ and $(\mathsf{mpk}, \mathsf{st}, \{f(x_1)\})$ are statistically indistinguishable. If not, the experiment aborts and outputs 0.
6. If $b' = b$, the adversary wins and the game outputs 1. Otherwise, the adversary loses and the game outputs 0.

Hybrid$_6^{\mathcal{A}}$: When responding to KeyGen queries, now sample a uniformly random $\hat{r}$ instead of computing $r \leftarrow \mathsf{PRF.Eval}(K, s)$. This follows from PRF security, since the PRF key K is now completely hidden and unused.

1. $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.
2. Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ as follows:
   (a) $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
   (b) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
   (c) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (d) $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$.
   (e) $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.
3. Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0)$ as follows:
   (a) $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
   (b) $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\perp, \perp, 1, \mathsf{SKE.sk}))$.
   (c) $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \perp)$.
   (d) $\pi^* \leftarrow \mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, z)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$:
   "$\mathsf{PKE.ct}^*$ *correctly encrypts* $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$*, where* $(x, K, \alpha, \widehat{\mathsf{sk}})$ *is encrypted in* $\mathsf{FE.ct}^*$"
   (e) Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.
4. $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk}, \cdot), \mathcal{O}(\mathsf{msk}, \cdot, \cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$, where KeyGen and $\mathcal{O}$ are oracles defined as below:
   - $\mathsf{KeyGen}(\mathsf{msk}, f)$:
     (a) $s \leftarrow \{0,1\}^\lambda$.
     (b) $\hat{r} \leftarrow \{0,1\}^{\ell_{\mathsf{PRF}}}$.
     (c) $\hat{y} \leftarrow f(x_0; \hat{r})$.
     (d) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, \hat{y})$.
     (e) $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.
     (f) Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.
   - $\mathcal{O}(\mathsf{msk}, \mathsf{CT}, g)$:
     (a) If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.
     (b) $s \leftarrow \{0,1\}^\lambda$.
     (c) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.
     (d) Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
     (e) If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.
     (f) Compute $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}') \leftarrow \mathsf{PKE.Dec}(\mathsf{PKE.sk}, \mathsf{PKE.ct})$, and return $G[g, s, \mathsf{SKE.ct}]$ $(x, K, \alpha, \widehat{\mathsf{sk}})$.
5. Let $\{f\}$ be the set of function queries by $\mathcal{A}_2$ to the KeyGen oracle. We require that the distributions $(\mathsf{mpk}, \mathsf{st}, \{f(x_0)\})$ and $(\mathsf{mpk}, \mathsf{st}, \{f(x_1)\})$ are statistically indistinguishable. If not, the experiment aborts and outputs 0.

6. If $b' = b$, the adversary wins and the game outputs 1. Otherwise, the adversary loses and the game outputs 0.

Hybrid$_7^{\mathcal{A}}$: Now we change the challenge bit b from 0 to 1. This step follows from the fact that for all functions f queried by the adversary to the KeyGen oracle, $(\mathsf{mpk}, \mathsf{st}, \{f_(x_0)\})$ and $(\mathsf{mpk}, \mathsf{st}, \{f_(x_1)\})$ are statistically indistinguishable.

1. $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.
2. Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ as follows:
   (a) $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
   (b) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
   (c) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (d) $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$.
   (e) $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.
3. Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_1)$ as follows:
   (a) $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
   (b) $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\perp, \perp, 1, \mathsf{SKE.sk}))$.
   (c) $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \perp)$.
   (d) $\pi^* \leftarrow \mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, z)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$:
   "$\mathsf{PKE.ct}^*$ *correctly encrypts* $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$*, where* $(x, K, \alpha, \widehat{\mathsf{sk}})$ *is encrypted in* $\mathsf{FE.ct}^*$"
   (e) Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.
4. $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$, where KeyGen and $\mathcal{O}$ are oracles defined as below:
   - $\mathsf{KeyGen}(\mathsf{msk}, f)$:
     (a) $s \leftarrow \{0,1\}^\lambda$.
     (b) $\hat{r} \leftarrow \{0,1\}^{\ell_{\mathsf{PRF}}}$.
     (c) $\hat{y} \leftarrow f(x_1; \hat{r})$.
     (d) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, \hat{y})$.
     (e) $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.
     (f) Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.
   - $\mathcal{O}(\mathsf{msk}, \mathsf{CT}, g)$:
     (a) If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.
     (b) $s \leftarrow \{0,1\}^\lambda$.
     (c) $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.
     (d) Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
     (e) If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.
     (f) Compute $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}') \leftarrow \mathsf{PKE.Dec}(\mathsf{PKE.sk}, \mathsf{PKE.ct})$, and return $G[g, s, \mathsf{SKE.ct}]$ $(x, K, \alpha, \widehat{\mathsf{sk}})$.

5. Let $\{f\}$ be the set of function queries by $\mathcal{A}_2$ to the KeyGen oracle. We require that the distributions $(\mathsf{mpk}, \mathsf{st}, \{f(x_0)\})$ and $(\mathsf{mpk}, \mathsf{st}, \{f(x_1)\})$ are statistically indistinguishable. If not, the experiment aborts and outputs 0.
6. If $b' = b$, the adversary wins and the game outputs 1. Otherwise, the adversary loses and the game outputs 0.

$\mathbf{Hybrid}_8^{\mathcal{A}}$, $\mathbf{Hybrid}_9^{\mathcal{A}}$, $\mathbf{Hybrid}_{10}^{\mathcal{A}}$, $\mathbf{Hybrid}_{11}^{\mathcal{A}}$, $\mathbf{Hybrid}_{12}^{\mathcal{A}}$, and $\mathbf{Hybrid}_{13}^{\mathcal{A}}$ will revert the changes introduced in $\mathbf{Hybrid}_6^{\mathcal{A}}$, $\mathbf{Hybrid}_5^{\mathcal{A}}$, $\mathbf{Hybrid}_4^{\mathcal{A}}$, $\mathbf{Hybrid}_3^{\mathcal{A}}$, $\mathbf{Hybrid}_2^{\mathcal{A}}$ and $\mathbf{Hybrid}_1^{\mathcal{A}}$ respectively, with $\mathbf{Hybrid}_{13}^{\mathcal{A}}$ being the same as $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rFE}}$ where the challenge bit is fixed to be 1.

Proof of Hybrid Arguments

Lemma 5.3. *If* NIZK *has computational zero-knowledge, then no PPT adversary* $\mathcal{A}$ *can distinguish between* $\mathbf{Hybrid}_0^{\mathcal{A}}$ *and* $\mathbf{Hybrid}_1^{\mathcal{A}}$ *with non-negligible probability.*

Proof. We prove this by reduction to the zero-knowledgeness of NIZK. Specifically, we show how an adversary $\mathcal{A}$ that distinguishes between $\mathbf{Hybrid}_0^{\mathcal{A}}$ and $\mathbf{Hybrid}_1^{\mathcal{A}}$ can be used to construct an adversary $\mathcal{B}$ that breaks zero-knowledgeness of NIZK.

$\mathcal{B}^{\mathcal{O}_{\mathsf{NIZK}}(\cdot,\cdot)}(\mathsf{crs})$:

- $\mathcal{B}$ runs $\mathcal{A}_1(1^\lambda)$ to obtain (x_0, x_1, st).
- Sample $(\mathsf{mpk}, \mathsf{msk})$ as follows:
  - $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
  - $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
  - $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
  - $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \mathsf{crs}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \mathsf{crs}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.
- Compute CT^* as follows:
  - $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
  - $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (x_0, K, 0, \perp))$.
  - $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, (x_0, K, 0, \perp, \mathsf{FE.ct}^*))$.
  - Let z be the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$:
    "$\mathsf{PKE.ct}^*$ *correctly encrypts* $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$*, where* $(x, K, \alpha, \widehat{\mathsf{sk}})$ *is encrypted in* $\mathsf{FE.ct}^*$"
    and the witness w be the tuple $(x_0, K, 0, \perp)$. Query the oracle $\mathcal{O}_{\mathsf{NIZK}}$ with (z, w) and receive π^*.
  - Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.
- Run $\mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$ while simulating the following two oracles:
  - $\mathsf{KeyGen}(\mathsf{msk}, f)$:
    * $s \leftarrow \{0,1\}^\lambda$.
    * $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|f(\cdot)|}$ is randomly sampled.

- * $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.
  * Output $\mathsf{SK}_f = (\mathsf{crs}, \mathsf{FE.sk}_{G_f})$.
- $\mathcal{O}(\mathsf{msk}, \mathsf{CT}, g)$:
  * If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.
  * $s \leftarrow \{0,1\}^\lambda$.
  * $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.
  * $\mathsf{FE.sk}_{G_g} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[g, s, \mathsf{SKE.ct}])$.
  * Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
  * If $\mathsf{NIZK.Verify}(\mathsf{crs}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.
  * Return $\mathsf{FE.Dec}(\mathsf{FE.sk}_{G_g}, \mathsf{FE.ct})$.

- If $\mathcal{A}$ output it's in $\mathbf{Hybrid}_0^{\mathcal{A}}$, output 0. Otherwise, output 1.

Note that if $\mathcal{A}$ successfully distinguishes $\mathbf{Hybrid}_0^{\mathcal{A}}$ and $\mathbf{Hybrid}_1^{\mathcal{A}}$, then $\mathcal{B}$ also successfully distinguishes whether it's interacting with the real NIZK or the simulators $\mathsf{NIZK.Sim}_1$ and $\mathsf{NIZK.Sim}_2$. $\square$

Lemma 5.4. *If* PKE *has IND-CPA security, then no PPT adversary* $\mathcal{A}$ *can distinguish between* $\mathbf{Hybrid}_1^{\mathcal{A}}$ *and* $\mathbf{Hybrid}_2^{\mathcal{A}}$ *with non-negligible probability.*

Proof. We show this by a reduction to IND-CPA security of PKE. Specifically, we show that there exists PPT $\mathcal{A}$ that can distinguish between $\mathbf{Hybrid}_1^{\mathcal{A}}$ and $\mathbf{Hybrid}_2^{\mathcal{A}}$, then we can construct $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ using $\mathcal{A}$ as a subroutine that breaks IND-CPA security of PKE as follows:

$\mathcal{B}_1(\mathsf{PKE.pk})$:

- $\mathcal{B}_1$ runs $\mathcal{A}_1(1^\lambda)$ to obtain (x_0, x_1, st).
- Sample mpk and partial $\overline{\mathsf{msk}}$ as follows:
  - $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
  - $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
  - $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$.
  - Set $\mathsf{mpk} = (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}})$ and $\overline{\mathsf{msk}} = (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{FE.msk}, \mathsf{SKE.sk})$. Notice that $\overline{\mathsf{msk}}$ is missing $\mathsf{PKE.sk}$ compared to a normal msk.
- $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
- $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (x_0, K, 0, \perp))$.
- Output $((x_0, K, 0, \perp, \mathsf{FE.ct}^*), \perp, \mathsf{st}' = (\mathsf{st}, \mathsf{mpk}, \overline{\mathsf{msk}}, K, \mathsf{FE.ct}^*, \tau))$.

$\mathcal{B}_2(\mathsf{PKE.ct}^*, \mathsf{st}')$:

- Finish computing CT^* as follows:
  - $\widetilde{\pi}^* \leftarrow \mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, z)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$: "$\mathsf{PKE.ct}^*$ *correctly encrypts* $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$*, where* $(x, K, \alpha, \widehat{\mathsf{sk}})$ *is encrypted in* $\mathsf{FE.ct}^*$"
  - Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.

- Run $\mathcal{A}_2^{\mathsf{KeyGen}(\overline{\mathsf{msk}},\cdot),\mathcal{O}(\overline{\mathsf{msk}},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$ while simulating the following two oracles:
  - $\mathsf{KeyGen}(\overline{\mathsf{msk}}, f)$:
    * $s \leftarrow \{0,1\}^\lambda$.
    * $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|f(\cdot)|}$ is randomly sampled.
    * $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.
    * Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.
  - $\mathcal{O}(\overline{\mathsf{msk}}, \mathsf{CT}, g)$:
    * If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.
    * $s \leftarrow \{0,1\}^\lambda$.
    * $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.
    * $\mathsf{FE.sk}_{G_g} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[g, s, \mathsf{SKE.ct}])$.
    * Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
    * If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.
    * Return $\mathsf{FE.Dec}(\mathsf{FE.sk}_{G_g}, \mathsf{FE.ct})$.
- If $\mathcal{A}$ output it's in $\mathbf{Hybrid}_1^{\mathcal{A}}$, output 0. Otherwise, output 1.

Notice that if $\mathcal{A}$ outputs that it is in $\mathbf{Hybrid}_1^{\mathcal{A}}$, that means $\mathsf{PKE.ct}^*$ encrypts $(x, K, 0, \perp, \mathsf{FE.ct}^*)$, and hence $\mathcal{B}$ outputting 0 is correct. Otherwise, if $\mathcal{A}$ outputs it is in $\mathbf{Hybrid}_2^{\mathcal{A}}$ where $\mathsf{PKE.ct}^*$ encrypts $\perp$, $\mathcal{B}$ outputting 1 is also correct. $\square$

Lemma 5.5. *If* NIZK *has simulation soundness, then no PPT adversary $\mathcal{A}$ can distinguish between* $\mathbf{Hybrid}_2^{\mathcal{A}}$ *and* $\mathbf{Hybrid}_3^{\mathcal{A}}$ *with non-negligible probability.*

Proof. We prove this by reduction to the simulation soundness of NIZK. Let $\mathcal{A}$ be an adversary that distinguishes between $\mathbf{Hybrid}_2^{\mathcal{A}}$ and $\mathbf{Hybrid}_3^{\mathcal{A}}$, we construct an adversary $\mathcal{B}$ for the NIZK simulation soundness game as follows.

$\mathcal{B}^{\mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}},\tau,\cdot)}(\widetilde{\mathsf{crs}})$:

- Runs $\mathcal{A}_1(1^\lambda)$ to obtain (x_0, x_1, st).
- Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ as follows:
  1. $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
  2. $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
  3. $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
  4. $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.
- Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0)$ as follows:
  1. $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
  2. $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (x_0, K, 0, \perp))$.
  3. $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \perp)$.

4. Query the oracle $\mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, \cdot)$ on the following statement z on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$ to obtain $\widetilde{\pi}^*$:

   " $\mathsf{PKE.ct}^*$ *correctly encrypts* $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$*, where* $(x, K, \alpha, \widehat{\mathsf{sk}})$ *is encrypted in* $\mathsf{FE.ct}^*$*"*

5. Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.

- Sample a uniform bit $b \leftarrow \{0, 1\}$.
- Run $b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$ while simulating the following two oracles:
  - $\mathsf{KeyGen}(\mathsf{msk}, f)$:
    1. $s \leftarrow \{0, 1\}^\lambda$.
    2. $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0, 1\}^{|f(\cdot)|}$ is randomly sampled.
    3. $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.
    4. Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.
  - $\mathcal{O}(\mathsf{msk}, \mathsf{CT}, g)$:
    1. If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.
    2. $s \leftarrow \{0, 1\}^\lambda$.
    3. $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0, 1\}^{|g(\cdot)|}$ is randomly sampled.
    4. $\mathsf{FE.sk}_{G_g} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[g, s, \mathsf{SKE.ct}])$.
    5. Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
    6. If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.
    7. Compute $(x, K, \alpha, \widehat{\mathsf{sk}}) \leftarrow \mathsf{FE.Dec}(\mathsf{FE.sk}_{\mathcal{I}}, \mathsf{FE.ct})$ by sampling an FE functional key for the identity function $\mathcal{I}$ as $\mathsf{FE.sk}_{\mathcal{I}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, \mathcal{I})$.
    8. Compute $(x', K', \alpha', \widehat{\mathsf{sk}}', \mathsf{FE.ct}') \leftarrow \mathsf{PKE.Dec}(\mathsf{PKE.sk}, \mathsf{PKE.ct})$.
    9. If $(x, K, \alpha, \widehat{\mathsf{sk}}) \neq (x', K', \alpha', \widehat{\mathsf{sk}}')$, $\mathcal{B}$ outputs $((\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi)$ to the NIZK simulation soundness game (and continues finishing simulating the oracles for $\mathcal{A}_2$).
    10. If $b = 0$, return $\mathsf{FE.Dec}(\mathsf{SK}_{G_g}, \mathsf{FE.ct})$. If $b = 1$, compute and return $G[g, s, \mathsf{SKE.ct}]$ $(x', K', \alpha', \widehat{\mathsf{sk}}')$.

Kindly note that $b = 0$ corresponds to the case where $\mathcal{A}$ is in $\mathbf{Hybrid}_2^{\mathcal{A}}$ and $b = 1$ corresponds to $\mathbf{Hybrid}_3^{\mathcal{A}}$.

Now, notice that if $\mathcal{B}$ ever outputs $((\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi)$ when answering a $\mathcal{O}$ query (CT, g), it *will* win the NIZK simulation soundness game. Recall that winning the simulation soundness game requires: (1) $((\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi)$ is not returned by a prior query to $\mathsf{NIZK.Sim}_2$; (2) $(\mathsf{FE.ct}, \mathsf{PKE.ct})$ is not in the language; (3) $\mathsf{Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi)$ outputs 1. (1) is true because $\mathsf{CT} \neq \mathsf{CT}^*$, otherwise $\mathcal{O}$ will return $\perp$ earlier on in Step 1; (2) is true since $\mathcal{B}$ will only output if $\mathsf{FE.ct}$ and $\mathsf{PKE.ct}$ encrypt different $(x, K, \alpha, \widehat{\mathsf{sk}})$ tuples and hence not in the language; and (3) is true because otherwise $\mathcal{O}$ will return $\perp$ in Step 6. So indeed if $\mathcal{B}$ ever outputs $((\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi)$, $\mathcal{B}$ will win the simulation soundness game.

So if $\mathcal{B}$ does not win, that means it never outputs anything. This implies that for all the $\mathcal{O}$ queries made by $\mathcal{A}_2$, the response is either $\perp$, or $\mathsf{FE.ct}$ and $\mathsf{PKE.ct}$ encrypt the exact same $(x, K, \alpha, \widehat{\mathsf{sk}})$ tuple. Observe that if $\mathsf{FE.ct}$ and $\mathsf{PKE.ct}$ encrypt the same $(x, K, \alpha, \widehat{\mathsf{sk}})$ tuple, then for $b = 0$, $\mathcal{O}$ would respond with $\mathsf{FE.Dec}(\mathsf{SK}_{G_g}, \mathsf{FE.ct}) = G[g, s, \mathsf{SKE.ct}](x, K, \alpha, \widehat{\mathsf{sk}})$, which is exactly the same as what the oracle response would be for $b = 1$. This means, if $\mathcal{B}$ does not win, then for all $\mathcal{O}$ queries, the responses will be exactly the same *regardless of the challenge bit* b, and therefore $\mathcal{A}$ cannot possibly guess b correctly with non-negligible advantage. By contrapositive, if $\mathcal{A}$ can successfully distinguish $\mathbf{Hybrid}_2^{\mathcal{A}}$ and $\mathbf{Hybrid}_3^{\mathcal{A}}$ by guessing the bit b correctly, $\mathcal{B}$ would also win the simulation soundness game for NIZK.

Lastly, kindly observe that step 4d is no longer needed once we switch to $\mathbf{Hybrid}_3^{\mathcal{A}}$. $\square$

Lemma 5.6. *If* SKE *is a IND-CPA secure SKE scheme, then no PPT adversary* $\mathcal{A}$ *can distinguish between* $\mathbf{Hybrid}_3^{\mathcal{A}}$ *and* $\mathbf{Hybrid}_4^{\mathcal{A}}$ *with non-negligible probability.*

Proof. Notice that the only difference between $\mathbf{Hybrid}_3^{\mathcal{A}}$ and $\mathbf{Hybrid}_4^{\mathcal{A}}$ is that in $\mathbf{Hybrid}_3^{\mathcal{A}}$ $\mathsf{SKE.ct}$ encrypts a random y, while in $\mathbf{Hybrid}_4^{\mathcal{A}}$, $\mathsf{SKE.ct}$ encrypts $\hat{y}$ that are function evaluation on x_0 and PRF outputs. This easily reduces to the left-or-right CPA security of the underlying SKE.

For completeness, we show a reduction by building an adversary $\mathcal{B}$ that breaks the CPA security of SKE by using $\mathcal{A}$ as a subroutine.

$\mathcal{B}(1^\lambda)$:

- $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.
- Sample mpk and partial $\overline{\mathsf{msk}}$ as follows:
  1. $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
  2. $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
  3. $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$.
  4. $(\mathsf{mpk}, \overline{\mathsf{msk}}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{FE.msk}, \mathsf{PKE.sk}))$, where $\overline{\mathsf{msk}}$ is the normal msk but missing the $\mathsf{SKE.sk}$ part.
- Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0)$ as follows:
  1. $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
  2. $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (x_0, K, 0, \perp))$.
  3. $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \perp)$.
  4. $\pi^* \leftarrow \mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, z)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$: "$\mathsf{PKE.ct}^*$ *correctly encrypts* $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$, *where* $(x, K, \alpha, \widehat{\mathsf{sk}})$ *is encrypted in* $\mathsf{FE.ct}^*$"
  5. Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.
- Run $\mathcal{A}_2^{\mathsf{KeyGen}(\overline{\mathsf{msk}},\cdot),\mathcal{O}(\overline{\mathsf{msk}},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$, while simulating the following two oracles:
  - $\mathsf{KeyGen}(\overline{\mathsf{msk}}, f)$:
    1. $s \leftarrow \{0,1\}^\lambda$.
    2. $r \leftarrow \mathsf{PRF.Eval}(K, s)$.
    3. Sample/compute $y_0 \leftarrow \{0,1\}^{|f(\cdot)|}$ and $y_1 \leftarrow f(x_0; r)$.
    4. Submit challenge messages (y_0, y_1) and receive a challenge ciphertext $\mathsf{SKE.ct}$.
    5. $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.
    6. Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.

– $\mathcal{O}(\overline{\mathsf{msk}}, \mathsf{CT}, g)$:

1. If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.
2. $s \leftarrow \{0,1\}^\lambda$.
3. Sample $y_0, y_1 \leftarrow \{0,1\}^{|g(\cdot)|}$.
4. Submit challenge messages (y_0, y_1) and receive a challenge ciphertext $\mathsf{SKE.ct}$.
5. Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
6. If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.
7. Compute $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}') \leftarrow \mathsf{PKE.Dec}(\mathsf{PKE.sk}, \mathsf{PKE.ct})$, and return $G[g, s, \mathsf{SKE.ct}](x, K, \alpha, \widehat{\mathsf{sk}})$.

- If $\mathcal{A}$ outputs it is in $\mathbf{Hybrid}_3^{\mathcal{A}}$, output 0. Otherwise, output 1.

It is easy to see that if $\mathcal{A}$ wins, $\mathcal{B}$ also wins. □

Lemma 5.7. *If* FE *is a selectively secure FE scheme, then no PPT adversary $\mathcal{A}$ can distinguish between $\mathbf{Hybrid}_4^{\mathcal{A}}$ and $\mathbf{Hybrid}_5^{\mathcal{A}}$ with non-negligible probability.*

Proof. We show a reduction to selective FE security. Let $\mathcal{A}$ be an adversary that can distinguish between $\mathbf{Hybrid}_4^{\mathcal{A}}$ and $\mathbf{Hybrid}_5^{\mathcal{A}}$, we construct an adversary $\mathcal{B}$ that can break selective security of the underlying FE scheme.

$\mathcal{B}(1^\lambda)$:

- $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.
- Sample partial $\overline{\mathsf{mpk}}$ and partial $\overline{\mathsf{msk}}$ as follows:
  1. $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
  2. $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
  3. $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$.
  4. Set $(\overline{\mathsf{mpk}}, \overline{\mathsf{msk}}) = ((\mathsf{PKE.pk}, \widetilde{\mathsf{crs}}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$, where $\overline{\mathsf{mpk}}$ and $\overline{\mathsf{msk}}$ are the normal mpk and msk but missing the $\mathsf{FE.mpk}$ and $\mathsf{FE.msk}$ part.
- Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\overline{\mathsf{mpk}}, x_0)$ as follows:
  1. $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
  2. Submit challenge messages $m_0 = (x_0, K, 0, \perp)$ and $m_1 = (\perp, \perp, 1, \mathsf{SKE.sk})$, and receive a challenge ciphertext $\mathsf{FE.ct}^*$ together with $\mathsf{FE.mpk}$.
  3. $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \perp)$.
  4. $\pi^* \leftarrow \mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, z)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$: "$\mathsf{PKE.ct}^*$ *correctly encrypts* $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$*, where* $(x, K, \alpha, \widehat{\mathsf{sk}})$ *is encrypted in* $\mathsf{FE.ct}^*$"
  5. Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.
- Run $\mathcal{A}_2^{\mathsf{KeyGen}(\overline{\mathsf{msk}}, \cdot), \mathcal{O}(\overline{\mathsf{msk}}, \cdot, \cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$, , while simulating the following two oracles:

  – $\mathsf{KeyGen}(\overline{\mathsf{msk}}, f)$:

1. $s \leftarrow \{0,1\}^\lambda$.
2. $r \leftarrow \mathsf{PRF.Eval}(K, s)$.
3. $\hat{y} \leftarrow f(x_0; r)$.
4. $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, \hat{y})$.
5. Submit a function query for $G[f, s, \mathsf{SKE.ct}]$ and receive $\mathsf{FE.sk}_{G_f}$.
6. Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.

– $\mathcal{O}(\overline{\mathsf{msk}}, \mathsf{CT}, g)$:

1. If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.
2. $s \leftarrow \{0,1\}^\lambda$.
3. $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.
4. Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
5. If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.
6. Compute $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}') \leftarrow \mathsf{PKE.Dec}(\mathsf{PKE.sk}, \mathsf{PKE.ct})$, and return $G[g, s, \mathsf{SKE.ct}](x, K, \alpha, \widehat{\mathsf{sk}})$.

- If $\mathcal{A}$ outputs it is in $\mathbf{Hybrid}_4^{\mathcal{A}}$, output 0. Otherwise, output 1.

First, notice that the only function queries $\mathcal{B}$ needs to submit are for $G[f, s, \mathsf{SKE.ct}]$, where we have

$$G[f, s, \mathsf{SKE.ct}](x_0, K, 0, \perp) = f(x_0; r) = G[f, s, \mathsf{SKE.ct}](\perp, \perp, 1, \mathsf{SKE.sk}),$$

so all the function queries are valid.

Then, it is easy to verify that if $\mathcal{A}$ wins, $\mathcal{B}$ also wins. □

Lemma 5.8. *If* PRF *is a secure PRF, then no PPT adversary* $\mathcal{A}$ *can distinguish between* $\mathbf{Hybrid}_5^{\mathcal{A}}$ *and* $\mathbf{Hybrid}_6^{\mathcal{A}}$ *with non-negligible probability.*

Proof. We construct the following adversary $\mathcal{B}$ for the $\mathsf{Expt}_{\mathcal{B}}^{\mathsf{PRF}}$ game by using $\mathcal{A}$ as a subroutine.

$\mathcal{B}(1^\lambda)$:

- $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$.
- Sample $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ as follows:
  1. $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
  2. $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
  3. $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
  4. $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Sim}_1(1^\lambda)$.
  5. $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}), (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \widetilde{\mathsf{crs}}, \mathsf{FE.msk}, \mathsf{PKE.sk}, \mathsf{SKE.sk}))$.
- Compute $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_0)$ as follows:
  1. $\mathsf{FE.ct}^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\perp, \perp, 1, \mathsf{SKE.sk}))$.
  2. $\mathsf{PKE.ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \perp)$.

3. $\pi^* \leftarrow \mathsf{NIZK.Sim}_2(\widetilde{\mathsf{crs}}, \tau, z)$ where z is the following statement on $\mathsf{FE.ct}^*$ and $\mathsf{PKE.ct}^*$: "*$\mathsf{PKE.ct}^*$ correctly encrypts $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}^*)$, where $(x, K, \alpha, \widehat{\mathsf{sk}})$ is encrypted in $\mathsf{FE.ct}^*$*"
4. Set $\mathsf{CT}^* = (\mathsf{FE.ct}^*, \mathsf{PKE.ct}^*, \pi^*)$.

- Run $\mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}(\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{CT}^*, \mathsf{st})$, while simulating the following two oracles:
  - $\mathsf{KeyGen}(\mathsf{msk}, f)$:
    1. $s \leftarrow \{0,1\}^\lambda$.
    2. Submit s and receive r.
    3. $\hat{y} \leftarrow f(x_0; r)$.
    4. $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, \hat{y})$.
    5. $\mathsf{FE.sk}_{G_f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G[f, s, \mathsf{SKE.ct}])$.
    6. Output $\mathsf{SK}_f = (\widetilde{\mathsf{crs}}, \mathsf{FE.sk}_{G_f})$.
  - $\mathcal{O}(\mathsf{msk}, \mathsf{CT}, g)$:
    1. If $\mathsf{CT} = \mathsf{CT}^*$, return $\perp$.
    2. $s \leftarrow \{0,1\}^\lambda$.
    3. $\mathsf{SKE.ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, y)$, where $y \leftarrow \{0,1\}^{|g(\cdot)|}$ is randomly sampled.
    4. Parse $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.
    5. If $\mathsf{NIZK.Verify}(\widetilde{\mathsf{crs}}, (\mathsf{FE.ct}, \mathsf{PKE.ct}), \pi) = 0$, return $\perp$.
    6. Compute $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct}') \leftarrow \mathsf{PKE.Dec}(\mathsf{PKE.sk}, \mathsf{PKE.ct})$, and return $G[g, s, \mathsf{SKE.ct}](x, K, \alpha, \widehat{\mathsf{sk}})$.
- If $\mathcal{A}$ outputs it is in $\mathbf{Hybrid}_5^{\mathcal{A}}$, output 0. Otherwise, output 1.

Note that the r $\mathcal{B}$ receives will either be $\mathsf{PRF.Eval}(K, s)$ or a uniformly random value, depending on the challenge bit. And these cases correspond exactly to $\mathbf{Hybrid}_5^{\mathcal{A}}$ and $\mathbf{Hybrid}_6^{\mathcal{A}}$. It's easy to see that if $\mathcal{A}$ wins, $\mathcal{B}$ also wins. □

Lemma 5.9. *No adversary $\mathcal{A}$ can distinguish between $\mathbf{Hybrid}_6^{\mathcal{A}}$ and $\mathbf{Hybrid}_7^{\mathcal{A}}$ with non-negligible probability.*

Proof. Note that the only difference between $\mathbf{Hybrid}_6^{\mathcal{A}}$ and $\mathbf{Hybrid}_7^{\mathcal{A}}$ is that we switch from $f(x_0; \hat{r})$ to $f(x_1; \hat{r})$, where $\hat{r}$ is a uniformly random value. Recall that the experiment requires that for all f queried by $\mathcal{A}_2$ to the KeyGen oracle, the distributions of $f(x_0)$ and $f(x_1)$ are statistically indistinguishable. Therefore, no adversary can distinguish between them with non-negligible probability as desired. □

The proofs of the hybrid arguments for $\mathbf{Hybrid}_7^{\mathcal{A}}$ to $\mathbf{Hybrid}_{13}^{\mathcal{A}}$ follow analogously from the lemmas above, yielding us the final theorem result.

Theorem 5.2. *If PKE is a CPA-secure PKE, SKE is a CPA-secure SKE, NIZK is a NIZK with simulation soundness, FE is a selectively-secure FE scheme, and PRF is a secure PRF, then Construction 1 is IND-secure per Definition 5.1.*

Proof. The lemmas above show a sequence of a polynomial number of hybrid experiments where no PPT adversary can distinguish one from the next with non-negligible probability. The first hybrid $\mathbf{Hybrid}_0^{\mathcal{A}}$ corresponds to the $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{rFE}}$ game where $b = 0$, and the last one $\mathbf{Hybrid}_{13}^{\mathcal{A}}$ corresponds to the case where $b = 1$. The security of the indistinguishability game follows. □

5.4 Issue with Counterexample Construction (Construction 1

Now we highlight why construction 1 is problematic against a malicious encryptor. The high level intuition is that a malicious encryptor can cause the randomized function evaluation to use a randomness of its own choice.

Specifically, let $\mathsf{PRF}' = (\mathsf{Setup}, \mathsf{Eval})$ be a secure PRF, and consider the following PRF construction $\mathsf{PRF} = (\mathsf{Setup}, \mathsf{Eval})$:

- $\mathsf{Setup}(1^\lambda)$: Compute $K' \leftarrow \mathsf{PRF}'.\mathsf{Setup}(1^\lambda)$, and output $K = (K', 0, \perp)$.
- $\mathsf{Eval}(K, s)$: First, parse $K = (K', b, r)$. If $b = 0$, output $\mathsf{PRF}'.\mathsf{Eval}(K', s)$. Otherwise, output r.

Essentially, it is a PRF with a "trapdoor" mode. Normally, when $b = 0$, it behaves like the underlying PRF'. But when $b = 1$, it completely ignores the seed s, and simply outputs some hardcoded randomness r. Notice that by security of PRF', the construction PRF is also secure.

Now imagine using this PRF construction as the PRF in Construction 1. Here is what a malicious encryptor $\mathcal{A}$ can do:

Malicious Encryptor $\mathcal{A}(\mathsf{mpk}, x)$:

1. Parse $\mathsf{mpk} = (\mathsf{FE.mpk}, \mathsf{PKE.pk}, \mathsf{crs})$.
2. $K' \leftarrow \mathsf{PRF}'.\mathsf{Setup}(1^\lambda)$.
3. Pick any preferred randomness r, and set $K = (K', 1, r)$.
4. $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (x, K, 0, \perp))$.
5. $\mathsf{PKE.ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, (x, K, 0, \perp, \mathsf{FE.ct}))$.
6. $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, z, w)$ where z is the statement that $\mathsf{PKE.ct}$ correctly encrypts $(x, K, \alpha, \widehat{\mathsf{sk}}, \mathsf{FE.ct})$, where $(x, K, \alpha, \widehat{\mathsf{sk}})$ is encrypted in $\mathsf{FE.ct}$.
7. Output $\mathsf{CT} = (\mathsf{FE.ct}, \mathsf{PKE.ct}, \pi)$.

Notice that decrypting this malicious CT yields $G[f, s, \mathsf{SKE.ct}](x, K, 0, \perp)$. But in this case $\mathsf{PRF.Eval}$ simply outputs the hardcoded r which is arbitrarily chosen by the malicious encryptor. In other words, the malicious encryptor can set the randomness used to evaluate the function f to any favorful value it wants.

6 Constructing Adaptively Secure rMIFE

In this section we present our construction of rMIFE. It is inspired by [GJO16], and we build upon the adaptively secure deterministic MIFE construction of Goldwasser et al. [GGG+14].

For our construction, we utilize a subexponentially-secure indistinguishability obfuscation ($i\mathcal{O}$), a plain PKE, a puncturable PRF and injective One Way Functions (OWFs), with parameters specified in the following subsection. The definitions of these tools can be found in Section 3 of the supplementary material.

6.1 Parameters

- PKE
  - Security parameter λ.
  - Ciphertexts of length s on inputs x_i.
- $i\mathcal{O} = i\mathcal{O}$ with $(1, 2^{-3ns-\lambda_{i\mathcal{O}}})$ weak extractability. This means for any two equivalent circuits, the security gap of the obfuscation is bounded by $2^{-3ns-\lambda_{i\mathcal{O}}}$ (any algorithm that distinguishes obfuscations of two circuits with more than this gap can be used to extract a differing point).
  - Security parameter $\lambda_{i\mathcal{O}} = \lambda$.
- $\mathsf{PRF1}$ = Puncturable PRF with security $2^{-\lambda_{\mathsf{PRF1}}^{c_{\mathsf{PRF1}}}}$. This is the PRF using K_i as keys.
  - Security parameter $\lambda_{\mathsf{PRF1}} > (2ns + \lambda_{i\mathcal{O}})^{(1/c_{\mathsf{PRF1}})}$.
- $\mathsf{PRF2}$ = Puncturable PRF with security $2^{-\lambda_{\mathsf{PRF2}}^{c_{\mathsf{PRF2}}}}$. This is the PRF using K_f as keys.
  - Security parameter $\lambda_{\mathsf{PRF2}} > (2ns + \lambda_{i\mathcal{O}})^{(1/c_{\mathsf{PRF2}})}$.
- InjOWF = Injective OWF with security $2^{-\lambda_{\mathsf{OWF}}^{c_{\mathsf{OWF}}}}$.
  - Security parameter $\lambda_{\mathsf{OWF}} > (3ns + \lambda_{i\mathcal{O}})^{1/c_{\mathsf{OWF}}}$.
  - Output length $> \max\{(5ns + 2\lambda_{i\mathcal{O}})^{1/c_{\mathsf{OWF}}}, (3ns + \lambda_{i\mathcal{O}})^{1/c_{\mathsf{PRF1}}}, (3ns + \lambda_{i\mathcal{O}})^{1/c_{\mathsf{PRF2}}}\}$.

6.2 Construction

Now we present our main construction of rMIFE.

Construction 2. Let λ be the security parameter and n be the number of inputs. Let $i\mathcal{O}$ be an indistinguishability obfuscation scheme, $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ be a plain model PKE, $\mathsf{PRF1}, \mathsf{PRF2} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Punc})$ be puncturable PRFs, InjOWF be an injective OWF. We construct our $\mathsf{rMIFE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ as follows:

- $\mathsf{Setup}(1^\lambda, n)$:
  1. For $i \in [n]$,
     (a) $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
     (b) For $b \in \{0,1\}$, $(\mathsf{pk}_i^b, \mathsf{sk}_i^b) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
     (c) $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i])$.
     (d) $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.
  2. $\mathsf{MSK} = \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i\}_{i\in[n]}$.
  3. Output $(\mathsf{MSK}, \{\mathsf{EK}_i\}_{i\in[n]})$.

$E_i[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i](c_i^0, c_i^1, x_i, r_i^0, r_i^1)$:

1. For $b \in \{0,1\}$, if $c_i^b \neq \mathsf{PKE.Enc}(\mathsf{pk}_i^b, x_i; r_i^b)$, output $\perp$.
2. Output $z_i = \mathsf{PRF1.Eval}(K_i, (c_i^0, c_i^1))$.

- $\mathsf{Enc}(\mathsf{EK}_i, x_i)$:
  1. Parse $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.
  2. $r_i^0, r_i^1 \leftarrow \{0,1\}^\lambda$.
  3. For $b \in \{0,1\}$, $c_i^b = \mathsf{PKE.Enc}(\mathsf{pk}_i^b, x_i; r_i^b)$.
  4. $z_i = \widetilde{E}_i(c_i^0, c_i^1, x_i, r_i^0, r_i^1)$.
  5. Output $\mathsf{CT}_i = (c_i^0, c_i^1, z_i)$.
- $\mathsf{KeyGen}(\mathsf{MSK}, f)$:
  1. Parse $\mathsf{MSK} = (\{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i\}_{i\in[n]})$.
  2. Sample $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
  3. $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G[f, \{\mathsf{sk}_i^0, K_i\}_{i\in[n]}, K_f, \mathsf{InjOWF}])$.
  4. Output $\mathsf{SK}_f = \widetilde{G}_f$.

$G[f, \{\mathsf{sk}_i^0, K_i\}_{i\in[n]}, K_f, \mathsf{InjOWF}]((c_i^0, c_i^1, z_i)_{i\in[n]})$:

1. For $i \in [n]$,
   (a) If $\mathsf{InjOWF}(z_i) \neq \mathsf{InjOWF}(\mathsf{PRF1.Eval}(K_i, c_i^0, c_i^1))$, output $\perp$.
   (b) $x_i = \mathsf{PKE.Dec}(\mathsf{sk}_i^0, c_i^0)$.
2. $r_f = \mathsf{PRF2.Eval}(K_f, (c_i^0, c_i^1)_{i\in[n]})$.
3. Output $y = f(x_1, x_2, \ldots, x_n; r_f)$.

- $\mathsf{Dec}(\mathsf{SK}_f, \{\mathsf{CT}_i\}_{i\in[n]})$.
  1. Parse $\mathsf{SK}_f = \widetilde{G}_f$, and for $i \in [n]$, parse $\mathsf{CT}_i = (c_i^0, c_i^1, z_i)$.
  2. Output $y = \widetilde{G}_f(\{c_i^0, c_i^1, z_i\}_{i\in[n]})$.

Correctness follows from correctness of the underlying schemes. We next argue our construction is secure against both malicious decryptors and malicious encryptors.

6.3 Security against malicious decryptors

We first prove security against malicious decryptors.

Theorem 6.1. *If* PKE *is CPA-secure, $i\mathcal{O}$ is a subexponentially-secure indistinguishable obfuscation scheme,* PRF*s and* InjOWF *are secure with the parameters outlines in the Parameters section, then Construction 2 is IND secure against malicious decryptors for $\epsilon = 2^{-2ns-\lambda_{i\mathcal{O}}}$-distinguishable distributions.*

We prove this through a sequence of hybrids.

Sequence of Hybrids

Hybrid$_0^{\mathcal{A}}(1^\lambda)$: Real world experiment with $b = 0$.

1. **Setup:**

   (a) For $i \in [n]$,

      i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.

      ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.

      iii. $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i])$.

      iv. $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.

   - **Function Query:**

     (a) $\mathcal{A}$ outputs a function f_ℓ.

     (b) $K_{f_\ell} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.

     (c) $\widetilde{G}_{f_\ell} \leftarrow i\mathcal{O}(1^\lambda, G[f_\ell, \{\mathsf{sk}_i^0, K_i\}_{i\in[n]}, K_{f_\ell}, \mathsf{InjOWF}])$.

     (d) Send $\mathsf{SK}_{f_\ell} = \widetilde{G}_{f_\ell}$ to $\mathcal{A}$.

   - **Encryption Key Query:**

     (a) $\mathcal{A}$ outputs an index $i \in [n]$.

     (b) Send EK_i to $\mathcal{A}$.

   - **Challenge Message Query:**

     (a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.

     (b) For $i \in [n]$,

        i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.

        ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.

        iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^0; r_{j,i}^1)$.

        iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.

        v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.

     (c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

Hybrid$_1^{\mathcal{A}}(1^\lambda)$: We encrypt $x_{j,i}^1$ into $c_{j,i}^1$, that is we compute $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$. This follows by PKE security.

1. **Setup:**

   (a) For $i \in [n]$,

      i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.

      ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.

      iii. $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i])$.

      iv. $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.

- **Function Query:**
  (a) $\mathcal{A}$ outputs a function f_ℓ.
  (b) $K_{f_\ell} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
  (c) $\widetilde{G}_{f_\ell} \leftarrow i\mathcal{O}(1^\lambda, G[f_\ell, \{\mathsf{sk}_i^0, K_i\}_{i\in[n]}, K_{f_\ell}, \mathsf{InjOWF}])$.
  (d) Send $\mathsf{SK}_{f_\ell} = \widetilde{G}_{f_\ell}$ to $\mathcal{A}$.
- **Encryption Key Query:**
  (a) $\mathcal{A}$ outputs an index $i \in [n]$.
  (b) Send EK_i to $\mathcal{A}$.
- **Challenge Message Query:**
  (a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.
  (b) For $i \in [n]$,
    i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.
    ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.
    iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$.
    iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.
    v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.
  (c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

Hybrid$_{2,w,0}^{\mathcal{A}}(1^\lambda)_{w\in[0,2^{2sn}]}$: Here s denotes the length of the PKE ciphertexts. In this sequence of hybrids, we consider only one secret key query from the adversary for some function f. Multiple key queries can be handled via a standard hybrid argument. While generating the secret key queried by the adversary we construct the program G_f, by switching to computing x_i using sk_i^1 and c_i^1 for all inputs with $(c_i^0, c_i^1)_{i\in[n]} < w$. For $w = 0$, this is indistinguishable from the previous hybrid by $i\mathcal{O}$ since the program G in the two hybrids is exactly identical in functionality.

1. **Setup:**
   (a) For $i \in [n]$,
     i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
     ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
     iii. $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i])$.
     iv. $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.
2. $\mathcal{A}$ may make any number of the following queries in any order.
   - **Function Query:**
     (a) $\mathcal{A}$ outputs a function f.
     (b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
     (c) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G'[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i\}_{i\in[n]}, w, K_f, \mathsf{InjOWF}])$.
     (d) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.
   - **Encryption Key Query:**
     (a) $\mathcal{A}$ outputs an index $i \in [n]$.

(b) Send EK_i to $\mathcal{A}$.

- **Challenge Message Query:**

(a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.

(b) For $i \in [n]$,

i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.

ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.

iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$.

iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.

v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.

(c) Send $\{\mathsf{CT}_{j,i}\}_{i \in [n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

$G'[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i\}_{i \in [n]}, w, K_f, \mathsf{InjOWF}]((c_i^0, c_i^1, z_i)_{i \in [n]})$:

1. For $i \in [n]$,

(a) If $\mathsf{InjOWF}(z_i) \neq \mathsf{InjOWF}(\mathsf{PRF1.Eval}(K_i, c_i^0, c_i^1))$, output $\perp$.

(b) If $(c_i^0, c_i^1)_{i \in [n]} \geq w$, $x_i = \mathsf{PKE.Dec}(\mathsf{sk}_i^0, c_i^0)$.

(c) If $(c_i^0, c_i^1)_{i \in [n]} < w$, $x_i = \mathsf{PKE.Dec}(\mathsf{sk}_i^1, c_i^1)$.

2. $r_f = \mathsf{PRF2.Eval}(K_f, (c_i^0, c_i^1)_{i \in [n]})$.

3. Output $y = f(x_1, x_2, \ldots, x_n; r_f)$.

Hybrid$_{2,w,1}^{\mathcal{A}}(1^\lambda)_{w \in [0, 2^{2sn}]}$: Puncture the $\mathsf{PRF1}$ keys K_i at values associated with w.

1. **Setup:**

(a) For $i \in [n]$,

i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.

ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.

(b) **Compute Values for w:**

i. Parse $w = (d_i^0, d_i^1)_{i \in [n]}$.

ii. For $i \in [n]$, $\alpha \in \{0,1\}$, $x_i^\alpha = \mathsf{PKE.Dec}(\mathsf{sk}_i^\alpha, d_i^\alpha)$.

iii. Define $A_w = \{i \in [n] : x_i^0 = x_i^1\}$.

iv. For $i \in A_w, u_i = \mathsf{PRF1.Eval}(K_i, (d_i^0, d_i^1))$.

v. For $i \in [n], K_i[d_i^0, d_i^1] = \mathsf{PRF1.Punc}(K_i, (d_i^0, d_i^1))$.

(c) Compute:

i. For $i \in A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i'[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1, u_i])$.

ii. For $i \in [n] \setminus A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i''[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1])$.

(d) For $i \in [n]$, set $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.

- **Function Query:**
  (a) $\mathcal{A}$ outputs a function f.
  (b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
  (c) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G''[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1\}_{i\in[n]}, \{\mathsf{InjOWF}(u_i)\}_{i\in A_w}, w, K_f, \mathsf{InjOWF}])$.
  (d) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.
- **Encryption Key Query:**
  (a) $\mathcal{A}$ outputs an index $i \in [n]$.
  (b) Send EK_i to $\mathcal{A}$.
- **Challenge Message Query:**
  (a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.
  (b) For $i \in [n]$,
    i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.
    ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.
    iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$.
    iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.
    v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.
  (c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

$E_i'[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1, u_i](c_i^0, c_i^1, x_i, r_i^0, r_i^1)$:

1. For $b \in \{0,1\}$, if $c_i^b \neq \mathsf{PKE.Enc}(\mathsf{pk}_i^b, x_i; r_i^b)$, output $\perp$.
2. If $(c_i^0, c_i^1) = (d_i^0, d_i^1)$, output u_i.
3. Else, output $z_i = \mathsf{PRF1.Eval}(K_i[d_i^0, d_i^1], (c_i^0, c_i^1))$.

$E_i''[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1](c_i^0, c_i^1, x_i, r_i^0, r_i^1)$:

1. For $b \in \{0,1\}$, if $c_i^b \neq \mathsf{PKE.Enc}(\mathsf{pk}_i^b, x_i; r_i^b)$, output $\perp$.
2. Output $z_i = \mathsf{PRF1.Eval}(K_i[d_i^0, d_i^1], (c_i^0, c_i^1))$.

$G''[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1\}_{i\in[n]}, \{\mathsf{InjOWF}(u_i)\}_{i\in A_w}, w, K_f, \mathsf{InjOWF}]$
$((c_i^0, c_i^1, z_i)_{i\in[n]})$:

1. For $i \in [n]$,
   (a) If $(c_i^0, c_i^1) = (d_i^0, d_i^1)$, if $\mathsf{InjOWF}(z_i) \neq \mathsf{InjOWF}(u_i)$, output $\perp$.
   (b) Else, if $\mathsf{InjOWF}(z_i) \neq \mathsf{InjOWF}(\mathsf{PRF1.Eval}(K_i[d_i^0, d_i^1], c_i^0, c_i^1))$, output $\perp$.
   (c) If $(c_i^0, c_i^1)_{i\in[n]} \geq w$, $x_i = \mathsf{PKE.Dec}(\mathsf{sk}_i^0, c_i^0)$.
   (d) If $(c_i^0, c_i^1)_{i\in[n]} < w$, $x_i = \mathsf{PKE.Dec}(\mathsf{sk}_i^1, c_i^1)$.
2. $r_f = \mathsf{PRF2.Eval}(K_f, (c_i^0, c_i^1)_{i\in[n]})$.

3. Output $y = f(x_1, x_2, \ldots, x_n; r_f)$.

Hybrid$^{\mathcal{A}}_{2,w,2}(1^\lambda)$: Choose u_i uniformly at random. This follows by $\mathsf{PRF1}$ security.

1. **Setup:**
   (a) For $i \in [n]$,
      i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
      ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
   (b) **Compute Values for** w**:**
      i. Parse $w = (d_i^0, d_i^1)_{i\in[n]}$.
      ii. For $i \in [n]$, $\alpha \in \{0,1\}$, $x_i^\alpha = \mathsf{PKE.Dec}(\mathsf{sk}_i^\alpha, d_i^\alpha)$.
      iii. Define $A_w = \{i \in [n] : x_i^0 = x_i^1\}$.
      iv. For $i \in A_w, u_i \leftarrow \{0,1\}^\lambda$.
      v. For $i \in [n], K_i[d_i^0, d_i^1] = \mathsf{PRF1.Punc}(K_i, (d_i^0, d_i^1))$.
   (c) Compute:
      i. For $i \in A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i'[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1, u_i])$.
      ii. For $i \in [n] \setminus A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i''[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1])$.
   (d) For $i \in [n]$, set $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.
   - **Function Query:**
     (a) $\mathcal{A}$ outputs a function f.
     (b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
     (c) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G''[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1\}_{i\in[n]}, \{\mathsf{InjOWF}(u_i)\}_{i\in A_w}, w, K_f, \mathsf{InjOWF}])$.
     (d) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.
   - **Encryption Key Query:**
     (a) $\mathcal{A}$ outputs an index $i \in [n]$.
     (b) Send EK_i to $\mathcal{A}$.
   - **Challenge Message Query:**
     (a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.
     (b) For $i \in [n]$,
        i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.
        ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.
        iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$.
        iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.
        v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.
     (c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

Hybrid$^{\mathcal{A}}_{2,w,3}(1^\lambda)$: Puncture each K_f at w. This step also follows from $i\mathcal{O}$.

1. **Setup:**
   (a) For $i \in [n]$,
      i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
      ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
   (b) **Compute Values for w:**
      i. Parse $w = (d_i^0, d_i^1)_{i\in[n]}$.
      ii. For $i \in [n]$, $\alpha \in \{0,1\}$, $x_i^\alpha = \mathsf{PKE.Dec}(\mathsf{sk}_i^\alpha, d_i^\alpha)$.
      iii. Define $A_w = \{i \in [n] : x_i^0 = x_i^1\}$.
      iv. For $i \in A_w, u_i \leftarrow \{0,1\}^\lambda$.
      v. For $i \in [n], K_i[d_i^0, d_i^1] = \mathsf{PRF1.Punc}(K_i, (d_i^0, d_i^1))$.
   (c) Compute:
      i. For $i \in A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E'_i[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1, u_i])$.
      ii. For $i \in [n] \setminus A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E''_i[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1])$.
   (d) For $i \in [n]$, set $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.
2. $\mathcal{A}$ may make any number of the following queries in any order.
   - **Function Query:**
     (a) $\mathcal{A}$ outputs a function f.
     (b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
     (c) $K_f[w] = \mathsf{PRF2.Punc}(K_f, w)$.
     (d) $r^* \leftarrow \mathsf{PRF2.Eval}(K_f, w)$.
     (e) For $i \in [n], x_i^0 \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_i^0, d_i^0)$.
     (f) $y^* = f(x_1^0, x_2^0, \ldots, x_n^0; r^*)$.
     (g) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G'''[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1\}_{i\in[n]}, \{\mathsf{InjOWF}(u_i)\}_{i\in A_w}, w, K_f[w], y^*, \mathsf{InjOWF}])$.
     (h) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.
   - **Encryption Key Query:**
     (a) $\mathcal{A}$ outputs an index $i \in [n]$.
     (b) Send EK_i to $\mathcal{A}$.
   - **Challenge Message Query:**
     (a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.
     (b) For $i \in [n]$,
        i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.
        ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.
        iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$.
        iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.
        v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.
     (c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.
3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

$G'''[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1\}_{i\in[n]}, \{\mathsf{InjOWF}(u_i)\}_{i\in A_w}, w, K_f[w], y^*, \mathsf{InjOWF}]$ $((c_i^0, c_i^1, z_i)_{i\in[n]})$:

1. For $i \in [n]$,
   (a) If $(c_i^0, c_i^1) = (d_i^0, d_i^1)$, if $\mathsf{InjOWF}(z_i) \neq \mathsf{InjOWF}(u_i)$, output $\perp$.
   (b) Else, if $\mathsf{InjOWF}(z_i) \neq \mathsf{InjOWF}(\mathsf{PRF1.Eval}(K_i[d_i^0, d_i^1], c_i^0, c_i^1))$, output $\perp$.
   (c) If $(c_i^0, c_i^1)_{i\in[n]} > w$, $x_i = \mathsf{PKE.Dec}(\mathsf{sk}_i^0, c_i^0)$.
   (d) If $(c_i^0, c_i^1)_{i\in[n]} = w$, output y^*.
   (e) If $(c_i^0, c_i^1)_{i\in[n]} < w$, $x_i = \mathsf{PKE.Dec}(\mathsf{sk}_i^1, c_i^1)$.
2. $r_f = \mathsf{PRF2.Eval}(K_f[w], (c_i^0, c_i^1)_{i\in[n]})$.
3. Output $y = f(x_1, x_2, \ldots, x_n; r_f)$.

Hybrid$^{\mathcal{A}}_{2,w,4}(1^\lambda)$: Now replace r^* with a uniform value. This follows from puncturable PRF2 security.

1. **Setup:**
   (a) For $i \in [n]$,
      i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
      ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
   (b) **Compute Values for** w**:**
      i. Parse $w = (d_i^0, d_i^1)_{i\in[n]}$.
      ii. For $i \in [n]$, $\alpha \in \{0,1\}$, $x_i^\alpha = \mathsf{PKE.Dec}(\mathsf{sk}_i^\alpha, d_i^\alpha)$.
      iii. Define $A_w = \{i \in [n] : x_i^0 = x_i^1\}$.
      iv. For $i \in A_w, u_i \leftarrow \{0,1\}^\lambda$.
      v. For $i \in [n], K_i[d_i^0, d_i^1] = \mathsf{PRF1.Punc}(K_i, (d_i^0, d_i^1))$.
   (c) Compute:
      i. For $i \in A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i'[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1, u_i])$.
      ii. For $i \in [n] \setminus A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i''[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1])$.
   (d) For $i \in [n]$, set $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.
2. $\mathcal{A}$ may make any number of the following queries in any order.
   - **Function Query:**
     (a) $\mathcal{A}$ outputs a function f.
     (b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
     (c) $K_f[w] = \mathsf{PRF2.Punc}(K_f, w)$.
     (d) $r^* \leftarrow \{0,1\}^\lambda$.
     (e) For $i \in [n], x_i^0 \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_i^0, d_i^0)$.
     (f) $y^* = f(x_1^0, x_2^0, \ldots, x_n^0; r^*)$.
     (g) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G'''[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1\}_{i\in[n]}, \{\mathsf{InjOWF}(u_i)\}_{i\in A_w}, w, K_f[w], y^*,$ $\mathsf{InjOWF}])$.

(h) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.

- **Encryption Key Query:**
  (a) $\mathcal{A}$ outputs an index $i \in [n]$.
  (b) Send EK_i to $\mathcal{A}$.
- **Challenge Message Query:**
  (a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.
  (b) For $i \in [n]$,
    i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.
    ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.
    iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$.
    iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.
    v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.
  (c) Send $\{\mathsf{CT}_{j,i}\}_{i \in [n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

Hybrid$_{2,w,5}^{\mathcal{A}}(1^\lambda)$: In Function Queries, we now compute $x_i^1 \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_i^1, d_i^1)$ instead of $x_i^0 \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_i^0, d_i^0)$, and update y^* to be evaluation on the x_i^1's accordingly. This step follows from $i\mathcal{O}$, InjOWF and $\mathcal{I}$-randomized-compatibility.

1. **Setup:**
  (a) For $i \in [n]$,
    i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
    ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
  (b) **Compute Values for** w**:**
    i. Parse $w = (d_i^0, d_i^1)_{i \in [n]}$.
    ii. For $i \in [n]$, $\alpha \in \{0,1\}$, $x_i^\alpha = \mathsf{PKE.Dec}(\mathsf{sk}_i^\alpha, d_i^\alpha)$.
    iii. Define $A_w = \{i \in [n] : x_i^0 = x_i^1\}$.
    iv. For $i \in A_w, u_i \leftarrow \{0,1\}^\lambda$.
    v. For $i \in [n], K_i[d_i^0, d_i^1] = \mathsf{PRF1.Punc}(K_i, (d_i^0, d_i^1))$.
  (c) Compute:
    i. For $i \in A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i'[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1, u_i])$.
    ii. For $i \in [n] \setminus A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i''[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1])$.
  (d) For $i \in [n]$, set $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.

- **Function Query:**
  (a) $\mathcal{A}$ outputs a function f.
  (b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
  (c) $K_f[w] = \mathsf{PRF2.Punc}(K_f, w)$.
  (d) $r^* \leftarrow \{0,1\}^\lambda$.
  (e) For $i \in [n], x_i^1 \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_i^1, d_i^1)$.

(f) $y^* = f(x_1^1, x_2^1, \ldots, x_n^1; r^*)$.

(g) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G'''[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1\}_{i\in[n]}, \{\mathsf{InjOWF}(u_i)\}_{i\in A_w}, w, K_f[w], y^*, \mathsf{InjOWF}])$.

(h) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.

- **Encryption Key Query:**

  (a) $\mathcal{A}$ outputs an index $i \in [n]$.

  (b) Send EK_i to $\mathcal{A}$.

- **Challenge Message Query:**

  (a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.

  (b) For $i \in [n]$,

    i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.

    ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.

    iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$.

    iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.

    v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.

  (c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

$\mathbf{Hybrid}_{2,w,6}^{\mathcal{A}}(1^\lambda)$: Now we undo the changes introduced in $\mathbf{Hybrid}_{2,w,4}^{\mathcal{A}}$ by changing r^* back to PRF2 evaluation.

1. **Setup:**

   (a) For $i \in [n]$,

     i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.

     ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.

   (b) **Compute Values for** w**:**

     i. Parse $w = (d_i^0, d_i^1)_{i\in[n]}$.

     ii. For $i \in [n]$, $\alpha \in \{0,1\}$, $x_i^\alpha = \mathsf{PKE.Dec}(\mathsf{sk}_i^\alpha, d_i^\alpha)$.

     iii. Define $A_w = \{i \in [n] : x_i^0 = x_i^1\}$.

     iv. For $i \in A_w, u_i \leftarrow \{0,1\}^\lambda$.

     v. For $i \in [n], K_i[d_i^0, d_i^1] = \mathsf{PRF1.Punc}(K_i, (d_i^0, d_i^1))$.

   (c) Compute:

     i. For $i \in A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i'[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1, u_i])$.

     ii. For $i \in [n] \setminus A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i''[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1])$.

   (d) For $i \in [n]$, set $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.

- **Function Query:**

  (a) $\mathcal{A}$ outputs a function f.

  (b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.

  (c) $K_f[w] = \mathsf{PRF2.Punc}(K_f, w)$.

(d) $r^* \leftarrow \mathsf{PRF2.Eval}(K_f, w)$.
(e) For $i \in [n], x_i^1 \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_i^1, d_i^1)$.
(f) $y^* = f(x_1^1, x_2^1, \ldots, x_n^1; r^*)$.
(g) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G'''[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1\}_{i\in[n]}, \{\mathsf{InjOWF}(u_i)\}_{i\in A_w}, w, K_f[w], y^*, \mathsf{InjOWF}])$.
(h) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.

- **Encryption Key Query:**
  (a) $\mathcal{A}$ outputs an index $i \in [n]$.
  (b) Send EK_i to $\mathcal{A}$.
- **Challenge Message Query:**
  (a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.
  (b) For $i \in [n]$,
    i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.
    ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.
    iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$.
    iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.
    v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.
  (c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

$\mathbf{Hybrid}_{2,w,7}^{\mathcal{A}}(1^\lambda)$: Now we undo the changes introduced in $\mathbf{Hybrid}_{2,w,3}^{\mathcal{A}}$ by removing the puncturing on K_f. Notice that results in program G'', but now with index $w+1$, since we are now running $\mathsf{PKE.Dec}(\mathsf{sk}_i^1, c_i^1)$ for w. This step follows from $i\mathcal{O}$.

1. **Setup:**
   (a) For $i \in [n]$,
     i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
     ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
   (b) **Compute Values for w:**
     i. Parse $w = (d_i^0, d_i^1)_{i\in[n]}$.
     ii. For $i \in [n]$, $\alpha \in \{0,1\}$, $x_i^\alpha = \mathsf{PKE.Dec}(\mathsf{sk}_i^\alpha, d_i^\alpha)$.
     iii. Define $A_w = \{i \in [n] : x_i^0 = x_i^1\}$.
     iv. For $i \in A_w, u_i \leftarrow \{0,1\}^\lambda$.
     v. For $i \in [n], K_i[d_i^0, d_i^1] = \mathsf{PRF1.Punc}(K_i, (d_i^0, d_i^1))$.
   (c) Compute:
     i. For $i \in A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i'[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1, u_i])$.
     ii. For $i \in [n] \setminus A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i''[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1])$.
   (d) For $i \in [n]$, set $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.

- **Function Query:**

(a) $\mathcal{A}$ outputs a function f.

(b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.

(c) $\cancel{K_f[w] = \mathsf{PRF2.Punc}(K_f, w).}$

(d) $\cancel{r^* \leftarrow \mathsf{PRF2.Eval}(K_f, w).}$

(e) $\cancel{\text{For } i \in [n], x_i^1 \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_i^1, d_i^1).}$

(f) $\cancel{y^* = f(x_1^1, x_2^1, \ldots, x_n^1; r^*).}$

(g) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G''[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1\}_{i\in[n]}, \{\mathsf{InjOWF}(u_i)\}_{i\in A_w}, w+1, K_f \cancel{[w], y^*}, \mathsf{InjOWF}])$.

(h) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.

- **Encryption Key Query:**

(a) $\mathcal{A}$ outputs an index $i \in [n]$.

(b) Send EK_i to $\mathcal{A}$.

- **Challenge Message Query:**

(a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.

(b) For $i \in [n]$,

i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.

ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.

iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$.

iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.

v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.

(c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

$\mathbf{Hybrid}_{2,w,8}^{\mathcal{A}}(1^\lambda)$: Now we undo the changes introduced in $\mathbf{Hybrid}_{2,w,2}^{\mathcal{A}}$ by changing u_i's back to PRF1 outputs. This follows from PRF1 security.

1. **Setup:**

(a) For $i \in [n]$,

i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.

ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.

(b) **Compute Values for w:**

i. Parse $w = (d_i^0, d_i^1)_{i\in[n]}$.

ii. For $i \in [n]$, $\alpha \in \{0,1\}$, $x_i^\alpha = \mathsf{PKE.Dec}(\mathsf{sk}_i^\alpha, d_i^\alpha)$.

iii. Define $A_w = \{i \in [n] : x_i^0 = x_i^1\}$.

iv. For $i \in A_w, u_i \leftarrow \mathsf{PRF1.Eval}(K_i, (d_i^0, d_i^1)$.

v. For $i \in [n], K_i[d_i^0, d_i^1] = \mathsf{PRF1.Punc}(K_i, (d_i^0, d_i^1))$.

(c) Compute:

i. For $i \in A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i'[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1, u_i])$.

ii. For $i \in [n] \setminus A_w, \widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i''[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1])$.

(d) For $i \in [n]$, set $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.

- **Function Query:**
  (a) $\mathcal{A}$ outputs a function f.
  (b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
  (c) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G''[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1\}_{i\in[n]}, \{\mathsf{InjOWF}(u_i)\}_{i\in A_w}, w{+}1, K_f, \mathsf{InjOWF}])$.
  (d) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.
- **Encryption Key Query:**
  (a) $\mathcal{A}$ outputs an index $i \in [n]$.
  (b) Send EK_i to $\mathcal{A}$.
- **Challenge Message Query:**
  (a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.
  (b) For $i \in [n]$,
    i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.
    ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.
    iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$.
    iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.
    v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.
  (c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

$\mathbf{Hybrid}_{2,w,9}^{\mathcal{A}}(1^\lambda)$: Now we undo the changes introduced in $\mathbf{Hybrid}_{2,w,1}^{\mathcal{A}}$ by removing the puncturing on K_i's (and correspondingly change G'' back to G', and E'' and E' back to E). Notice that this hybrid is now exactly the same as $\mathbf{Hybrid}_{2,w+1,0}^{\mathcal{A}}$.

1. **Setup:**
  (a) For $i \in [n]$,
    i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
    ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
    iii. $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i])$.
    iv. $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.
  (b) ~~**Compute Values for w:**~~
    i. ~~Parse $w = (d_i^0, d_i^1)_{i\in[n]}$.~~
    ii. ~~For $i \in [n]$, $\alpha \in \{0,1\}$, $x_i^\alpha = \mathsf{PKE.Dec}(\mathsf{sk}_i^\alpha, d_i^\alpha)$.~~
    iii. ~~Define $A_w = \{i \in [n] : x_i^0 = x_i^1\}$.~~
    iv. ~~For $i \in A_w$, $u_i \leftarrow \mathsf{PRF1.Eval}(K_i, (d_i^0, d_i^1)$.~~
    v. ~~For $i \in [n]$, $K_i[d_i^0, d_i^1] = \mathsf{PRF1.Punc}(K_i, (d_i^0, d_i^1))$.~~
  (c) ~~Compute:~~
    i. ~~For $i \in A_w$, $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i'[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1, u_i])$.~~
    ii. ~~For $i \in [n] \setminus A_w$, $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i''[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i[d_i^0, d_i^1], d_i^0, d_i^1])$.~~

2. $\mathcal{A}$ may make any number of the following queries in any order.

- **Function Query:**
  (a) $\mathcal{A}$ outputs a function f.
  (b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
  (c) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G'[f, \{\mathsf{sk}_i^0, \mathsf{sk}_i^1, K_i \cancel{[d_i^0, d_i^1], d_i^0, d_i^1}\}_{i\in[n]}, \cancel{\{\mathsf{InjOWF}(u_i)\}_{i\in A_w},} w+1, K_f, \mathsf{InjOWF}])$.
  (d) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.
- **Encryption Key Query:**
  (a) $\mathcal{A}$ outputs an index $i \in [n]$.
  (b) Send EK_i to $\mathcal{A}$.
- **Challenge Message Query:**
  (a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.
  (b) For $i \in [n]$,
    i. $r_{j,i}^0, r_{j,i}^1 \leftarrow \{0,1\}^\lambda$.
    ii. $c_{j,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{j,i}^0; r_{j,i}^0)$.
    iii. $c_{j,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{j,i}^1; r_{j,i}^1)$.
    iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c_{j,i}^0, c_{j,i}^1))$.
    v. $\mathsf{CT}_{j,i} = (c_{j,i}^0, c_{j,i}^1, z_{j,i})$.
  (c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

Hybrid$_3^{\mathcal{A}}(1^\lambda)$: Now we remove sk_i^0 and w from G'. This step follows by $i\mathcal{O}$.

1. **Setup:**
   (a) For $i \in [n]$,
     i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
     ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
     iii. $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i])$.
     iv. $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.

- **Function Query:**
  (a) $\mathcal{A}$ outputs a function f.
  (b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
  (c) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G^\dagger[f, \{\cancel{\mathsf{sk}_i^0}, \mathsf{sk}_i^1, K_i\}_{i\in[n]}, \cancel{w+1}, K_f, \mathsf{InjOWF}])$.
  (d) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.
- **Encryption Key Query:**
  (a) $\mathcal{A}$ outputs an index $i \in [n]$.
  (b) Send EK_i to $\mathcal{A}$.
- **Challenge Message Query:**
  (a) $\mathcal{A}$ outputs $(X_j^0, X_j^1) = ((x_{j,1}^0, \ldots, x_{j,n}^0), (x_{j,1}^1, \ldots, x_{j,n}^1))$.
  (b) For $i \in [n]$,

i. $r^0_{j,i}, r^1_{j,i} \leftarrow \{0,1\}^\lambda$.

ii. $c^0_{j,i} = \mathsf{PKE.Enc}(\mathsf{pk}^0_i, x^0_{j,i}; r^0_{j,i})$.

iii. $c^1_{j,i} = \mathsf{PKE.Enc}(\mathsf{pk}^1_i, x^1_{j,i}; r^1_{j,i})$.

iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c^0_{j,i}, c^1_{j,i}))$.

v. $\mathsf{CT}_{j,i} = (c^0_{j,i}, c^1_{j,i}, z_{j,i})$.

(c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

(Difference from G highlighted)

$G^\dagger[f, \{\mathsf{sk}^1_i, K_i\}_{i\in[n]}, K_f, \mathsf{InjOWF}]((c^0_i, c^1_i, z_i)_{i\in[n]})$:

1. For $i \in [n]$,
   (a) If $\mathsf{InjOWF}(z_i) \neq \mathsf{InjOWF}(\mathsf{PRF1.Eval}(K_i, c^0_i, c^1_i))$, output $\perp$.
   (b) $x_i = \mathsf{PKE.Dec}(\mathsf{sk}^1_i, c^1_i)$.
2. $r_f = \mathsf{PRF2.Eval}(K_f, (c^0_i, c^1_i)_{i\in[n]})$.
3. Output $y = f(x_1, x_2, \ldots, x_n; r_f)$.

Hybrid$^{\mathcal{A}}_4(1^\lambda)$: Now we change $c^0_{j,i}$ to an encryption of $x^1_{j,i}$. This follows by PKE security.

1. **Setup:**
   (a) For $i \in [n]$,
      i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
      ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}^\alpha_i, \mathsf{sk}^\alpha_i) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
      iii. $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i[\mathsf{pk}^0_i, \mathsf{pk}^1_i, K_i])$.
      iv. $\mathsf{EK}_i = (\mathsf{pk}^0_i, \mathsf{pk}^1_i, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.
   - **Function Query:**
     (a) $\mathcal{A}$ outputs a function f.
     (b) $K_f \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
     (c) $\widetilde{G}_f \leftarrow i\mathcal{O}(1^\lambda, G^\dagger[f, \{\mathsf{sk}^1_i, K_i\}_{i\in[n]}, K_f, \mathsf{InjOWF}])$.
     (d) Send $\mathsf{SK}_f = \widetilde{G}_f$ to $\mathcal{A}$.
   - **Encryption Key Query:**
     (a) $\mathcal{A}$ outputs an index $i \in [n]$.
     (b) Send EK_i to $\mathcal{A}$.
   - **Challenge Message Query:**
     (a) $\mathcal{A}$ outputs $(X^0_j, X^1_j) = ((x^0_{j,1}, \ldots, x^0_{j,n}), (x^1_{j,1}, \ldots, x^1_{j,n}))$.
     (b) For $i \in [n]$,
        i. $r^0_{j,i}, r^1_{j,i} \leftarrow \{0,1\}^\lambda$.
        ii. $c^0_{j,i} = \mathsf{PKE.Enc}(\mathsf{pk}^0_i, x^1_{j,i}; r^0_{j,i})$.

iii. $c^1_{j,i} = \mathsf{PKE.Enc}(\mathsf{pk}^1_i, x^1_{j,i}; r^1_{j,i})$.

iv. $z_{j,i} = \mathsf{PRF1.Eval}(K_i, (c^0_{j,i}, c^1_{j,i}))$.

v. $\mathsf{CT}_{j,i} = (c^0_{j,i}, c^1_{j,i}, z_{j,i})$.

(c) Send $\{\mathsf{CT}_{j,i}\}_{i\in[n]}$ to $\mathcal{A}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

Notice that now we have successfully switched $c^0_{j,i}$ and $c^1_{j,i}$ from encrypting $x^0_{j,i}$ to encrypting $x^1_{j,i}$. This is almost the real experiment with $b = 1$, apart from the fact that we are using $G^\dagger$ instead of G in answering the function queries. The remaining hybrids will follow a similar process as we've been doing to change $G^\dagger$ back to G.

Proof of Hybrid Arguments

Lemma 6.2. *If* PKE *is IND-CPA secure, then no PPT adversary can distinguish between* $\mathbf{Hybrid}^{\mathcal{A}}_0$ *and* $\mathbf{Hybrid}^{\mathcal{A}}_1$ *with non-negligible probability.*

Proof. Note that the only difference between the two hybrids is that $c^1_{j,i}$ encrypts $x^0_{j,i}$ in $\mathbf{Hybrid}^{\mathcal{A}}_0$ and $x^1_{j,i}$ in $\mathbf{Hybrid}^{\mathcal{A}}_1$. By the CPA security of PKE, without the private key, no PPT adversary should be able to tell which message is encrypted (and hence distinguish between the two hybrids). □

Lemma 6.3. *If* $i\mathcal{O}$ *is an indistinguishability obfuscation, then no PPT adversary can distinguish between* $\mathbf{Hybrid}^{\mathcal{A}}_1$ *and* $\mathbf{Hybrid}^{\mathcal{A}}_{2,0,0}$ *with non-negligible probability.*

Proof. Here we change from $i\mathcal{O}$ of the program G to $i\mathcal{O}$ of the program G'. In order to invoke the security of $i\mathcal{O}$, we need to show that G and G' and functionally equivalent. Note that for $w = 0$, we always have $(c^0_i, c^1_i) \geq w$. And hence G' always behaves exactly the same as G by decrypting c^0_i. □

Lemma 6.4. *If* $i\mathcal{O}$ *has* $(1, 2^{-3ns-\lambda_{i\mathcal{O}}})$ *weak extractability, then for any distinguisher* $\mathcal{D}$*, we have* $\left|\Pr[\mathcal{D}(\mathbf{Hybrid}^{\mathcal{A}}_{2,w,0}) - \mathcal{D}(\mathbf{Hybrid}^{\mathcal{A}}_{2,w,1})]\right| < O((n + p(\lambda_{i\mathcal{O}})) \cdot 2^{-3ns-\lambda_{i\mathcal{O}}})$ *for some polynomial* p.

Proof. Again here we wish to invoke $i\mathcal{O}$ security. Notice that here we are changing the program E_i to E' and E''_i, and G' to G''. We will argue about their functional equivalence.

- E'_i: Notice if $(c^0_i, c^1_i) = (d^0_i, d^1_i)$, E'_i outputs $u_i = \mathsf{PRF1.Eval}(K_i, (d^0_i, d^1_i))$, which is exactly the output of E_i. If $(c^0_i, c^1_i) \neq (d^0_i, d^1_i)$, then E'_i can successfully use the PRF key punctured at (d^0_i, d^1_i) to evaluate on the point (c^0_i, c^1_i), yielding $\mathsf{PRF1.Eval}(K_i[d^0_i, d^1_i], (c^0_i, c^1_i)) = \mathsf{PRF1.Eval}(K_i, (c^0_i, c^1_i))$, which is the same as the output of E_i.

- E''_i: Notice that E''_i are only handed out for $i \notin A_w$, meaning $x^0_i \neq x^1_i$. Under this case both E_i and E''_i would output $\perp$ and hence are equivalent.

- G'': Notice that the only differences are the cases where G' and G'' will output $\perp$. G' outputs $\perp$ if $\mathsf{InjOWF}(z_i) \neq \mathsf{InjOWF}(\mathsf{PRF1.Eval}(K_i, c^0_i, c^1_i))$. This correspond to the two cases in G''. If $(c^0_i, c^1_i) \neq (d^0_i, d^1_i)$, G'' uses the punctured key to perform the exact same check, and hence have the exact functionality. If $(c^0_i, c^1_i) = (d^0_i, d^1_i)$, then G'' will be checking $\mathsf{InjOWF}(z_i) = \mathsf{InjOWF}(u_i) = \mathsf{InjOWF}(\mathsf{PRF1.Eval}(K_i, (d^0_i, d^1_i)))$, giving us again the exact same check as in G'. Therefore G and G'' have the exact same functionality.

Therefore, all these switches reduce to $i\mathcal{O}$ security. Since we make n switches for E and $p(\lambda_{i\mathcal{O}})$ switches for G, the distinguisher's advantage is upper bounded by $(n + p(\lambda_{i\mathcal{O}})) \cdot 2^{-3ns-\lambda_{i\mathcal{O}}}$. $\square$

Lemma 6.5. *If* $\mathsf{PRF1}$ *has security* $2^{-\lambda_{\mathsf{PRF1}}^{c_{\mathsf{PRF1}}}}$ *with* $\lambda_{\mathsf{PRF1}} \geq (2ns + \lambda_{i\mathcal{O}})^{1/c_{\mathsf{PRF1}}}$*, then for any distinguisher* $\mathcal{D}$*,* $\left|\Pr[\mathcal{D}(\mathbf{Hybrid}^{\mathcal{A}}_{2,w,1}) - \mathcal{D}(\mathbf{Hybrid}^{\mathcal{A}}_{2,w,2})]\right| < O(n \cdot 2^{-2ns-\lambda_{i\mathcal{O}}})$.

Proof. This step follows directly from the security of the puncturable PRF. Since we are only giving out the key punctured at (d_i^0, d_i^1), by puncturable PRF security we can replace the PRF output on that point with a uniformly random value. Here we make $O(n)$ of such switches, and the sub-exponential security of $\mathsf{PRF1}$ guarantees that each switch can be spotted with advantage at most $2^{-\lambda_{\mathsf{PRF1}}^{c_{\mathsf{PRF1}}}} = 2^{-2ns-\lambda_{i\mathcal{O}}}$, therefore, the distinguisher's advantage is upper bounded by $O(n \cdot 2^{-2ns-\lambda_{i\mathcal{O}}})$. $\square$

Lemma 6.6. *If* $i\mathcal{O}$ *has* $(1, 2^{-3ns-\lambda_{i\mathcal{O}}})$ *weak extractability, then for any distinguisher* $\mathcal{D}$*, we have* $\left|\Pr[\mathcal{D}(\mathbf{Hybrid}^{\mathcal{A}}_{2,w,2}) - \mathcal{D}(\mathbf{Hybrid}^{\mathcal{A}}_{2,w,3})]\right| < O(p(\lambda_{i\mathcal{O}}) \cdot 2^{-3ns-\lambda_{i\mathcal{O}}})$ *for some polynomial* p*.*

Proof. Again, here we show functional equivalence to invoke $i\mathcal{O}$ security. Note that the only change is the behavior of the program G'' for cases where $(c_i^0, c_i^1) = w$.

Originally, in $\mathbf{Hybrid}^{\mathcal{A}}_{2,w,2}$, G'' will compute $x_i = \mathsf{PKE.Dec}(\mathsf{sk}_i^0, c_i^0)$ for all $i \in [n]$ and output $y = f(x_1, x_2, \ldots, x_n; r)$ with $r = \mathsf{PRF2.Eval}(K_f, (c_i^0, c_i^1))$.

But now in $\mathbf{Hybrid}^{\mathcal{A}}_{2,w,3}$, G''' will compute $r^* = \mathsf{PRF2.Eval}(K_f, w) = r$, $x_i^0 = \mathsf{PKE.Dec}(\mathsf{sk}_i^0, d_i^0) = \mathsf{PKE.Dec}(\mathsf{sk}_i^0, c_i^0)$ for all i and eventually $y^* = f(x_1^0, x_2^0, \ldots, x_n^0; r^*)$. By inspection, indeed we have $y = y^*$.

Notice here the number of switches we make is equal to the number of function queries, which is bounded by some polynomial p. Therefore, the distinguisher's advantage is bounded by $O(\mathsf{poly}(\lambda_{i\mathcal{O}}) \cdot 2^{-3ns-\lambda_{i\mathcal{O}}})$. $\square$

Lemma 6.7. *If* $\mathsf{PRF2}$ *has security* $2^{-\lambda_{\mathsf{PRF2}}^{c_{\mathsf{PRF2}}}}$ *with* $\lambda_{\mathsf{PRF2}} \geq (2ns + \lambda_{i\mathcal{O}})^{1/c_{\mathsf{PRF2}}}$*, then for any distinguisher* $\mathcal{D}$*,* $\left|\Pr[\mathcal{D}(\mathbf{Hybrid}^{\mathcal{A}}_{2,w,3}) - \mathcal{D}(\mathbf{Hybrid}^{\mathcal{A}}_{2,w,4})]\right| < O(p(\lambda_{i\mathcal{O}}) \cdot 2^{-2ns-\lambda_{i\mathcal{O}}})$ *for some polynomial* p*.*

Proof. Here we are switching $\mathsf{PRF2}$ output on point w with random with the key also punctured on w. We are making $p(\lambda_{i\mathcal{O}})$ switches, so the distinguisher's advantage is upper bounded by $O(p(\lambda_{i\mathcal{O}}) \cdot 2^{-2ns-\lambda_{i\mathcal{O}}})$. $\square$

Lemma 6.8. *If* $i\mathcal{O}$ *has* $(1, 2^{-3ns-\lambda_{i\mathcal{O}}})$ *weak extractability,* PRF*s and* InjOWF *satisfy the requirements outlined in the Parameters section (section 6.1), then for any distinguisher* $\mathcal{D}$*, we have* $\left|\Pr[\mathcal{D}(\mathbf{Hybrid}^{\mathcal{A}}_{2,w,4}) - \mathcal{D}(\mathbf{Hybrid}^{\mathcal{A}}_{2,w,5})]\right| < O(p(\lambda_{i\mathcal{O}}) \cdot 2^{-2ns-\lambda_{i\mathcal{O}}})$ *for some polynomial* p*.*

Proof. Assume towards contradiction that there exists a distinguisher $\mathcal{D}$ that can distinguish between $\mathbf{Hybrid}^{\mathcal{A}}_{2,w,4}$ and $\mathbf{Hybrid}^{\mathcal{A}}_{2,w,5}$ with probability at least $2^{-2ns-\lambda_{i\mathcal{O}}}$). Note that the only difference between $\mathbf{Hybrid}^{\mathcal{A}}_{2,w,4}$ and $\mathbf{Hybrid}^{\mathcal{A}}_{2,w,5}$ is how y^* is computed, which is then used in the generation of the circuit $\widetilde{G}_f$. Since $\widetilde{G}_f$ is an $i\mathcal{O}$ circuit, we break into the following cases:

1. **Case 1:** The circuit G''' in $\mathbf{Hybrid}^{\mathcal{A}}_{2,w,5}$ is functionally equivalent to the G''' circuit in $\mathbf{Hybrid}^{\mathcal{A}}_{2,w,4}$.
2. **Case 2:** The two circuits above are not functionally equivalent. Notice that this means the two y^* values are different in $\mathbf{Hybrid}^{\mathcal{A}}_{2,w,4}$ and $\mathbf{Hybrid}^{\mathcal{A}}_{2,w,5}$. This means $f(x_1^0, x_2^0, \ldots, x_n^0; r^*) \neq f(x_1^1, x_2^1, \ldots, x_n^1; r^*)$.

For Case 1, this reduces directly to the $(1, 2^{-3ns-\lambda_{i\mathcal{O}}})$ weak extractability of the underlying $i\mathcal{O}$ scheme. If any distinguisher can distinguish between the two hybrids in this case, it can also win the $i\mathcal{O}$ weak extractability game, but such winning probability is bounded by $2^{-3ns-\lambda_{i\mathcal{O}}}$. So in case 1, the probability of successfully distinguishing these two hybrids is at most $2^{-3ns-\lambda_{i\mathcal{O}}}$.

Case 2 happens when we have $f(x_1^0, x_2^0, \ldots, x_n^0; r^*) \neq f(x_1^1, x_2^1, \ldots, x_n^1; r^*)$. But notice that by the randomized compatibility requirement, we require their output distributions to be computationally indistinguishable with a sub-exponentially small distinguishing advantage. In this case, the argument proceeds as follow:

1. The only way for the two circuits to yield potentially different outputs is when step 1(d) in G''' is executed. So we will need to have $(c_i^0, c_i^1)_{i\in[n]} = w = (d_i^0, d_i^1)_{i\in[n]}$.
2. In order to proceed to step 1(d), we must fail the check in step 1(a), because otherwise, the circuit will output $\perp$. This means $\mathsf{InjOWF}(z_i) = \mathsf{InjOWF}(u_i)$ for all $i \in [n]$. By security of the injective OWF, it translates to that for each i, with all but $2^{-3ns-\lambda_{i\mathcal{O}}}$ probability, $z_i = u_i$, because otherwise we would have successfully inverted the injective OWF. So union bounding over all i's, we have that with all but $n \cdot 2^{-3ns-\lambda_{i\mathcal{O}}}$, $z_i = u_i$ for all i.
3. Then, we can invoke the randomized compatibility requirement. In order for a distinguisher to distinguish between these two hybrids, it must now distinguish between $f(x_1^0, x_2^0, \ldots, x_n^0; r^*)$ and $f(x_1^1, x_2^1, \ldots, x_n^1; r^*)$. But, by randomized compatibility, for a PPT adversary $\mathcal{A}$, the probability that it can distinguish between the random outputs of $f(x_1^0, x_2^0, \ldots, x_n^0)$ and $f(x_1^1, x_2^1, \ldots, x_n^1)$ is at most $2^{-2ns-\lambda_{i\mathcal{O}}}$. This means the distinguisher's success probability is upper bounded by the adversary's success probability of $2^{-2ns-\lambda_{i\mathcal{O}}}$.

Therefore, the distinguisher's success probability is Case 2 is upper bounded by the probability of inverting the injective OWF and the probability of breaking the randomzied compatibility, and is hence at most $2^{-2ns-\lambda_{i\mathcal{O}}}$ as desired.

We have shown that in both cases, the distinguisher succeeds with probability at most $2^{-2ns-\lambda_{i\mathcal{O}}}$, which concludes the proof. □

The rest of the hybrid arguments follow analogously from the lemmas above.

Notice that to go from $\mathbf{Hybrid}_1^{\mathcal{A}}$ to $\mathbf{Hybrid}_3^{\mathcal{A}}$, we have to go through an exponential $\Theta(2^{2ns})$ number of sub-hybrids. But with the lemmas above, we still have

$$\begin{aligned}
\left|\Pr[\mathcal{D}(\mathbf{Hybrid}_1^{\mathcal{A}}) - \mathcal{D}(\mathbf{Hybrid}_3^{\mathcal{A}})]\right| &= \left|\Pr[\mathcal{D}(\mathbf{Hybrid}_{2,0,0}^{\mathcal{A}}) - \mathcal{D}(\mathbf{Hybrid}_{2,n+1,0}^{\mathcal{A}})]\right| \\
&< 2^{2ns}\Big(2\big((n+p(\lambda)) \cdot 2^{-3ns-\lambda} + n \cdot 2^{-2ns-\lambda} \\
&\quad + p(\lambda) \cdot 2^{-3ns-\lambda} + p(\lambda) \cdot 2^{-2ns-\lambda}\big) + p(\lambda) \cdot 2^{-2ns-\lambda}\Big) \\
&< q(\lambda) \cdot 2^{2ns} \cdot 2^{-2ns-\lambda} = \mathsf{negl}(\lambda),
\end{aligned}$$

where q is some polynomial and $\lambda = \lambda_{i\mathcal{O}}$.

Therefore, we have shown a sequence of polynomial number of hybrids (not counting sub-hybrids; above we have shown the exponential number of sub-hybrids with sub-exponential distinguisher advantages between each pair yieding a nelgigible advantage between the two larger hybrids), where no PPT distinguisher can distinguish adjacent ones with non-negligible probability. Hence we have shown that Construction 2 is secure against malicious decryptors.

6.4 Security against malicious encryptors

Lastly, we prove its security against malicious encryptors.

Theorem 6.9. *If $i\mathcal{O}$ is correct, and* $\mathsf{PRF2}$ *is a secure puncturable PRF, then Construction 2 is secure against malicious encryptors.*

Proof. We prove this through a simple sequence of hybrids.

Sequence of Hybrids

$\mathbf{Hybrid}_0^{\mathcal{A}}(1^\lambda)$: Real world experiment with $b = 0$.

1. **Setup:**
   (a) For $i \in [n]$,
      i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
      ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
      iii. $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i])$.
      iv. $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.
2. $\mathcal{A}$ may make any number of the following queries in any order.
   - **Functional Key Query:**
     (a) $\mathcal{A}$ outputs a function f_ℓ.
     (b) Sample $K_{f_\ell} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
     (c) Compute $\widetilde{G}_{f_\ell} \leftarrow i\mathcal{O}(1^\lambda, G[f_\ell, \{\mathsf{sk}_i^0, K_i\}_{i\in[n]}, K_{f_\ell}, \mathsf{InjOWF}])$.
     (d) Send $\mathsf{SK}_{f_\ell} = \widetilde{G}_{f_\ell}$ to $\mathcal{A}$.
   - **Encryption Key Query:**
     (a) $\mathcal{A}$ outputs an index $i \in [n]$.
     (b) Send EK_i to $\mathcal{A}$.
   - **Function Store Query:**
     (a) $\mathcal{A}$ submits a function g_k.
     (b) If (g_k, sk_{g_k}) already exists in $\mathsf{KeyStore}$, which was initialized as $\mathsf{KeyStore} = \emptyset$ at the beginning of experiment, output 0.
     (c) Sample $K_{g_k} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
     (d) Compute $\widetilde{G}_{g_k} \leftarrow i\mathcal{O}(1^\lambda, G[g_k, \{\mathsf{sk}_i^0, K_i\}_{i\in[n]}, K_{g_k}, \mathsf{InjOWF}])$.
     (e) Store the pair $(g_k, \mathsf{sk}_{g_k} = \widetilde{G}_{g_k})$ in $\mathsf{KeyStore}$ and output 1.
   - **Encryption Query:**
     (a) $\mathcal{A}$ outputs $X_t = (x_{t,1}, \ldots, x_{t,n})$.
     (b) For $i \in [n]$,
        i. $r_{t,i}^0, r_{t,i}^1 \leftarrow \{0,1\}^\lambda$.
        ii. $c_{t,i}^0 = \mathsf{PKE.Enc}(\mathsf{pk}_i^0, x_{t,i}; r_{t,i}^0)$.
        iii. $c_{t,i}^1 = \mathsf{PKE.Enc}(\mathsf{pk}_i^1, x_{t,i}; r_{t,i}^1)$.
        iv. $z_{t,i} = \widetilde{E}_i(c_{t,i}^0, c_{t,i}^1, x_{t,i}, r_{t,i}^0, r_{t,i}^1)$.

v. $\mathsf{CT}_{t,i} = (c^0_{t,i}, c^1_{t,i}, z_{t,i})$.

(c) Send $\{\mathsf{CT}_{t,i}\}_{i\in[n]}$ to $\mathcal{A}$.

- **Challenge Decryption Query:**
  (a) $\mathcal{A}$ submits $\mathsf{CT}_j = (\mathsf{CT}_{j,1}, \ldots, \mathsf{CT}_{j,n})$.
  (b) If $(\mathsf{CT}_j, \{y_{g_k,j}\})$ already exists in $\mathsf{DecStore}$, which is initialized as $\mathsf{DecStore} = \emptyset$ at the beginning of the experiment, output $\{y_{g_k,j}\}$.
  (c) For all $(g_k, \mathsf{sk}_{g_k} = \widetilde{G}_{g_k}) \in \mathsf{KeyStore}$, compute $y_{g_k,j} = \widetilde{G}_{g_k}(\mathsf{CT}_{j,1}, \ldots, \mathsf{CT}_{j,n})$.
  (d) $\mathsf{Store}(\mathsf{CT}_j, \{y_{g_k,j}\}) \in \mathsf{DecStore}$ and output $\{y_{g_k,j}\}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

Hybrid$_1^{\mathcal{A}}(1^\lambda)$: Now when processing function store queries, instead of computing and storing (g_k, sk_{g_k}), store (g_k, K_{g_k}) and later when decryption oracle uses $\mathsf{KeyStore}$, calculate sk_{g_k} using the stored K_{g_k} values. This step follows from the correctness guarantee of the $i\mathcal{O}$ program.

1. **Setup:**
   (a) For $i \in [n]$,
      i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.
      ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}_i^\alpha, \mathsf{sk}_i^\alpha) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
      iii. $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i[\mathsf{pk}_i^0, \mathsf{pk}_i^1, K_i])$.
      iv. $\mathsf{EK}_i = (\mathsf{pk}_i^0, \mathsf{pk}_i^1, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.

- **Functional Key Query:**
  (a) $\mathcal{A}$ outputs a function f_ℓ.
  (b) Sample $K_{f_\ell} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
  (c) Compute $\widetilde{G}_{f_\ell} \leftarrow i\mathcal{O}(1^\lambda, G[f_\ell, \{\mathsf{sk}_i^0, K_i\}_{i\in[n]}, K_{f_\ell}, \mathsf{InjOWF}])$.
  (d) Send $\mathsf{SK}_{f_\ell} = \widetilde{G}_{f_\ell}$ to $\mathcal{A}$.
- **Encryption Key Query:**
  (a) $\mathcal{A}$ outputs an index $i \in [n]$.
  (b) Send EK_i to $\mathcal{A}$.
- **Function Store Query:**
  (a) $\mathcal{A}$ submits a function g_k.
  (b) If (g_k, K_{g_k}) already exists in $\mathsf{KeyStore}$, which was initialized as $\mathsf{KeyStore} = \emptyset$ at the beginning of experiment, output 0.
  (c) Sample $K_{g_k} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
  (d) ~~Compute~~ $\cancel{\widetilde{G}_{g_k} \leftarrow i\mathcal{O}(1^\lambda, G[g_k, \{\mathsf{sk}_i^0, K_i\}_{i\in[n]}, K_{g_k}, \mathsf{InjOWF}])}$.
  (e) Store the pair (g_k, K_{g_k}) in $\mathsf{KeyStore}$ and output 1.
- **Encryption Query:**
  (a) $\mathcal{A}$ outputs $X_t = (x_{t,1}, \ldots, x_{t,n})$.
  (b) For $i \in [n]$,
      i. $r^0_{t,i}, r^1_{t,i} \leftarrow \{0,1\}^\lambda$.

ii. $c^0_{t,i} = \mathsf{PKE.Enc}(\mathsf{pk}^0_i, x_{t,i}; r^0_{t,i})$.

iii. $c^1_{t,i} = \mathsf{PKE.Enc}(\mathsf{pk}^1_i, x_{t,i}; r^1_{t,i})$.

iv. $z_{t,i} = \widetilde{E}_i(c^0_{t,i}, c^1_{t,i}, x_{t,i}, r^0_{t,i}, r^1_{t,i})$.

v. $\mathsf{CT}_{t,i} = (c^0_{t,i}, c^1_{t,i}, z_{t,i})$.

(c) Send $\{\mathsf{CT}_{t,i}\}_{i\in[n]}$ to $\mathcal{A}$.

- **Challenge Decryption Query:**

(a) $\mathcal{A}$ submits $\mathsf{CT}_j = (\mathsf{CT}_{j,1}, \ldots, \mathsf{CT}_{j,n})$.

(b) If $(\mathsf{CT}_j, \{y_{g_k,j}\})$ already exists in $\mathsf{DecStore}$, which is initialized as $\mathsf{DecStore} = \emptyset$ at the beginning of the experiment, output $\{y_{g_k,j}\}$.

(c) For all $(g_k, K_{g_k}) \in \mathsf{KeyStore}$:

i. Compute $\widetilde{G}_{g_k} \leftarrow i\mathcal{O}(1^\lambda, G[g_k, \{\mathsf{sk}^0_i, K_i\}_{i\in[n]}, K_{g_k}, \mathsf{InjOWF}])$.

ii. Compute $y_{g_k,j} = \widetilde{G}_{g_k}(\mathsf{CT}_{j,1}, \ldots, \mathsf{CT}_{j,n})$.

(d) $\mathsf{Store}(\mathsf{CT}_j, \{y_{g_k,j}\}) \in \mathsf{DecStore}$ and output $\{y_{g_k,j}\}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

Hybrid$^{\mathcal{A}}_2(1^\lambda)$: Now we change the PRF2 keys to be lazily and uniformly sampled in the Decryption oracle. This step is a syntactical change.

1. **Setup:**

(a) For $i \in [n]$,

i. $K_i \leftarrow \mathsf{PRF1.Setup}(1^\lambda)$.

ii. For $\alpha \in \{0,1\}$, $(\mathsf{pk}^\alpha_i, \mathsf{sk}^\alpha_i) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.

iii. $\widetilde{E}_i = i\mathcal{O}(1^\lambda, E_i[\mathsf{pk}^0_i, \mathsf{pk}^1_i, K_i])$.

iv. $\mathsf{EK}_i = (\mathsf{pk}^0_i, \mathsf{pk}^1_i, \widetilde{E}_i)$.

2. $\mathcal{A}$ may make any number of the following queries in any order.

- **Functional Key Query:**

(a) $\mathcal{A}$ outputs a function f_ℓ.

(b) Sample $K_{f_\ell} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.

(c) Compute $\widetilde{G}_{f_\ell} \leftarrow i\mathcal{O}(1^\lambda, G[f_\ell, \{\mathsf{sk}^0_i, K_i\}_{i\in[n]}, K_{f_\ell}, \mathsf{InjOWF}])$.

(d) Send $\mathsf{SK}_{f_\ell} = \widetilde{G}_{f_\ell}$ to $\mathcal{A}$.

- **Encryption Key Query:**

(a) $\mathcal{A}$ outputs an index $i \in [n]$.

(b) Send EK_i to $\mathcal{A}$.

- **Function Store Query:**

(a) $\mathcal{A}$ submits a function g_k.

(b) If (g_k, K_{g_k}) already exists in $\mathsf{KeyStore}$, which was initialized as $\mathsf{KeyStore} = \emptyset$ at the beginning of experiment, output 0.

(c) ~~Sample $K_{g_k} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.~~

(d) Store the pair $(g_k, \perp)$ in $\mathsf{KeyStore}$ and output 1.

- **Encryption Query:**

(a) $\mathcal{A}$ outputs $X_t = (x_{t,1}, \ldots, x_{t,n})$.

(b) For $i \in [n]$,

i. $r^0_{t,i}, r^1_{t,i} \leftarrow \{0,1\}^\lambda$.

ii. $c^0_{t,i} = \mathsf{PKE.Enc}(\mathsf{pk}^0_i, x_{t,i}; r^0_{t,i})$.

iii. $c^1_{t,i} = \mathsf{PKE.Enc}(\mathsf{pk}^1_i, x_{t,i}; r^1_{t,i})$.

iv. $z_{t,i} = \widetilde{E}_i(c^0_{t,i}, c^1_{t,i}, x_{t,i}, r^0_{t,i}, r^1_{t,i})$.

v. $\mathsf{CT}_{t,i} = (c^0_{t,i}, c^1_{t,i}, z_{t,i})$.

(c) Send $\{\mathsf{CT}_{t,i}\}_{i\in[n]}$ to $\mathcal{A}$.

- **Challenge Decryption Query:**

(a) $\mathcal{A}$ submits $\mathsf{CT}_j = (\mathsf{CT}_{j,1}, \ldots, \mathsf{CT}_{j,n})$.

(b) If $(\mathsf{CT}_j, \{y_{g_k,j}\})$ already exists in $\mathsf{DecStore}$, which is initialized as $\mathsf{DecStore} = \emptyset$ at the beginning of the experiment, output $\{y_{g_k,j}\}$.

(c) For all $(g_k, K_{g_k}) \in \mathsf{KeyStore}$:

i. If $K_{g_k} = \perp$, sample $K_{g_k} \leftarrow \{0,1\}^\lambda$ and update (g_k, K_{g_k}) in $\mathsf{KeyStore}$.

ii. Compute $\widetilde{G}_{g_k} \leftarrow i\mathcal{O}(1^\lambda, G[g_k, \{\mathsf{sk}^0_i, K_i\}_{i\in[n]}, K_{g_k}, \mathsf{InjOWF}])$.

iii. Compute $y_{g_k,j} = \widetilde{G}_{g_k}(\mathsf{CT}_{j,1}, \ldots, \mathsf{CT}_{j,n})$.

(d) $\mathsf{Store}(\mathsf{CT}_j, \{y_{g_k,j}\}) \in \mathsf{DecStore}$ and output $\{y_{g_k,j}\}$.

3. **Output:** $\mathcal{A}$ outputs a bit b'. Output b' if the queries are compatible, and 0 otherwise.

Notice now when we answer decryption queries, we effectively sample a uniformly random $\mathsf{PRF2}$ key K_{g_k}, and evaluate it on the ciphertext to obtain the randomness. Notice that a malicious encryptor cannot affect our uniform choice of $\mathsf{PRF2}$ key in any way, therefore by $\mathsf{PRF2}$ security, the $\mathsf{PRF2}$ output will always be close to uniformly random. Consequently, the functions will be evaluated using a uniform randomness that the malicious encryptor cannot tamper with.

□

7 Acknowledgements

This research was supported in part from a Simons Investigator Award, DARPA SIEVE award, NTT Research, NSF grant2333935, BSF grant 2022370, a Xerox Faculty Research Award, a Google Faculty Research Award, an Okawa Foundation Research Grant, and the Symantec Chair of Computer Science. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024.

8 References

[ABB10a] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572. Springer, Berlin, Heidelberg, May / June 2010.

[ABB10b] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 98–115. Springer, Berlin, Heidelberg, August 2010.

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

[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 Martijn Stam, editor, *14th IMA International Conference on Cryptography and Coding*, volume 8308 of *LNCS*, pages 65–84. Springer, Berlin, Heidelberg, December 2013.

[ABG19] Michel Abdalla, Fabrice Benhamouda, and Romain Gay. From single-input to multi-client inner-product functional encryption. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 552–582. Springer, Cham, December 2019.

[ABKW19] Michel Abdalla, Fabrice Benhamouda, Markulf Kohlweiss, and Hendrik Waldner. Decentralizing inner-product functional encryption. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 128–157. Springer, Cham, April 2019.

[ABM$^+$20] Michel Abdalla, Florian Bourse, Hugo Marival, David Pointcheval, Azam Soleimanian, and Hendrik Waldner. Multi-client inner-product functional encryption in the random-oracle model. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 525–545. Springer, Cham, September 2020.

[ACF$^+$18] Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 597–627. Springer, Cham, August 2018.

[ACF$^+$20] Shweta Agrawal, Michael Clear, Ophir Frieder, Sanjam Garg, Adam O'Neill, and Justin Thaler. Ad hoc multi-input functional encryption. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 40:1–40:41. LIPIcs, January 2020.

[ACGU20] Michel Abdalla, Dario Catalano, Romain Gay, and Bogdan Ursu. Inner-product functional encryption with fine-grained access control. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 467–497. Springer, Cham, December 2020.

[AFS21] Miguel Ambrona, Dario Fiore, and Claudio Soriente. Controlled functional encryption revisited: Multi-authority extensions and efficient schemes for quadratic functions. *PoPETs*, 2021(1):21–42, January 2021.

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

[AGRW17] Michel Abdalla, Romain Gay, Mariana Raykova, and Hoeteck Wee. Multi-input inner-product functional encryption from pairings. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 601–626. Springer, Cham, April / May 2017.

[AGT21] Shweta Agrawal, Rishab Goyal, and Junichi Tomida. Multi-input quadratic functional encryption from pairings. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 208–238, Virtual Event, August 2021. Springer, Cham.

[AGT22] Shweta Agrawal, Rishab Goyal, and Junichi Tomida. Multi-input quadratic functional encryption: Stronger security, broader functionality. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 711–740. Springer, Cham, November 2022.

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

[AGW20] Michel Abdalla, Junqing Gong, and Hoeteck Wee. Functional encryption for attribute-weighted sums from k-Lin. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 685–716. Springer, Cham, August 2020.

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

[AJS18] Prabhanjan Ananth, Aayush Jain, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: iO from LWE, bilinear maps, and weak pseudorandomness. Cryptology ePrint Archive, Report 2018/615, 2018.

[AKM$^+$22] Shweta Agrawal, Fuyuki Kitagawa, Anuja Modi, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Bounded functional encryption for Turing machines: Adaptive security from general assumptions. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 618–647. Springer, Cham, November 2022.

[AKY24] Shweta Agrawal, Simran Kumari, and Shota Yamada. Attribute based encryption for turing machines from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 352–386. Springer, Cham, August 2024.

[ALMT20] Shweta Agrawal, Benoît Libert, Monosij Maitra, and Radu Titiu. Adaptive simulation security for inner product functional encryption. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 34–64. Springer, Cham, May 2020.

[ALS15] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. Cryptology ePrint Archive, Report 2015/608, 2015.

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

[ARYY23] Shweta Agrawal, Mélissa Rossi, Anshu Yadav, and Shota Yamada. Constant input attribute based (and predicate) encryption from evasive and tensor LWE. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 532–564. Springer, Cham, August 2023.

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for Turing machines. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 125–153. Springer, Berlin, Heidelberg, January 2016.

[Att14] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 557–577. Springer, Berlin, Heidelberg, May 2014.

[ATY23] Shweta Agrawal, Junichi Tomida, and Anshu Yadav. Attribute-based multi-input FE (and more) for attribute-weighted sums. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 464–497. Springer, Cham, August 2023.

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

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

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

[AYY22] Shweta Agrawal, Anshu Yadav, and Shota Yamada. Multi-input attribute based encryption and predicate encryption. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 590–621. Springer, Cham, August 2022.

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

[BBL17] Fabrice Benhamouda, Florian Bourse, and Helger Lipmaa. CCA-secure inner-product functional encryption from projective hash functions. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 36–66. Springer, Berlin, Heidelberg, March 2017.

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

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 52–73. Springer, Berlin, Heidelberg, February 2014.

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

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

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

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Berlin, Heidelberg, March 2014.

[BGJS15] Saikrishna Badrinarayanan, Divya Gupta, Abhishek Jain, and Amit Sahai. Multi-input functional encryption for unbounded arity functions. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 27–51. Springer, Berlin, Heidelberg, November / December 2015.

[BGJS16] Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. Verifiable functional encryption. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 557–587. Springer, Berlin, Heidelberg, December 2016.

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

[BJK+18] Zvika Brakerski, Aayush Jain, Ilan Komargodski, Alain Passelègue, and Daniel Wichs. Non-trivial witness encryption and null-iO from standard assumptions. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 425–441. Springer, Cham, September 2018.

[BKMT21] Pramod Bhatotia, Markulf Kohlweiss, Lorenzo Martinico, and Yiannis Tselekounis. Steel: Composable hardware-based stateful and randomised functional encryption. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 709–736. Springer, Cham, May 2021.

[BKS16] Zvika Brakerski, Ilan Komargodski, and Gil Segev. Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 852–880. Springer, Berlin, Heidelberg, May 2016.

[BKSW18] Saikrishna Badrinarayanan, Dakshita Khurana, Amit Sahai, and Brent Waters. Upgrading to functional encryption. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 629–658. Springer, Cham, November 2018.

[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 Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 563–594. Springer, Berlin, Heidelberg, April 2015.

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

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Berlin, Heidelberg, December 2013.

[BZ16] Mark Bun and Mark Zhandry. Order-revealing encryption and the hardness of private learning. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 176–206. Springer, Berlin, Heidelberg, January 2016.

[CDG+18a] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 703–732. Springer, Cham, December 2018.

[CDG+18b] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Multi-client functional encryption with repetition for inner product. Cryptology ePrint Archive, Report 2018/1021, 2018.

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

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

[CGKW18] Jie Chen, Junqing Gong, Lucas Kowalczyk, and Hoeteck Wee. Unbounded ABE via bilinear entropy expansion, revisited. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 503–534. Springer, Cham, April / May 2018.

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

[CLT18] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 733–764. Springer, Cham, December 2018.

[CLWW16] Nathan Chenette, Kevin Lewi, Stephen A. Weis, and David J. Wu. Practical order-revealing encryption with limited leakage. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 474–493. Springer, Berlin, Heidelberg, March 2016.

[CMR17] Brent Carmer, Alex J. Malozemoff, and Mariana Raykova. 5Gen-C: Multi-input functional encryption and program obfuscation for arithmetic circuits. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 747–764. ACM Press, October / November 2017.

[CW23] Valerio Cini and Hoeteck Wee. ABE for circuits with poly (λ)-sized keys from LWE. In *64th FOCS*, pages 435–446. IEEE Computer Society Press, November 2023.

[DDM16] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Functional encryption for inner product with full function privacy. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 164–195. Springer, Berlin, Heidelberg, March 2016.

[DI13] Angelo De Caro and Vincenzo Iovino. On the power of rewinding simulators in functional encryption. Cryptology ePrint Archive, Report 2013/752, 2013.

[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 Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 519–535. Springer, Berlin, Heidelberg, August 2013.

[DIPV17] Yvo Desmedt, Vincenzo Iovino, Giuseppe Persiano, and Ivan Visconti. Controlled homomorphic encryption: Definition and construction. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *FC 2017 Workshops*, volume 10323 of *LNCS*, pages 107–129. Springer, Cham, April 2017.

[DOT18a] Pratish Datta, Tatsuaki Okamoto, and Katsuyuki Takashima. Adaptively simulation-secure attribute-hiding predicate encryption. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 640–672. Springer, Cham, December 2018.

[DOT18b] Pratish Datta, Tatsuaki Okamoto, and Junichi Tomida. Full-hiding (unbounded) multi-input inner product functional encryption from the k-Linear assumption. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 245–277. Springer, Cham, March 2018.

[DP21] Pratish Datta and Tapas Pal. (Compact) adaptively secure FE for attribute-weighted sums from k-lin. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 434–467. Springer, Cham, December 2021.

[DPT22] Pratish Datta, Tapas Pal, and Katsuyuki Takashima. Compact FE for unbounded attribute-weighted sums for logspace from SXDH. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 126–159. Springer, Cham, December 2022.

[FFMV23] Danilo Francati, Daniele Friolo, Giulio Malavolta, and Daniele Venturi. Multi-key and multi-input predicate encryption from learning with errors. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 573–604. Springer, Cham, April 2023.

[FFMV24] Danilo Francati, Daniele Friolo, Giulio Malavolta, and Daniele Venturi. Multi-key and multi-input predicate encryption (for conjunctions) from learning with errors. *Journal of Cryptology*, 37(3):24, July 2024.

[Gay20] Romain Gay. A new paradigm for public-key functional encryption for degree-2 polynomials. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 95–120. Springer, Cham, May 2020.

[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 Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, Berlin, Heidelberg, May 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 *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 518–535. Springer, Berlin, Heidelberg, August 2014.

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

[GGLW22] Rachit Garg, Rishab Goyal, George Lu, and Brent Waters. Dynamic collusion bounded functional encryption from identity-based encryption. In Orr Dunkelman and Stefan

Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 736–763. Springer, Cham, May / June 2022.

[GJKS15] Vipul Goyal, Abhishek Jain, Venkata Koppula, and Amit Sahai. Functional encryption for randomized functionalities. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 325–351. Springer, Berlin, Heidelberg, March 2015.

[GJO16] Vipul Goyal, Aayush Jain, and Adam O'Neill. Multi-input functional encryption with unbounded-message security. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 531–556. Springer, Berlin, Heidelberg, December 2016.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In Chris Umans, editor, *58th FOCS*, pages 612–621. IEEE Computer Society Press, October 2017.

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

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

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 503–523. Springer, Berlin, Heidelberg, August 2015.

[GW20] Junqing Gong and Hoeteck Wee. Adaptively secure ABE for DFA from k-Lin and more. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 278–308. Springer, Cham, May 2020.

[GWW19] Junqing Gong, Brent Waters, and Hoeteck Wee. ABE for DFA from k-Lin. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 732–764. Springer, Cham, August 2019.

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th FOCS*, pages 415–434. IEEE Computer Society Press, November 2023.

[HLL24] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. A general framework for lattice-based ABE using evasive inner-product functional encryption. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 433–464. Springer, Cham, May 2024.

[ITZ16] Vincenzo Iovino, Qiang Tang, and Karol Zebrowski. On the power of public-key function-private functional encryption. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16*, volume 10052 of *LNCS*, pages 585–593. Springer, Cham, November 2016.

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

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

[KLM$^+$18] Sam Kim, Kevin Lewi, Avradip Mandal, Hart Montgomery, Arnab Roy, and David J. Wu. Function-hiding inner product encryption is practical. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 544–562. Springer, Cham, September 2018.

[KNT18] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Obfustopia built on secret-key functional encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 603–648. Springer, Cham, April / May 2018.

[KNT21] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Simple and generic constructions of succinct functional encryption. *Journal of Cryptology*, 34(3):25, July 2021.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[KS17] Ilan Komargodski and Gil Segev. From minicrypt to obfustopia via private-key functional encryption. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 122–151. Springer, Cham, April / May 2017.

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

[KSY15] Ilan Komargodski, Gil Segev, and Eylon Yogev. Functional encryption for randomized functionalities in the private-key setting from minimal assumptions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 352–377. Springer, Berlin, Heidelberg, March 2015.

[KW20] Lucas Kowalczyk and Hoeteck Wee. Compact adaptively secure ABE for NC^1 from k-Lin. *Journal of Cryptology*, 33(3):954–1002, July 2020.

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

[LL20a] Huijia Lin and Ji Luo. Compact adaptively secure ABE from k-Lin: Beyond NC^1 and towards NL. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 247–277. Springer, Cham, May 2020.

[LL20b] Huijia Lin and Ji Luo. Succinct and adaptively secure ABE for ABP from k-Lin. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 437–466. Springer, Cham, December 2020.

[LOS+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 62–91. Springer, Berlin, Heidelberg, May / June 2010.

[LT19] Benoît Libert and Radu Titiu. Multi-client functional encryption for linear functions in the standard model from LWE. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 520–551. Springer, Cham, December 2019.

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

[LW10] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 455–479. Springer, Berlin, Heidelberg, February 2010.

[LW12] Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 180–198. Springer, Berlin, Heidelberg, August 2012.

[LW16] Kevin Lewi and David J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1167–1178. ACM Press, October 2016.

[LZ20] Muhua Liu and Ping Zhang. An adaptively secure functional encryption for randomized functions. *The Computer Journal*, 63(1):1247–1258, 2020.

[NPP22] Ky Nguyen, Duong Hieu Phan, and David Pointcheval. Multi-client functional encryption with fine-grained access control. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 95–125. Springer, Cham, December 2022.

[NPP23a] Dinh Duy Nguyen, Duong Hieu Phan, and David Pointcheval. Verifiable decentralized multi-client functional encryption for inner product. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part V*, volume 14442 of *LNCS*, pages 33–65. Springer, Singapore, December 2023.

[NPP23b] Ky Nguyen, Duong Hieu Phan, and David Pointcheval. Optimal security notion for decentralized multi-client functional encryption. In Mehdi Tibouchi and Xiaofeng Wang, editors, *ACNS 23International Conference on Applied Cryptography and Network Security, Part II*, volume 13906 of *LNCS*, pages 336–365. Springer, Cham, June 2023.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 191–208. Springer, Berlin, Heidelberg, August 2010.

[OT12] Tatsuaki Okamoto and Katsuyuki Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 591–608. Springer, Berlin, Heidelberg, April 2012.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 433–444. Springer, Berlin, Heidelberg, August 1992.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.

[SV23] Elaine Shi and Nikhil Vanjani. Multi-client inner product encryption: Function-hiding instantiations without random oracles. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 622–651. Springer, Cham, May 2023.

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

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

[Tom19] Junichi Tomida. Tightly secure inner product functional encryption: Multi-input and function-hiding constructions. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 459–488. Springer, Cham, December 2019.

[Tom23] Junichi Tomida. Unbounded quadratic functional encryption and more from pairings. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 543–572. Springer, Cham, April 2023.

[TT18] Junichi Tomida and Katsuyuki Takashima. Unbounded inner product functional encryption from bilinear maps. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 609–639. Springer, Cham, December 2018.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 619–636. Springer, Berlin, Heidelberg, August 2009.

[Wat12] Brent Waters. Functional encryption for regular languages. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 218–235. Springer, Berlin, Heidelberg, August 2012.

[Wee14] Hoeteck Wee. Dual system encryption via predicate encodings. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 616–637. Springer, Berlin, Heidelberg, February 2014.

[Wee17] Hoeteck Wee. Attribute-hiding predicate encryption in bilinear groups, revisited. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 206–233. Springer, Cham, November 2017.

[Wee20] Hoeteck Wee. Functional encryption for quadratic functions from k-lin, revisited. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 210–228. Springer, Cham, November 2020.

[Wee21] Hoeteck Wee. Broadcast encryption with size $N^{1/3}$ and more from k-lin. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 155–178, Virtual Event, August 2021. Springer, Cham.

[Wee22] Hoeteck Wee. Optimal broadcast encryption and CP-ABE from evasive lattice assumptions. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 217–241. Springer, Cham, May / June 2022.

[Wee24] Hoeteck Wee. Circuit ABE with $\mathsf{poly}(\text{depth}, \lambda)$-sized ciphertexts and keys from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 178–209. Springer, Cham, August 2024.

[WW24a] Brent Waters and David J. Wu. Adaptively-sound succinct arguments for NP from indistinguishability obfuscation. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th ACM STOC*, pages 387–398. ACM Press, June 2024.

[WW24b] Brent Waters and David J. Wu. A pure indistinguishability obfuscation approach to adaptively-sound SNARGs for NP. Cryptology ePrint Archive, Report 2024/933, 2024.