Lower Bounds on Inner-Product Functional Encryption from All-or-Nothing Encryption Primitives[*]

Jinye He
University of Virginia
qfn5bh@virginia.edu

Shiyu Li
University of Virginia
tkp2ra@virginia.edu

Wei-Kai Lin
University of Virginia
wklin@viginia.edu

November 30, 2025

Abstract

A functional encryption (FE) is a versatile encryption, where the functional key $\mathsf{sk}_{\mathbf{v}}$ decrypts a ciphertext to the function $\mathbf{v}(\mathbf{m})$ evaluated on the plaintext $\mathbf{m}$. The security requires that even when an adversary colludes multiple functional keys, it learns only the evaluations but nothing more about the plaintext. This work considers the adversary obtaining an *unbounded number of colluded keys*, a natural setting for many FE applications. We aim to understand which cryptographic primitives are sufficient to construct an unbounded-collusion FE.

This work proves negative results: we separate unbounded-collusion FEs from many encryption primitives that are "all-or-nothing." Introduced by Garg, Mahmoody and Mohammed (CRYPTO '17), an all-or-nothing encryption scheme allows an adversary to learn either *the whole plaintext* if the authorized secret key is given; otherwise, the adversary learns *nothing*. Hence, we rule out such FE construction from fully homomorphic encryption (FHE), attribute-based encryption (ABE), and predicate encryptions (PE). Our impossibility holds for private-key and selectively secure FEs for an minimal function class, *inner products*, making the impossibility stronger. Our separation is in the "monolithic model," similar to the black-box model, but the primitive is allowed to evaluate a circuit that queries the primitive itself as an oracle.

Our result extends the beautiful work of Hajiabadi et al. (TCC '24), which separated inner-product functional encryption (IPFE) from any primitives that exist in the random oracle model. Our result is perhaps surprising as IPFE was proposed by Abdalla et al. (PKC '15) to be easier to construct, and indeed there are several IPFEs based on standard *algebraic* assumptions, such as Decisional Diffie-Hellman or Learning-with-Errors. Moreover, *bounded*-collusion FEs for all circuits are constructed in a black-box way from minimal assumptions, i.e., public-key FE from public-key encryption, and secret-key FE from one-way functions, by Ananth and Vaikuntanathan (TCC '19).

[*]©IACR 2025. This article is a minor revision of the version published by Springer-Verlag available at ¡DOI¿.

Contents

1 Introduction

Functional encryptions (FE) are encryption schemes that provide extremely versatile access control to ciphertexts [BDOP04, SW05, O'N10, BSW12, GGH+13, Wat15, GGHZ16]. An FE scheme is defined for a class of functions $\mathcal{C}$. Given a function $\mathbf{v} \in \mathcal{C}$, FE generates a functional secret key $\mathsf{sk}_{\mathbf{v}}$. Independent of $\mathbf{v}$, FE encrypts a private message $\mathbf{m}$ into a ciphertext c. Then, using the functional secret key $\mathsf{sk}_{\mathbf{v}}$, the functional decryption of c returns the evaluated result $\mathbf{v}(\mathbf{m})$. There can be many functional secret keys, which makes FE versatile: a different user, who holds another secret key $\mathsf{sk}_{\mathbf{v}'}$ which is generated for a different function $\mathbf{v}' \in \mathcal{C}$, can learn a different result $\mathbf{v}'(\mathbf{m})$ from the same ciphertext c. The security of FE requires that an adversary who colludes a set of keys $\mathsf{sk}_{\mathbf{v}_1}, \mathsf{sk}_{\mathbf{v}_2}, \ldots, \mathsf{sk}_{\mathbf{v}_t}$ for some t learns only $\mathbf{v}_1(\mathbf{m}), \mathbf{v}_2(\mathbf{m}), \ldots, \mathbf{v}_t(\mathbf{m})$ but nothing more about the private message $\mathbf{m}$. Hence, the functional secret key controls the access to $\mathbf{m}$ in a fine-grained way. The more expressive the class $\mathcal{C}$ is, the better control over the private message we achieve.

Unbounded collusion, private-key, and selective settings. A standard security of FE is an unbounded number of colluded functional keys, i.e., the adversary requests an arbitrary polynomial number of functional keys for any functions of the adversary's choice [BSW11]. Particularly, the FE scheme shall guarantee security, even though the scheme was not given the number of collusions in advance. Recall that *bounded-collusion* FEs are constructed from attribute-based encryption (ABE) and fully homomorphic encryption (FHE) in an earlier work [GKP+13], and then the construction is improved and based only on minimal assumptions of public-key encryption or one-way functions [AV19]. Henceforth, we consider *unbounded-collusion* FEs throughout this paper, and we aim to answer what cryptographic primitives are necessary to construct unbounded-collusion FEs.

There are a couple of FE settings in addition to bounded or unbounded collusion. Firstly, the encryption can be either *public-key or private-key*. In the public-key setting, the adversary gets the public key to encrypt any message of its choice. In contrast, in the private-key setting, the adversary only gets chosen-plaintext queries. Secondly, in the security game, the adversary may *selectively or adaptively* request the colluded functional keys and ciphertexts. We will consider private-key and selectively secure FEs to make our impossibility stronger.

Minimal class of functions: inner products. The *class of functions* of FE schemes can differ from all polynomial-size boolean circuits to a set of simple functions, such as inner products. Clearly, it is more difficult, or at least no easier, to construct an FE for a more expressive class of functions. It is well-known that using indistinguishable obfuscation (IO) [BGI+01], we can construct unbounded-collusion FEs for all circuits [GGH+13, Wat14]. On the other hand, bounded-collusion but compact and public-key FEs for all circuits imply IO [AJ15, BV15], where compactness requires that the encryption key is a fixed polynomial in the security parameter (but independent of the circuit size). Hence, unbounded-collusion and public-key FE for all circuits is existentially equivalent to IO. However, almost all IO candidates rely on strong *algebraic or combinatorial* assumptions, e.g., [JLS21, JLS22]. Moreover, previous works separate IO from many other "fancy" and strong *cryptographic primitives*, such as attribute-based encryption (ABE), fully homomorphic encryption (FHE), and predicate encryption (PE) [AS15, GMM17].[1]

[1] In this paper, PEs refer to the attribute-hiding notion [BW07], while ABEs are attribute-revealing.

Hence, we focus on the feasibility of the weakest FEs. That is, FEs for less expressive classes, and it is stronger in terms of impossibility. Historically, FE for inner products (IPFE) was proposed to achieve better efficiency and circumvent the impossibilities, at the cost of restricting the class to inner products [ABDP15], a minimal but non-trivial class. IPFE is parameterized by the vector space $\mathbb{Z}_p^n$ for some modulus p and dimension n. Each function $\mathbf{v}$ and each message $\mathbf{m}$ is a vector in $\mathbb{Z}_p^n$, and the function $\mathbf{v}(\mathbf{m})$ is defined to be the inner product $\langle \mathbf{v}, \mathbf{m} \rangle$ over $\mathbb{Z}_p$. Following FE, given the functional secret key $\mathsf{sk}_\mathbf{v}$ and a ciphertext c encrypting $\mathbf{m}$, one can perform functional decryption and learn the inner product $\langle \mathbf{v}, \mathbf{m} \rangle$, but nothing more about $\mathbf{m}$. IPFE applies to many real-world scenarios, e.g., the two-party computation of weighted mean [ABDP15] and nearest-neighbor search on encrypted data [KLM+18]. IPFE relies only on relatively weaker assumptions, such as Discrete Diffie-Hellman (DDH), Decisional Composite Residuosity (DCR), or Learning With Errors (LWE) [ABDP15, ALS16, KLM+18].

The assumptions of the minimal FE/IPFE. To the best of our knowledge, almost all constructions of IPFE are based on *algebraic* assumptions, such as DDH, DCR, or LWE as mentioned above. Recently, Hajiabadi et al. [HLOW24] initialized the study and separated even the *private-key* IPFE from any primitives that exist in the random oracle model, such as collision-resistant hash functions and private-key encryptions. That is, the capability of functional decryption is beyond any black-box use of the random oracle. One may conjecture that we can separate IPFE from stronger primitives, e.g., trapdoor permutations, and the question can be extended to even stronger primitives, such as FHE. We aim to assess whether even the most expressive primitives suffice to construct the minimal form of FE, i.e., IPFE.

> *Can we construct the weaker variants of (inner-product) functional encryption schemes from the "mildly fancy" encryption schemes, such as IBE, ABE, PE, or even FHE?*

To prove the impossibility, a potential way is to construct a sufficiently strong primitive, such as identity-based encryption (IBE), from a variant of IPFE, or even from FE for a mildly expressive class of functions. Because there is no black-box construction of IBE from trapdoor permutations [BPR+08], by constructing IBE, we might prove that there is no black-box construction of IPFE from trapdoor permutations as well. Other separations between predicate encryptions (PE) [KY09, GKLM12] might also be helpful in this way. Unfortunately, Asharov and Segev [AS15] proved that there is no fully black-box construction of a key-agreement protocol with perfect completeness from general-purpose private-key functional encryptions. Since the public-key primitives (i.e., IBE and PE) imply key agreement [GKM+00], an IBE from private-key (IP)FE would contradict Asharov and Segev.

Proving impossibility seems more challenging when we look at the DDH-based IPFE schemes, e.g., Abdalla et al. [ABDP15]—the IPFEs can alternatively be based on the generic group model (GGM). As GGMs intuitively provide linear homomorphism, one might also attempt to construct IPFE from FHEs. Alternatively, recalling the earlier FE constructions from FHE and ABE [GKP+13], one might attempt to construct an FE for a smaller class that is secure against unbounded collusion.

1.1 Our Results

We answered our main question negatively—we ruled out any construction of unbounded-collusion IPFE from IBE, ABE, PE, or even Attribute-Based FHE. It is perhaps surprising because private-key and selective IPFE schemes are arguably the weakest non-trivial FE. Still, the constructions become impossible when unbounded collusion is required. Also, IPFEs can be based on relatively weaker assumptions, such as DDH, while basing FHE on DDH remains open after decades of research [RAD78, DH76, Mer78].

Conceptually, our separation shows that IPFE cannot be constructed from any "all-or-nothing" encryption schemes. The all-or-nothing concept is borrowed from the elegant work of Garg, Mahmoody and Mohammed [GMM17], henceforth GMM for short. Roughly speaking, an encryption scheme is all-or-nothing if one can either recover the whole plaintext given the "authorized" key, or learn nothing without the key. While the security definitions of IBE, ABE, PE, and FHE are all-or-nothing, in FE schemes, we require that the functional key holder learn exactly a function of the plaintext, which is neither the whole plaintext nor nothing. More formally, the result is proved and claimed in two different oracle worlds, where the first oracle captures IBE, ABE, and FHE.

Theorem 1.1 (Monolithic separation of IPFE from IBE, ABE, and FHE, informal)**.** *There is no construction of unbounded-collusion, private-key, and selectively secure inner-product functional encryption from Attribute-Based Fully Homomorphic Encryption (AB-FHE) in the "monolithic" model, elaborated below. The separation extends to all primitives implied by AB-FHE, including IBE, ABE, and FHE.*

The result extends beyond IBE, ABE, and FHE. Technically, we prove that there is no construction of IPFE in the ideal world where a variant of witness encryption oracle, called *Instance-Revealing Homomorphic Witness Encryption (IRHWE)*, exists. Roughly speaking, IRHWE is a witness encryption (WE), which consists of an encryption oracle and a decryption oracle. The encryption maps an instance and a plaintext to a ciphertext, and the decryption maps a witness and a ciphertext to the plaintext only when the witness is verified against the instance of the ciphertext. Moreover, by homomorphic, IRHWE provides another oracle that performs homomorphic evaluation on the same-instance ciphertexts. By instance revealing, each ciphertext is associated with its instance in the clear. The IRHWE oracle is proposed by GMM, who also showed that ABE, FHE, and even Attribute-Based FHE exist in the IRHWE oracle world.

The *monolithic model* is a natural extension of the black-box model. In traditional separation, cryptographic primitives are modeled as black-box oracles [IR89, RTV04]. However, in constructions based on primitive X for some X being FHE or ABE, X often takes a circuit as input, but the circuit itself invokes X, a non-black-box use of X. Indeed, many constructions rely on the capability of evaluating a circuit of the primitive itself, such as the bootstrapping of FHE. Fortunately, many natural constructions use circuits that only invoke X through oracle queries. Introduced by GMM [GMM17], the monolithic model captures such *oracle-aided* circuits, allowing a circuit to query the primitive as an oracle gate. That is, in the monolithic model, the constructions still query the primitive in a black-box way, except for using oracle-aided circuits. Moreover, any circuit-invoked oracle query may take in another oracle-aided circuit, and so on, which makes the lower bound stronger. GMM also justified that major non-black-box constructions fall within the monolithic model. In our result, we use the same monolithic IRHWE oracle, allowing FHE/ABE to perform arbitrary oracle-aided circuits.

The second oracle captures predicate encryptions.

Theorem 1.2 (Two-layer separation of IPFE from PE, informal)**.** *There is no construction of unbounded-collusion, private-key, and selectively secure inner-product functional encryption from predicate encryptions in the "two-layer black-box" model, elaborated below.*

Technically, we prove that there is no construction of IPFE in the ideal world such that another variant of witness encryption oracle, called *Instance-Hiding Witness Encryption (IHWE)*, exists. Similar to WE, IHWE consists of encryption and decryption oracles. Unlike IRHWE, we require that the instance of each ciphertext in IHWE must be hidden, i.e., similar to predicate encryptions, which hides the attributes (see [BW07]). Also, IHWE is not equipped with homomorphic computation. GMM showed that PE exists in the IHWE oracle world.

The *two-layer black-box* model of IHWE is an extension of the black-box model, but it is a weakening of the monolithic model of IHWE. Specifically, in the decryption oracle of IHWE, the witness verification may invoke IHWE itself in an oracle way, but the invoked oracle query cannot further invoke IHWE; equivalently, the oracle always invokes itself in a black-box way. Following GMM, using two-layer IHWE, a two-layer PE can be constructed. By two-layer, the PE can run a decryption in the verification step of another decryption, but the former decryption cannot further execute the PE itself. Our analysis was unsuccessful when we considered more layers. To the best of our knowledge, all applications of PE rely on oracle-aided circuits of at most two layers, e.g., the real-world applications [GVW15], such as searching encrypted data, use PE in the black-box way, which is one-layer. Hence, while we are unable to rule out FE constructions from deeper than two layers, it is also unclear how to utilize PEs of deeper layers.

For both oracles, our attacker rules out any IPFE scheme that achieves the weaker *indistinguishability-based* security. Given that *simulation-based* FEs are in general stronger than indistinguishability-based ones [BSW11], our result is stronger.

Optimality. Our result shows that the FE schemes of Goldwasser et al. [GKP+13] and Ananth and Vaikuntanathan [AV19] are optimal in the sense of bounded-collusion security, where both schemes fall in the monolithic/black-box model. In contrast, the adversary of Theorem 1.1 utilizes an *unbounded* number of colluded functional keys. Hence, by Theorem 1.1, it is impossible to extend the construction and achieve unbounded-collusion (IP)FE in the monolithic model.

Our result also complements the impossibility of constructing key agreements from private-key functional encryptions by Asharov and Segev [AS15]. Notice that the public-key encryptions imply key agreement [GKM+00], so private-key FE does not imply public-key encryptions. Meanwhile, we show that public-key encryptions do not imply private-key (IP)FE, and thus, they are mutually incomparable.

1.2 Related Works and Future Directions

It is well-known that private-key encryptions are separated from public-key encryptions (PKE) [IR89, RTV04], which are separated from IBEs [BPR+08] and ABEs for some predicates [KY09]. Also, IBEs are separated from ABE for threshold predicates [GKLM12]. In the sequence of primitives, each primitive is separated from the next, but the opposite direction is implied, often directly. Theorem 1.1 places IPFE between ABE and more-expressive FEs because IPFE is implied by FE, but we separated IPFE from ABE. While there is no construction of public-key ABE from private-key IPFE by Asharov and Segev [AS15], whether public-key IPFE implies ABE is open.

Theorem 1.1 extends a result of GMM [GMM17], which proved that assuming $\mathbf{NP} \not\subseteq \mathbf{coAM}$, IO is separated from any primitives that exist relative to the monolithic IRHWE oracle. To see why, recall that by $\mathbf{NP} \not\subseteq \mathbf{coAM}$, statistically secure IO does not exist [GR07]. Assuming for contradiction that computational IO exists in the IRHWE oracle world, then as IO implies FE [GGH+13], FE exists as well in the same world, which contradicts Theorem 1.1. Another result of GMM proved that IO is separated from the *monolithic* IHWE oracle, but it is incomparable to our separation of (IP)FE from *two-layer* PE (Theorem 1.2).

Some DDH-based IPFEs can be based on the generic group model, e.g., Abdalla et al. [ABDP15]. However, they are *restricted* in a small range of inner-product values; i.e., the inner product is defined over $\mathbb{Z}_p$ for p that can be exponentially large in the security parameter λ, but the correctness is only guaranteed when the inner produce is in a small range that is polynomial in λ. Castagnos, Laguillaumie and Tucker [CLT18] proposed an unrestricted IPFE based on a new algebraic assumption. As mentioned by Hajiabadi et al. [HLOW24], it is open whether restricted IPFE can be constructed in the random oracle model.

In addition to the unbounded collusion of functional keys, our attacker also corrupts an unbounded polynomial number of *ciphertexts*. The number of ciphertexts is necessary for private-key IPFEs—supposing that the number is a priori bounded by B, then there is an information-theoretically secure IPFE as follows. The encryption of a plaintext vector $\mathbf{m}$ is simply $c \leftarrow \mathbf{m} + \mathbf{r}$ for a random vector $\mathbf{r}$, and the functional key of vector $\mathbf{v}$ is simply the pair $(\mathbf{v}, \langle \mathbf{v}, \mathbf{r} \rangle)$, so that the decryption $\langle c, \mathbf{v} \rangle - \langle \mathbf{v}, \mathbf{r} \rangle$ follows by linearity. The master secret key consists of B such random vectors, a polynomial in the security parameter λ. In contrast, the previous work of Agrawal et al. [AGVW13] ruled out FE for a family of weak pseudo-random functions in the one-ciphertext and unbounded-collusion but simulation-based security and public-key setting. Their result is incomparable to ours as we rule out indistinguishability-based and private-key settings, and the function families differ.

A future direction is to strengthen the oracles. We proved that IPFE does not exist in the world with *one of the two* oracles, IRHWE or IHWE. While many known constructions rely on only one of the two, it is still open whether using both oracles helps to overcome our impossibility. In the monolithic model, two independent oracles may or may not call each other. We conjecture that the impossibility still holds if two oracles never call each other. It is entirely open when two oracles mutually call or even when they are dependent. Another question arises as multi-key Attribute-Based FHE exists in the "multi-key" IRHWE oracle model, as per GMM [GMM17], and one may ask whether IPFE is separated. We believe our impossibility extends to multi-key FHE. Particularly, the only difference is that the multi-key homomorphic evaluation oracle takes input ciphertexts encrypted by multiple keys, and then outputs ciphertexts encrypting secret shares of the evaluation result. Because our attacker already simulates the evaluation result, additionally simulating the shares is doable.

Ideally, we want to separate IPFE from the *monolithic* IHWE oracle, but we achieved only the two-layer one in Theorem 1.2 due to the probabilistic analysis, elaborated later in Section 2.6.

Finally, for some function family, the feasibility of FE is open. While (IP)FE is impossible for any function family that is a superset of inner product, our result says nothing about non-supersets. For example, FE for point functions can be based on one-way functions [HLOW24], but AC0, the class of constant-depth and polynomial fan-in circuits, is incomparable to inner products and remains open.

2 Technical Overview

In this overview, we will begin with recapping the witness encryption oracles in the monolithic model [GMM17]. We will also recap the recent impossibility on IPFE in the random oracle model, where Hajiabadi et al. provides a framework and a powerful combinatorial lemma [HLOW24]. With that, we give a warmup that extends the impossibility to the trapdoor permutation world, and then we illustrate the impossibility of IPFE from the stronger witness encryption oracles. On our way, some challenges arise from the fact that the "fancy" oracle answers different queries in a correlated way. Some of challenges arise from the fact that the monolithic oracles indirectly query itself. Our attacker will learn and simulate the answers to correlated and/or indirect queries.

2.1 Witness Encryption Oracles [GMM17]

We will consider IPFE schemes in either the Instance-Revealing Homomorphic Witness Encryption (IRHWE) or the Instance-Hiding Witness Encryption (IHWE) oracle world. The IRHWE and IHWE oracles are introduced by Garg, Mahmoody and Mohammed [GMM17], where they proved that attribute-based FHE exists in the IRHWE world, which implies the existence of both FHE and ABE, and they proved that predicate encryption exists in the IHWE world. At a high level, both IRHWE and IHWE model witness encryptions, and hence they provide suboracles to encrypt plaintexts and decrypt ciphertexts. While we use the terminologies "instance" and "witness" as in witness encryption, for readability, we will henceforth call the encrypt and decrypt suboracles as "permuting" a "preimage" into a "label" and "inverting" a label back to the preimage. That is, we reserve for IPFE the terminologies "encrypt/decrypt" a "plaintext/ciphertext" and the encryption/decryption "keys."

We summarize the IRHWE oracle below. Notice that both the homomorphic evaluator f and the witness verifier V are both oracle-aided and may call the IRHWE itself as an oracle.

Definition 2.1 (IRHWE oracle, informal)**.** *Let V be an oracle-aided universal circuit-evaluator that takes an instance-witness pair (a, w) outputs 1 for acceptance or 0 for rejection. For every security parameter $\lambda \in \mathbb{N}$, the* instance-revealing homomorphic witness encryption *(IRHWE) oracle consists of a private random function $H : \{0,1\}^* \to \{0,1\}^\lambda$, a set of private and independently random permutations $\{\pi_a : \{0,1\}^{2\lambda} \to \{0,1\}^{2\lambda}\}_{a \in \{0,1\}^*}$, and the following suboracles.*

- *$e(a, x)$ is a permuting suboracle that maps a preimage $x \in \{0,1\}^\lambda$ to a label $\mathsf{lb} \in \{0,1\}^*$: it permutes $y \leftarrow \pi_a(x \| H(a \| x))$ using the permutation π_a of the instance, and then outputs the label $\mathsf{lb} = (a, y)$ as a pair.*

- *$eval(f, \mathsf{lb}_1, \mathsf{lb}_2, \ldots)$ is a homomorphic evaluation suboracle, and it outputs another label lb. It parses the input labels as $\mathsf{lb}_i = (a_i, y_i)$ for each i. If any two inputs are associated with different instances, i.e., there exists $\mathsf{lb}_{i_1} = (a_{i_1}, y_{i_1}), \mathsf{lb}_{i_2} = (a_{i_2}, y_{i_2}), a_{i_1} \neq a_{i_2}$ for some $i_1 \neq i_2$, then reject and output $\perp$. Otherwise, $a_i = a$ for all i, it finds the preimages $x_1, x_2, \ldots$ of the input labels, i.e., x_i is the λ-bit prefix of $\pi_a^{-1}(y_i)$; then, it evaluates the concatenated string $x_1 \| x_2 \| \ldots$ as an* oracle-aided *boolean circuit, where $f^{(\cdot)}$ may perform oracle queries to the IRHWE itself. Let x' be the result of evaluating f, permuted x' to y by $y \leftarrow \pi_a(x' \| H(f \| a \| y_1 \| y_2 \| \ldots))$, and the output is the label $\mathsf{lb} := (a, y)$.*

- *$d_{\mathsf{V}}(w, \mathsf{lb})$ is an inverting suboracle that uses a witness w to recover the preimage of the label lb if w is a witness of the instance of $\mathsf{lb} = (a, y)$. That is, if $\mathsf{V}^{(\cdot)}(a, w) = 1$, then output the λ-bit prefix of $\pi_a^{-1}(y)$, where $\mathsf{V}^{(\cdot)}$ can perform oracle queries to the IRHWE itself; otherwise output $\perp$.*

We note that in the $eval$ suboracle, the permutation π_a rerandomizes the label even when the preimage is the same. The formal IRHWE is given in Definition 3.14, which differs from the above by that the preimage $x \in \{0,1\}^\lambda$ is mapped to 3λ bits by π_a—such sparse mappings ensure that the inversion $d_{\mathsf{V}}(\star, \mathsf{lb})$ is successful only if the label lb is obtained from the oracle.

As argued by GMM, random and sparse labels are necessary for constructing attribute-based FHE [GMM17, Section 8.3]. Roughly, the oracles $(e, eval, d_{\mathsf{V}})$ captures the encryption, homomorphic evaluation, and decryption of FHEs. To see why, for uniformly random secret key sk, we generate the public key $\mathsf{pk} \leftarrow e(\mathsf{sk})$; we use pk as the instance in the encryption e and homomorphic evaluation $eval$, and we use sk as the witness in the decryption d_{V}. Notice that V is oracle-aided in this example as V queries and checks if $e(\mathsf{sk}) = \mathsf{pk}$. It is also necessary in general—if not, V never queries the oracle itself, then given the instance, an unbounded-time adversary can find a legitimate witness without querying the oracle, which breaks the encryption.

The *instance-hiding* witness encryption (IHWE) oracle differs from the above IRHWE in a few places: there is no homomorphic evaluation, and the instance is hidden inside a label, and the oracle can only call itself once.

Definition 2.2 (Two-layer IHWE oracle, informal)**.** *Let V be an oracle-aided universal circuit-evaluator that takes an instance-witness pair (a, w) outputs 1 for acceptance or 0 for rejection. For every security parameter $\lambda \in \mathbb{N}$, the* instance-hiding witness encryption *(IHWE) oracle consists of a set of private and independently random permutations $\{\pi_\alpha : \{0,1\}^{\alpha+\lambda} \to \{0,1\}^{\alpha+\lambda}\}_{\alpha\in\mathbb{N}}$, and the following suboracles.*

- *$e(a, x)$ is a permuting suboracle that maps a preimage (a, x) for $a \in \{0,1\}^\alpha$ and $x \in \{0,1\}^\lambda$, where $\alpha = |a|$ is the length of a, to a label lb: it permutes and outputs $\mathsf{lb} \leftarrow \pi_\alpha(a\|x)$.*
- *$d_{\mathsf{V}}(w, \mathsf{lb})$ is an inverting suboracle that uses a witness w to recover the preimage of the label $\mathsf{lb} \in \{0,1\}^{\alpha+\lambda}$ if w is a witness of the instance and $\alpha \in \mathbb{N}$. That is, let $(a, x) \leftarrow \pi_\alpha^{-1}(\mathsf{lb})$, if $\mathsf{V}^{(\cdot)}(a, w) = 1$, then output (a, x); otherwise, output $\perp$, where $\mathsf{V}^{(\cdot)}$ can perform oracle queries to $\{e, d_{\mathsf{V}}^0\}$ but not d_{V} itself.*
- *$d_{\mathsf{V}}^0(w, \mathsf{lb})$ is similar to d_{V}, but the only difference is that V is a regular universal circuit-evaluator that cannot perform oracle queries.*

Notice that if we allowed d_{V} to perform oracle queries to itself, the above would be the monolithic IHWE as defined in GMM. Following GMM [GMM17, Section 8.1], a predicate encryption (PE) can be constructed in the IHWE world; moreover, for each $l \in \mathbb{N}$, l-layer PE exists in the l-layer IHWE world. W.l.o.g., we assume that the IPFE scheme never queries d_{V}^0 directly in this overview; we also abuse notation and denote d_{V}^0 as d_{V} when the context is clear. We remark that, unlike the instance-revealing variant, without querying d_{V} using a legitimate witness, an adversary cannot learn the instance of a IHWE label. Hence, both one-layer and two-layer IHWEs and the corresponding PEs are non-trivial.

Both IRHWE and IHWE oracles call the oracle itself, referred to as *indirect queries*. Observed by GMM, since each indirect query takes the description of a circuit as input, the total number of indirect queries is bounded by the circuit size of the initial direct query.

2.2 IPFE and a Framework of Attackers [HLOW24]

We consider IPFE schemes that are parameterized by $\mathbb{Z}_p^n$ for some modulus p and dimension n and consist of three algorithms, $\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}$. The master secret key msk is a uniformly random string of $\mathrm{poly}(\lambda)$ bits for security parameter λ; w.l.o.g., if a setup is needed to sample msk, we let msk be the randomness of the setup. For a key vector $\mathbf{v} \in \mathbb{Z}_p^n$, the functional key is generated by

$\mathsf{sk}_{\mathbf{v}} \leftarrow \mathsf{KGen}(\mathsf{msk}, \mathbf{v})$. For a message vector $\mathbf{m} \in \mathbb{Z}_p^n$, the ciphertext is encrypted by the private key, $c \leftarrow \mathsf{Enc}(\mathsf{msk}, \mathbf{m})$. The functional decryption is performed by $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}}, c)$, which is required by correctness to be the inner product $\langle \mathbf{v}, \mathbf{m} \rangle$. Assume for simplicity that KGen is a deterministic algorithm; it is a weak assumption and will be removed later, see Footnote 3.

The unbounded-collusion and selective security is defined by the following game. The adversary chooses and sends the colluded vectors $(\mathbf{v}_1, \ldots, \mathbf{v}_t)$ for some unbounded t and the challenge vector $\mathbf{v}$ to the challenger. The challenger requires that the challenge $\mathbf{v}$ is not in the linear span of $(\mathbf{v}_1, \ldots, \mathbf{v}_t)$ to prohibit trivial attacks, and it generates the colluded functional keys $(\mathsf{sk}_{\mathbf{v}_1}, \ldots, \mathsf{sk}_{\mathbf{v}_t})$ using a fresh master secret key msk; the challenger also samples a challenge ciphertexts by $c \leftarrow \mathsf{Enc}(\mathsf{msk}, \mathbf{m})$ for uniformly random message $\mathbf{m} \leftarrow \mathbb{Z}_p^n$. Given the colluded functional keys and the challenge c, the adversary is asked to distinguish between the inner product $\langle \mathbf{v}, \mathbf{m} \rangle$ and a uniform random value in $\mathbb{Z}_p$. We remark that the above security of random message is implied by standard chosen-plaintext-attack security, making the impossibility stronger. The adversary may request an arbitrary number of challenge ciphertexts.

Framework: separating private-key IPFE and random oracle. Hajiabadi et al. demonstrated a PSPACE attacker that breaks private-key IPFEs in the random oracle model using polynomially many oracle queries [HLOW24]. Their attacker consists of three phases:

1. (Colluding phase.) This is exactly the security game with parameters t, t_c. Sample random vectors $\mathbf{v}$ and $\mathbf{v}_1, \ldots, \mathbf{v}_t$ for a sufficiently large t, subject to the constraint that $\mathbf{v}$ is not in the span of $(\mathbf{v}_1, \ldots, \mathbf{v}_t)$. Get from the challenger the colluded functional keys $\mathsf{sk}_{\mathbf{v}_1}, \ldots, \mathsf{sk}_{\mathbf{v}_t}$ as well as the challenge ciphertext and product pairs $(c_i, z_i)_{i \in [t_c]}$. Recall that c_i encrypts a plaintext $\mathbf{m}_i$, and z_i is either the inner product $\langle \mathbf{v}, \mathbf{m}_i \rangle$ or an independent and fresh random value, and the goal is to distinguish whether the z_i's are random.

2. (Learning phase.) Emulate IPFE decryption using each of $\mathsf{sk}_{\mathbf{v}_1}, \ldots, \mathsf{sk}_{\mathbf{v}_t}$ on each of the challenge ciphertext c_i; record in Q the query-answer (Q-A) pairs made on the random oracle.

3. (Simulation phase.) Exhaustively enumerate each string $\mathsf{sk}'_{\mathbf{v}}$ and each list Q' of Q-A pairs that occurred when generating $\mathsf{sk}'_{\mathbf{v}}$, simulate IPFE decryption using $\mathsf{sk}'_{\mathbf{v}}$ on c_i; whenever the decryption performs an oracle query, find the answer in the list $Q \cup Q'$, or sample a random answer if not found.[2] If there is a pair $(\mathsf{sk}'_{\mathbf{v}}, Q')$ such that functionally decrypts c_i to the challenge product z_i for all $i \in [t_c]$, then the attacker outputs 'Yes' in the security game; otherwise, it outputs 'No'.

Notice that the attacker only queries the oracle during the learning phase, whereas the simulation phase needs PSPACE but no oracle query. Hence, the number of oracle queries is polynomial.

To prove that the attacker distinguishes the inner product from a random value, Hajiabadi et al. proved and employed the following combinatorial lemma, a combinatorial coverage bound [HLOW24], henceforth called HLOW Lemma.

Lemma 2.3 (HLOW Lemma [HLOW24, Lemma 4.1])**.** *For any $\lambda \in \mathbb{N}$, let $n := n(\lambda) \geq 3$ be a polynomial of λ, p be a prime such that p^{-n} is negligible in λ. For any polynomial $\ell := \ell(\lambda)$, any function*

[2] In this paper, we use the terms "Q-A list" and "Q-A set" interchangeably.

$F : \mathbb{Z}_p^n \to 2^{[\ell]}$, where $2^{[\ell]}$ is the power set of $[\ell]$, and any constant τ, there exists $t = \mathrm{poly}(\lambda)$ such that

$$\Pr\left[F(\mathbf{v}) \subseteq \bigcup_{i \in [t]} F(\mathbf{v}_i)\right] \geq 1 - \lambda^{-\tau},$$

where random variables $\mathbf{v}, \mathbf{v}_1, \ldots, \mathbf{v}_t$ are sampled by: uniformly sample vector $\mathbf{v} \leftarrow \mathbb{Z}_p^n$, sample a random $(n-1)$-dimensional subspace S such that $\mathbf{v} \notin \mathsf{S}$, and then sample vectors $\mathbf{v}_1, \ldots, \mathbf{v}_t \in \mathsf{S}$ uniformly at random.

HLOW Lemma states that for a randomly chosen target vector $\mathbf{v}$ and a randomly chosen high-dimensional subspace S such that $\mathbf{v} \notin \mathsf{S}$, any "information" that is conveyed by the target vector will also be conveyed by polynomially many random vectors in the subspace S, where the function F models the information conveyor.

Returning to the attacker, to understand why it works, the crux is to suppose that $(\mathsf{sk}'_{\mathbf{v}}, Q')$ is identical to the challenge functional key $\mathsf{sk}_{\mathbf{v}} \leftarrow \mathsf{KGen}(\mathsf{msk}, \mathbf{v})$ and the Q-A pairs. Moreover, it is guaranteed that for any query q made when encrypting $\mathbf{m}_i$ to c_i, if the same q is also queried during the decryption $(\mathsf{sk}'_{\mathbf{v}}, c_i)$, then q must be learned in Q w.h.p. The guarantee comes from HLOW Lemma. Call a query "essential" when it is made during both the encryption *and* decryption of c_i w.r.t. the functional secret key $\mathsf{sk}_{\mathbf{u}}$ for some vector $\mathbf{u}$. Fix the random oracle, msk, and c_i, and recall that KGen is deterministic.[3] Because $\mathsf{sk}_{\mathbf{u}}$ is a function of $\mathbf{u}$, the set of essential queries is a function; let $F(\mathbf{u})$ be the set. By HLOW Lemma, it holds that $F(\mathbf{v}) \subseteq \bigcup_{j \in [t]} F(\mathbf{v}_j)$ w.h.p. This concludes the guarantee as $\bigcup_{j \in [t]} F(\mathbf{v}_j) \subseteq Q$. With $Q \cup Q'$, the simulated response is statistically close to the real oracle, and hence the simulated decryption is identical to z_i if z_i is the inner product. Finally, when z_i is an independent random value, the repetition of ciphertexts c_i suppresses false positives because any $(\mathsf{sk}''_{\mathbf{v}}, Q'')$ and any simulation hits z_i for sufficiently many $i \in [t_c]$ with negligible probability.

2.3 Warmup: Separate IPFE from Trapdoor Permutations

It is tempting to apply HLOW Lemma and the framework to idealized models beyond random oracles. Indeed, HLOW Lemma holds for *any function F*, and one can define F to be the query-answer of any oracle. However, even HLOW Lemma holds, the attacker may not. It is because in other oracles, e.g., trapdoor permutations, the answers to distinct queries are *correlated*, while the answers were *independent* in random oracles. We illustrate the challenge and our solution in the trapdoor permutations (TDP) model.

A trapdoor permutation oracle consists of three suboracles: (1) $g(\tau)$ is a random function that maps a trapdoor τ to a key π; (2) $e(\pi, x)$ takes a key π as well as a preimage x and outputs a label, where for each π, $e(\pi, \cdot)$ is a random permutation; and (3) $d(\tau, \mathsf{lb})$ takes a trapdoor τ as well as a label lb and outputs the preimage of lb in the permutaion $e(\pi, \cdot)$ for some π satisfying $g(\tau) = \pi$.

Challenge 1: Correlated queries in trapdoor permutations. To separate IPFE from TDP, the attacker needs to break any IPFE with only polynomially many queries to TDP oracle. However, it

[3] We also fix in F the randomness of $c_i \leftarrow \mathsf{Enc}(\cdots)$. If Dec is randomized, fix the randomness in F as well. If KGen is randomized, one can obtain the randomness using $\mathbf{u}$ and another random function in F—Lemma 2.3 still holds w.h.p. because each $\mathbf{u}$ is distinct w.h.p. See Remark 3.24.

is insufficient to apply HLOW Lemma directly using the intersection of the Q-A lists. It is because there is a correlation between some e-queries and d-queries: an e-query's answer could completely determine a d-query's answer. An example is that the IPFE encryption only queries e, while the IPFE decryption only queries d. In this example, the intersection of two Q-A lists is empty. Hence, by directly applying HLOW Lemma, the issue is that the intersection of the "raw" Q-A lists does not fully capture the *essential information* about the IPFE encryption. The information from the Q-A list obtained through the learning phase, where IPFE decryption is performed using the colluded functional keys, may be useful; however, a raw Q-A list does not explicitly include all possible forms of queries, as seen in the example.

Solution 1: Closure of query-answer pairs. We address the issue by extending the Q-A lists to all forms of the correlated Q-A pairs. This is inspired by Goyal et al. [GKLM12], and we follow their terminology of "closure." The closure of a Q-A list is a superset of the list that includes all Q-A pairs that can be directly inferred from the original list. Namely, when there is a g-query that relates a permutation π and a trapdoor τ, if there is also an e-query that uses the same π, append the corresponding d-query into the closure; symmetrically, append an e-query if there is a corresponding d-query. It is involved, but the closure of a Q-A list is a *function of* $\mathbf{u}$ if the original Q-A list is a function of a vector $\mathbf{u}$. Hence, HLOW Lemma applies to the closures. Elaborately, the function $F(\mathbf{u})$ is defined to be the intersection between the two closures of each Q-A list of IPFE encryption and IPFE decryption with the functional key $\mathsf{sk}_{\mathbf{u}}$. The attacker simulates the answers intuitively: use the closure of the Q-A list learned from the colluded functional keys if the incoming query is in the list; otherwise, sample and answer randomly, while ensuring consistency with the previous answers. With care, we can show that the attacker can correctly simulate the answers except with $1/\mathrm{poly}(\lambda)$ probability.

A minor concern arises from the bijective property of TDPs. In the simulation phase of the attacker, a randomly sampled answer y to query x might collide with another query-answer pair (x', y) performed during IPFE encryption. Suppose that the IPFE encryption never uses the query-answer (x', y), and thus the attacker can never learn it. Still, the collision violates the bijection, and thus the simulation differs from the real TDP oracle. Fortunately, we argue that a collision occurs with negligible probability because each algorithm performs only a polynomial number of queries. This concludes the separation of TDP.

Extending the separation to attribute-based encryptions. The technique extends to separate IPFE from stronger primitives such as ABE in the black-box model, where the oracle evaluates plain circuits. The proof is almost identical because the attributes and their predicates can be manipulated by a closure similar to the permutation. Although this subsection serves as a warmup, both the techniques and results are novel to the best of our knowledge.

2.4 Separate IPFE from Attribute-Based FHE

Extending the impossibility to FHE seemed hopeless because the DDH-based IPFEs can alternatively be based on generic group models (GGM) [ABDP15, ABDP16, ALS16]. To this end, we observe that all DDH-based IPFEs are restricted in the same way: the correctness holds only when the inner product is within a polynomial-sized range, referred to as "restricted" in the literature. Moreover, the decryption directly tests the equality between a label and a short list of known labels in the GGM. That is, such DDH-based IPFEs encrypt a vector on the exponents of group elements, and then the inner product is computed by linear homomorphism on the exponent. Using the same functional key, the decryption recovers the identical group elements for any ciphertext that results in the same inner product value. It follows that testing equality for a small set of group elements suffices for restricted correctness.

Hence, such an approach of IPFEs cannot be based on FHE. Had we employed FHE instead of DDH to perform the linear homomorphic evaluation, using the same functional key, each vector-ciphertext would be rerandomized to a different FHE ciphertext; thus, the equality test wouldn't work even when the inner product value is identical. Notice that for either DDH-based or FHE-based IPFEs, the functional key must not include the secret key of the corresponding DDH or FHE, otherwise the IPFE would be insecure. Based on this observation, the DDH-based IPFEs are no longer a barrier toward impossibility. Indeed, as formalized in the monolithic oracle IRHWE (Section 2.1), the labels are always rerandomized in the homomorphic evaluation. Considering proper homomorphic evaluation resolves the above issue, but it also introduces challenges.

Recall that FHE can be constructed in the IRHWE world (Section 2.1). To separate IPFE from FHE, we will prove the stronger statement that there is no IPFE in the IRHWE world, which also rules out attribute-based FHE.

Challenge 2: More correlation due to homomorphic evaluation. Comparing trapdoor permutation (TDP) and IRHWE oracles, the homomorphic evaluation $eval$ suboracle is a significant augmentation. Moreover, when looking at the separation of TDP, we soon observe that the "closure" of queries in TDP no longer holds. In a TDP, the closure is defined by the label and preimage pairs, which form a bijection; moreover, distinct pairs are almost independent except with negligible probability. However, with homomorphic evaluation in IRHWE, we do not know how to define the closure of a label or a preimage when an $eval$ query homomorphically evaluates some input labels and outputs another label, no matter whether we know some of their preimages. In an IPFE scheme that makes many $eval$-queries, each taking input from the output of another $eval$-query, many distinct labels and their preimages become correlated. It seems that a useful "closure" would be too large to be learned.

A simple example that cannot be resolved using the closure technique is given below. Consider an IPFE scheme $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ that incurrs the following queries and answers, where $\star$ denotes an arbitrary input/output.

- $c \leftarrow \mathsf{Enc}(\star)$ queries $\mathsf{lb}_1 \leftarrow e(\star)$ and gets a label lb_1.
- $\mathsf{Dec}(\star, c)$ queries $\mathsf{lb}_2 \leftarrow eval(\mathsf{lb}_1, \mathsf{lb}')$ for some lb' and gets a homomorphically evaluated label lb_2, and then it queries $d_{\mathsf{V}}(w, \mathsf{lb}_2)$ for some witness w.

To see why the simple closure does not work, recall that in the simulation phase of the framework (Section 2.2), an attacker must correctly answer the query $d_{\mathsf{V}}(w, \mathsf{lb}_2)$. Notice that the answer depends on lb_2, which depends on the preimage of lb_1 (and lb'). There are two strawman approaches to learn it. The first is to learn the preimage of lb_1 in the learning phase. That is unlikely if in the learning phase, lb_1 has never been an output of e-query or an input of d-query; e.g., lb_1 is always an input of an $eval$-query of $\mathsf{Dec}(z, c)$ for all functional keys. The second is to learn the preimage of lb_2. That is also unlikely if the label lb' depends totally on the challenge functional secret key. That is, if lb' is provided only by $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}}, c)$ but not $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}_j}, c)$ for any colluded functional key $\mathsf{sk}_{\mathbf{v}_j}$, then the query $eval(\mathsf{lb}_1, \mathsf{lb}')$ and thus the d-query won't appear in the learning phase. Thus, neither approaches work.

Solution 2: Learn the preimages whenever a witness is given. From the above example, our observation is that, no matter how IPFE manipulates labels through $eval$-queries, a *legitimate witness* is always explicitly used in a successful d-query on at least one of the correlated labels if they carry meaningful information. For any correct and secure IPFE, the encryption procedure must encode a plaintext vector into some "essential" labels that are subsequently manipulated by the functional decryption. Since the set of essential labels is a *function of the challenge vector*, by HLOW Lemma, the colluded functional decryptions in the learning phase cover all essential labels.

With the observation, our attacker collects the witnesses that invert any label successfully in a d-query in the learning phase. Recall that each label lb is associated with an instance a in the form $\mathsf{lb} = (a, \cdot)$, and that the instance is the same for all correlated labels. Hence, whenever there is a successful query $d_{\mathsf{V}}(w, \mathsf{lb} = (a, y))$ for some (a, w) pair in the learning phase, the attacker additionally queries $d_{\mathsf{V}}(w, \mathsf{lb}' = (a, y'))$ and learns the preimage for all lb' of the same instance a. Since (a, w) is legitimate in the former d-query, the latter d-query must also be successful. Hence, the attacker is modified as follows.

2. (Learning phase.) Emulate IPFE decryption using each of $\mathsf{sk}_{\mathbf{v}_1}, \ldots, \mathsf{sk}_{\mathbf{v}_t}$ on each of the challenge ciphertext c_i; record in Q the query-answer (Q-A) pairs made on the IRHWE oracle (similar to Section 2.2, and the only difference is using IRHWE).
   For each d-query of form $d_{\mathsf{V}}(w, \mathsf{lb} = (a, y))$ that inverts to a preimage successfully in Q,
   (a) For each $\mathsf{lb}' = (a, y')$ that appeared in Q for some y', query the IRHWE oracle with $d_{\mathsf{V}}(w, \mathsf{lb}')$, and then append the query-answer pair to Q.

   Stop the for-loop when there is no new query-answer pair that can be appended to Q.

Since the IPFE performs a polynomial number of oracle queries, the number of witnesses and labels is polynomial. Moreover, the for-loop performs at most one additional oracle query per label-witness pair. Thereby, the number of queries remains polynomial. The additional learning facilitates analysis through HLOW Lemma—we can define the function $F(\mathbf{u})$ to be the set of essential instances where a legitimate witness is needed, i.e.,

- $F(\mathbf{u})$: Output the set of instances a such that
  - there exists an $\mathsf{lb} = (a, \star)$ that is the output of an e-query or $eval$-query during the challenge ciphertext $c \leftarrow \mathsf{Enc}(\cdot)$ is encrypted, and
  - there exists an $\mathsf{lb}' = (a, \star)$ that is inverted successfully by a query $d_{\mathsf{V}}(\star, \mathsf{lb}')$ during the decryption $\mathsf{Dec}(\mathsf{sk}_{\mathbf{u}}, c)$.

Plugging such F into HLOW Lemma solves the example. To see why, suppose that the query $d_{\mathsf{V}}(w, \mathsf{lb}_2)$ is inverted successfully during the real $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}}, c)$ using the real functional key $\mathsf{sk}_{\mathbf{v}}$ and challenge c. By the definition of the above F, the instance a of lb_2 is in $F(\mathbf{v})$, and by HLOW Lemma, $a \in F(\mathbf{v}_j)$ for some colluded key $\mathsf{sk}_{\mathbf{v}_j}$, which implies that a legitimate witness w' for a is learned from the functional decryption using $\mathbf{v}_j$. By the additional learning, all the preimages of the same-instance labels are learned. Hence, since lb_1 comes from Enc and is homomorphically evaluated into lb_2, the instance of lb_1 is the same as lb_2, and the preimage of lb_1 is learned through the additional learning. With that, the attacker can further compute the preimage of lb_2.

Informally, the above argument separates IPFE from black-box use of FHE. That is, we augment IRHWE with a mapping that generates the public key from a secret key, where the e, $eval$, and d_{V} suboracles correspond to encryption, homomorphic evaluation, and decryption, the witness and instance correspond to the secret key and public key; we disallow the oracle to call itself, and the verification of witness is restricted to the secret-public-key mapping.

Notice that the previous closure technique does not perform additional oracle queries. To the best of our knowledge, our solution is novel—both Garg, Mahmoody, and Mohammed [GMM17] and Hajiabadi et al. [HLOW24] rely only on some variants of heavy queries directly from the construction, without additional queries.

It remains to resolve the next challenge, which arises as the monolithic oracle may call itself.

Challenge 3: Indirect queries in the monolithic model. In a monolithic model, the oracle can query the oracle itself, but the query and its answer are not observable when the attacker emulates the IPFE procedure. Such queries made by the oracle are called *indirect*, while the attacker observes only the direct query-answers. However, in the simulation phase, the attacker must simulate indirect queries as well, because the answer to a direct query can depend on the corresponding indirect queries. To show the issue, the example of Challenge 2 is extended below.

- $c \leftarrow \mathsf{Enc}(\star)$ queries $\mathsf{lb}_1 \leftarrow e(\star)$ and gets a label lb_1 (same as the previous).
- $\mathsf{Dec}(\star, c)$ queries $\mathsf{lb}_2 \leftarrow eval(\mathsf{lb}_1, \mathsf{lb}')$ for some lb' and gets another homomorphically evaluated label lb_2, but the $eval$ suboracle evaluates an oracle-aided circuit $f^{(\cdot)}$, which *indirectly queries* $\mathsf{lb}_3 \leftarrow e(\star)$. Then, Dec queries $d_{\mathsf{V}}(w, \mathsf{lb}_2)$ for some witness w.

Observe that the answer $\mathsf{lb}_3 \leftarrow e(\star)$ affects the answer lb_2 to $eval$ and thus the answer to $d_{\mathsf{V}}(w, \mathsf{lb}_2)$. For example, the evaluation of f can be whether lb_3 equals a value. Thus, the preimage of lb_2 depends on learning and answering some indirect queries. Hence, the attacker needs to learn the preimage $e(\star) = \mathsf{lb}_3$ in the learning phase, even though the query is indirect.

The indirect queries also occur in d-queries, and learning some of them is necessary. Recall that d_{V} suboracle performs the oracle-aided verifier $\mathsf{V}^{(\cdot)}(a, w)$, which can call oracle itself. In the above example, even if the attacker already inferred the preimage of lb_2, it still needs to infer the verification $\mathsf{V}(a, w)$ to correctly answer $d_{\mathsf{V}}(w, \mathsf{lb}_2)$—depending on the witness w, the d_{V} shall answer the preimage only when w is verified. While a is given from lb_2 and fixed, recall that given the instance a, there can be exponentially many distinct but legitimate witnesses, which makes it uncertain to learn $\mathsf{V}(a, w)$. In general, the indirect queries of monolithic models make proving impossibility more challenging.

Solution 3: Recurrent canonicalization. We address Challenge 3 using an idea by Garg, Mahmoody, and Mohammed [GMM17], called *canonicalization*. The idea is to compile an oracle-aided algorithm $A^{\mathcal{O}}$ into another algorithm $\overline{A}^{\mathcal{O}}$, so that $\overline{A}$ is functionally identical to A, but $\overline{A}$ additionally queries $\mathcal{O}$ for each indirect query that can be "inferred" from the execution of $A^{\mathcal{O}}$. The hope is that indirect queries become direct in the canonical form, allowing an attacker to learn them. As mentioned in Section 2.1, due to the circuit size, the total number of indirect queries is at most a polynomial in the number of direct queries. Thus, canonicalization preserves the number of queries, up to a polynomial factor. The canonicalization of IPFE also preserves the security as it executes each individual algorithm as an oracle-aided black box.

Still, we need to pin down the *inferable* indirect queries and to argue that they are sufficient for the attacker. To infer an indirect query, we must learn which suboracle and which oracle-aided circuit invoked the query. We start from the inverting d suboracle. It is straightforward to infer the oracle-aided verification of any $d_{\mathsf{V}}(w, \mathsf{lb})$ query because the instance and witness are given in the clear—the witness is w, and the instance a is parsed by $\mathsf{lb} = (a, y)$, where y is a fixed-length suffix of 2λ bits. Performing $\mathsf{V}(a, w)$ turns the indirect queries into direct ones.

To infer the indirect queries made by an $eval$-query, the canonicalization needs the oracle-aided circuit $f^{(\cdot)}$, which is described by the preimages of the input labels of the $eval$. A legitimate witness is needed to invert the input labels, which depends on the execution context. Specifically, the witness is provided to a d-query, which may appear somewhere else in the execution $A^{\mathcal{O}}$; namely, in the above example, witness w shows up after the $eval$. To this end, our canonicalization compiler keeps a list of Q-A pairs it has made so far. Whenever the compiler observes a successful d-query so that a new legitimate witness w is learned, the compiler makes additional d-queries using w to invert all same-instance labels in its Q-A list, and it *goes back to the preceding $eval$-queries* in A, obtains, and evaluates the circuit $f^{(\cdot)}$ using the newly inverted preimages. Hence, the compiler turns the indirect queries of $eval$ to direct ones.

Finally, since an indirect query may invoke another deeper indirect query, our compiler *recurrently* performs the above to discover deeper indirect queries until there are no new queries. The canonicalization is given in Algorithm 1.

Algorithm 1. $\mathsf{Canonicalize}^{\mathcal{O}}(A)$, informal.

Input: An oracle-aided algorithm $A^{(\cdot)}$, which may include the input to A implicitly.
Oracle: An IRHWE oracle $\mathcal{O}$ is given.
Procedure: Initialize an empty list Q. Emulate $A^{(\cdot)}$ and respond to every (direct or indirect) query q as follows:

1. If q is $eval$-type of the form $eval(\mathsf{lb}_1, \cdots, \mathsf{lb}_l)$ for some l:
   (a) If the preimage x_i of each lb_i is learned from Q for all i, let $f := x_1 \| \ldots \| x_l$, and emulate $f^{(\cdot)}$; respond to all queries recursively using this procedure and the list Q.
2. If q is d-type of the form $d_{\mathsf{V}}(w, \mathsf{lb} = (a, y))$:
   (a) Emulate $\mathsf{V}^{(\cdot)}(a, w)$; respond to all queries recursively using this procedure and the list Q. Let b be the result.
   (b) (*Go back to previous $eval$ after learned a new legitimate witness.*) If $b = 1$ and the preimage of lb is not learned in Q, but there exists a query $eval(\mathsf{lb}_1, ..., \mathsf{lb}_l)$ that is associated with the answer lb in Q, then perform the following.
      i. For all same-instance labels $\mathsf{lb}' = (a, \cdot)$ appeared in Q, if the preimage of lb' is not yet learned in Q, query oracle $d_{\mathsf{V}}(w, \mathsf{lb}')$ to invert the label, and append the query and the preimage into Q.

   ii. For all $eval$ queries in Q, if the previous step newly obtained the preimages of the input labels of the $eval$ query, recursively invoke Step 1 on the $eval$ query using the list Q.

3. For any type of q, query oracle $a \leftarrow \mathcal{O}(q)$, and add the Q-A pair (q, a) into Q.

Finally, the canonicalization outputs the output of the emulated $A^{(\cdot)}$.

The canonicalization is applied to each IPFE algorithm, i.e., the attacker breaks $\mathsf{Canonicalize}(A)$ for $A \in \{\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}\}$ rather than the original IPFE. Particularly, in the learning phase, the attacker emulates the following

$$(\mathsf{Canonicalize}(\mathsf{Dec}))\,(\mathsf{sk}_{\mathbf{v}_j}, c_i) \quad \text{for all } i \in [t_c], j \in [t]$$

and learns all the oracle query-answers during the emulation. We remark that the canonicalization subsumes the "additional learning" in the solution to Challenge 2—it is because prior to going back and entering a previous $eval$ query, Canonicalize needs to invert and obtain the preimages of relevant labels, which is the same as additional learning.

The Canonicalize preserves the correctness and security of IPFE. With the "going back" trick, we need to argue that Canonicalize indeed halts in polynomial time and number of oracle queries, and we defer the analysis to Section 4. We note that the "going back" trick is novel compared to previous works.

Summary of our attacker and novelty. With the above techniques, our polynomial-query attacker Brk is summarized as follows. In the learning phase, Brk emulates IPFE decryption using the colluded functional keys, and it learns the yielded Q-A list from the real oracle. The decryption is canonicalized as per Solution 3 prior to the emulation, so Brk learns the aforementioned additional preimages and indirect queries. Then, in the simulation phase, as in the framework of Hajiabadi et al., Brk exhausts all key-generation of IPFE and attempts to simulate the functional decryption on the challenge ciphertext. The answers to the queries are simulated, similar to Solution 1 for trapdoor permutations, but Brk recursively simulates the circuit evaluation associated with $eval$ and d-queries whenever the circuit can be inferred, which is also canonicalization. If the answer to a query remains unclear, Brk responds with a random sample or $\perp$ accordingly. The simulation of answers is illustrated in Algorithm 2.

Algorithm 2. Simulator of IRHWE oracle, informal.

Initialization: A state S_Q is initialized to the Q-A list $Q \cup Q'$, where Q is learned from the learning phase, and Q' is the Q-A pairs of the exhaustively enumerated functional key.
Answer a query q: Once initialized, the simulator takes an input query q, simulates an answer to q using the following procedure, and then the current Q-A pair is appended to the state S_Q, so that a future answer is consistent with the current answer.

1. If q is identical to a query of a Q-A pair in S_Q, return the corresponding answer. Otherwise, proceed with the steps below.

2. If q is an e-query of the instance a, uniformly sample y that is unassociated with the

same instance a in S_Q, and then answer $\mathsf{lb} = (a, y)$.

3. If q is an $eval$-query,
   (a) If any two input labels are associated with different instances, answer $\perp$.
   (b) Otherwise, let a be the same instance of all labels, uniformly sample y that is unassociated with the same instance a in S_Q, and then answer $\mathsf{lb} = (a, y)$.
4. If q is a d-query of the form $d_{\mathsf{V}}(w, \mathsf{lb} = (a, y))$:
   (a) Find the preimage x of y in S_Q, and if needed, infer the preimage using the same-instance e and $eval$ queries in S_Q; if not found, answer $\perp$. (See Algorithm 7 for the details.)
   (b) Otherwise, the preimage x of y was learned. Simulate the output of $\mathsf{V}^{(\cdot)}(a, w)$, where every query q' asked by V is recursively answered by this procedure using the current state S_Q. If V outputs 0, answer $\perp$. Otherwise, answer x.

The attacker Brk works w.h.p. because the queries and the labels in the canonicalized execution is a function of the vector of a functional key.

Our novelty lies in making additional queries (Solution 2) and in the recurrent canonicalization (Solution 3). Also, our techniques are compatible with HLOW Lemma. Still, some minor issues could happen due to a collision in the real oracle or in the simulator; such events happen with an exponentially small probability. The details are deferred to Section 4.

2.5 Separate IPFE from Instance-Hiding Witness Encryptions

We illustrate that there is no IPFE in the two-layer instance-hiding witness encryption (IHWE, Definition 2.2) oracle world in this subsection. As mentioned earlier in Section 2.1, two-layer predicate encryption (PE) exists in the IHWE world. Thus, this separates IPFE from two-layer PEs. Recall that in the IHWE oracle, there is no homomorphic evaluation $eval$, but the permuting suboracle $e(a, \star)$ outputs a label lb that *hides the instance* a. Still, the inverting suboracle $d_{\mathsf{V}}(w, \mathsf{lb})$ verifies the witness w against the hidden instance, and the answer depends on the verification. Such verification is also oracle-aided and can make indirect queries on the oracle itself, which introduces completely new challenges.

The instance-hiding labels retard an attacker from performing the verification in several cases. That is, similar to the monolithic IRHWE, the answer to a d-query depends on the outcome of verification, which in turn depends on the indirect queries of the verification. However, differing from IRHWE, due to the instance-hiding property of IHWE, we may *not* know the instance of a label, which makes learning and simulating the indirect queries challenging. Recall that for each instance, there are exponentially many legitimate witnesses, so that learning the directly used witnesses (but not the verification) is insufficient, even when we have already learned the preimage of the instance.

Challenge 4: Retardant label from encryption. We begin with the labels created during the Enc procedure of the challenge ciphertext. Following Solution 2, whenever such a label lb encodes essential information about the plaintext, and if a query $d_{\mathsf{V}}(w, \mathsf{lb})$ is made in the decryption with the challenge functional key, then by HLOW Lemma, lb must also be queried by $d_{\mathsf{V}}(w_1, \mathsf{lb})$ in a decryption with a colluded key for some potentially different witness w_1. However, there is no guarantee that the verification can be learned. An example is shown below.

- $c \leftarrow \mathsf{Enc}(\star)$ queries $\mathsf{lb} \leftarrow e(a, x)$ and gets a label lb.
- For the challenge vector $\mathbf{v}$ and the functional key $\mathsf{sk}_{\mathbf{v}}$, $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}}, c)$ queries lb with some witness w, i.e., $d_{\mathsf{V}}(w, \mathsf{lb})$; moreover, the verification $\mathsf{V}(a, w)$ indirectly queries the suboracle $e(z)$ for some z that depends on (a, w). The attacker needs to answer this query $d_{\mathsf{V}}(w, \mathsf{lb})$ correctly.

We consider two cases: whether lb is inverted successfully by $d_{\mathsf{V}}(w, \mathsf{lb})$ during challenge functional decryption with $\mathbf{v}$. In the successful case, by HLOW Lemma, we must have a colluded key that also successfully inverted the same label lb. Hence, learning the preimage (a, x) and thus the instance a is guaranteed w.h.p. Unfortunately, the learning phase may yield different witnesses and mixed outcomes of inverting or not, as shown below. Hence, the attacker still needs to learn the verification $\mathsf{V}(a, w)$ in order to correctly answer $\perp$ or the preimage.

- (In learning phase.) For a colluded vector $\mathbf{v}_1$ and a colluded functional key $\mathsf{sk}_{\mathbf{v}_1}$, $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}_1}, c)$ queries lb with some witness w_1, which indirectly invoked $e(z_1)$, and the oracle answered $\perp \leftarrow d_{\mathsf{V}}(w_1, \mathsf{lb})$. For another colluded vector $\mathbf{v}_2$ and a colluded functional key $\mathsf{sk}_{\mathbf{v}_2}$, $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}_2}, c)$ queries lb with some witness w_2, which indirectly invoked $e(z_2)$, and the oracle answered the preimage $(a, x) \leftarrow d_{\mathsf{V}}(w_2, \mathsf{lb})$.

In the unsuccessful case, we have no guarantee that the instance has been learned or not. If the same label lb is always unsuccessfully in all d-queries in the learning phase, then by the contrapositive of HLOW Lemma, lb is also unsuccessful in the decryption using the challenge key w.h.p., and thus the attacker can safely answer $\perp$. Otherwise, label lb is sometimes successfully inverted, as in the above learning phase, the attacker learned the instance, but it also needs to learn the verification. We reiterate that in both cases, learning the verification is needed, but the witness w can differ from everything used in the learning phase.

We say such a label lb is *retardant*. Observe that in the above example, the functional key $\mathsf{sk}_{\mathbf{v}_2}$ successfully inverted lb, while another $\mathsf{sk}_{\mathbf{v}_1}$ failed the inversion. Hence, the verification $\mathsf{V}(a, w_2)$ and the indirect queries invoked by this verification can be learned. In contrast, the failed verification $\mathsf{V}(a, w_1)$ may not be learned. Notice that the label lb stopped the attacker to learn some indirect queries, i.e., $e(z_1)$ in the above example. Specifically, when the indirect queries are not learned for some colluded vectors, we cannot claim that the attacker has learned the indirect queries needed by the challenge vector—HLOW Lemma requires *all* colluded vectors. Without the correct indirect queries, the verification and thus the caller d-query cannot be correctly answered.

Solution 4: Learn from cross verification. To address the issue, we utilize a successfully revealed instance *across all colluded functional keys*. In the above example, the existence of a successful query $d_{\mathsf{V}}(w_2, \mathsf{lb})$ in the learning phase is indeed helpful—it reveals the instance a of the label lb! Given the instance a, the attacker can emulate the verification $\mathsf{V}^{(\cdot)}(a, w')$ for every query $d_{\mathsf{V}}(w', \mathsf{lb})$ on the same lb, for all colluded keys, no matter the verification $\mathsf{V}(a, w')$ is successful or not. The idea is elaborated as below.

2. (Learning phase.) For each of the challenge ciphertext c_i, repeat the following until there is no new query-answer (Q-A) pair appended into Q.
   (a) Emulate IPFE decryption $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}_j}, c_i)$ using all colluded keys for all $j \in [t]$; record in Q the Q-A pairs made on the IHWE oracle. Moreover, for each d-query of form $d_{\mathsf{V}}(w, \mathsf{lb})$, if the instance a has been learned in Q, emulate the verification $\mathsf{V}(a, w)$ and record in Q the Q-A pairs made on the IHWE oracle.

We claim that cross verification is sufficient for the retardant label from encryption. By the contrapositive of HLOW Lemma, if all queries $d_{\mathsf{V}}(\star, \mathsf{lb})$ are unsuccessful in the learning phase, then the challenge functional key must be unsuccessfully answering the query $d_{\mathsf{V}}(w, \mathsf{lb})$, which the attacker can simulate without learning the verification. Otherwise, some queries $d_{\mathsf{V}}(\star, \mathsf{lb})$ are successful and reveal the instance a in the learning phase, then the attacker learns all the indirect queries needed for the verification $\mathsf{V}^{(\cdot)}(a, \cdot)$, again by HLOW Lemma. The attacker can simulate the answer in both cases w.h.p.

We note that cross learning differs significantly from simply increasing the number of colluded keys, t. Specifically, an IPFE may always fail to invert a label lb whenever the vector $\mathbf{v}_1$ of the functional key is in a "bad" subset. Without using an instance from another key, only increasing the colluded keys does not resolve the issue.

Challenge 5: Retardant label due to key generation. Recalling that the attacker enumerates exhaustively all functional keys and the queries occur during key generation (Section 2.2), we initially thought that the attacker had full information, and thus simulating queries that depend on the challenge functional key was easy. Surprisingly, the following example raises a challenge.

- $\mathsf{sk}_{\mathbf{v}} \leftarrow \mathsf{KGen}(\mathsf{msk}, \mathbf{v})$ queries $\mathsf{lb} \leftarrow e(a, x)$ and gets a label lb.
- $c \leftarrow \mathsf{Enc}(\star)$ queries $\star \leftarrow e(z)$.
- For the functional key $\mathsf{sk}_{\mathbf{v}}$, $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}}, c)$ queries $d_{\mathsf{V}}(w, \mathsf{lb})$ with some witness w; moreover, the verification $\mathsf{V}(a, w)$ indirectly queries the suboracle $e(z)$. The attacker needs to answer this query $e(z)$ correctly.

It is again because when the attacker needs to answer a d-query in the simulation phase, the answer depends not only on (the instance and preimage of) the label lb but also on the verification, witness, and thus the indirect queries. As in the above example, the attacker already knows the preimage and instance of lb by exhaustive enumeration, but the verification and the answer to $d_{\mathsf{V}}(w, \mathsf{lb})$ can still be $\perp$ or the preimage, which can depend on the answer of $e(z)$. Thus, learning $e(z)$ is needed. However, in the learning phase, the d-query can be inverting *another label* lb_2 *from a colluded key*, and the inversion may be unsuccessful.

- $\mathsf{sk}_{\mathbf{v}_2} \leftarrow \mathsf{KGen}(\mathsf{msk}, \mathbf{v})$ queries $\mathsf{lb}_2 \leftarrow e(a_2, x_2)$.
- For the colluded key $\mathsf{sk}_{\mathbf{v}_2}$, $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}_2}, c)$ queries $d_{\mathsf{V}}(w_2, \mathsf{lb}_2)$ with some witness w_2; moreover, the verification $\mathsf{V}(a_2, w_2)$ indirectly queries the suboracle $e(z)$, and then $\mathsf{V}(a_2, w_2) = 0$. The attacker did not learn the query $e(z)$ because the instance a_2 remains hidden.

Because the instance a only occurs in the challenge functional key, it seemed that the learning phase guaranteed nothing about learning a_2 and thus nothing about $e(z)$. Such a label is called retardant due to key generation.

Solution 5: Compile the key generation with additional ciphertexts. Observe that if the challenge key $\mathsf{sk}_{\mathbf{v}}$ successfully inverts a label from its key generation for a ciphertext c, then there exists another colluded key that successfully inverts a label from its key generation for another ciphertext c'—this is also implied HLOW Lemma. Elaborately, we define the function F to be:

- $F(\mathbf{u})$ is hardwired with the IHWE oracle, the master secret key msk, the challenge ciphertext c, a set of "compilation" ciphertexts C_{com} sampled uniformly at random, and a fixed number k; it performs the following.
  - Let lb be the answer of the kth e-query occurred in the $\mathsf{sk}_{\mathbf{u}} \leftarrow \mathsf{KGen}(\mathsf{msk}, \mathbf{u})$.
  - Let S_{k} be the set of the indirect Q-A pairs invoked by the query $d_{\mathsf{V}}(\star, \mathsf{lb})$ in the challenge ciphertext decryption $\mathsf{Dec}(\mathsf{sk}_{\mathbf{u}}, c)$. Let S_{e} be the set of all Q-A pairs invoked by the encryption of c. Let $S := S_{\text{k}} \cap S_{\text{e}}$.
  - If there exists a ciphertext $c' \in C_{\text{com}}$ such that during the "compilation" decryption $\mathsf{Dec}(\mathsf{sk}_{\mathbf{u}}, c')$, the query $d_{\mathsf{V}}(\star, \mathsf{lb})$ occurs and inverts the preimage successfully, then output S; otherwise, output the bot-set $\{\perp\}$.

The intuition is that if the challenge vector $\mathbf{v}$ is "good with" its kth label, then there exists another colluded vector $\mathbf{v}_2$ that is also good with its corresponding kth label by HLOW Lemma, where "good" is defined to be the existence of $c' \in C_{\text{com}}$ that yields a successful inversion. Equivalently, if $F(\mathbf{v})$ takes the if-branch, then there exists $F(\mathbf{v}_2)$ that also takes the if-branch. The label lb is fixed by the kth query for a fixed k, ensuring both c and c' query the same label.

Notice that the potential output S is defined no matter whether the challenge c inverts $d_{\mathsf{V}}(\star, \mathsf{lb})$ successfully or not. Hence, an additional argument is needed—namely, for vector $\mathbf{v}$, if lb is successfully inverted for the challenge c, then we want $F(\mathbf{v})$ to take the if-branch. The argument is accomplished by the well-known compilation technique, which was employed in previous separations, also called heavy queries [MP12, CKP15]. That is, even when a label lb is created in the colluded functional key generation, which is not directly given to the attacker, if lb is inverted successfully in a random compilation ciphertext, then the preimage of lb is essentially given as part of the colluded key. Equivalently, anyone who gets the functional key can learn the instance of lb using sufficiently many random ciphertexts.

The idea is elaborated in the following attacker (Algorithm 3), where we deferred simulating the answers to Algorithm 4. Notice that in the simulation phase, we also compile the enumerated key $\mathsf{sk}'_{\mathbf{v}}$ by further enumerating the queries, marked by †—that is needed for analysis but also causes an issue, which we will elaborate and resolve soon.

Algorithm 3. IPFE attacker in the IHWE world, informal.

Colluding phase: Similar to the framework (Section 2.2), but additionally obtain sufficiently many random compilation ciphertexts from the challenger. Let C_{com} be the set of compilation ciphertexts. Recall that the challenger provides ciphertext-value pairs (c_i, z_i). Below, C_{com} consists of only the ciphertext c_i and discards z_i, and only those are not in C_{com} remain called "challenge ciphertexts."
Learning phase: Initialize Q to be an empty list. Perform the following.

1. (Compilation.) For each colluded key $\mathsf{sk}_{\mathbf{v}_j}$, for each $c' \in C_{\text{com}}$, emulate $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}_j}, c')$; append all occurred Q-A pairs into Q. Moreover, if a d-query $d_{\mathsf{V}}(w, \mathsf{lb})$ occurred on a

label lb and the instance a of lb is in Q, emulate the verification $\mathsf{V}(a, w)$, and append the occurred Q-A pairs into Q as well.

2. (Cross learning.) For each of the challenge ciphertext c_i, repeat the following until there is no new query-answer (Q-A) pair appended into Q.

   (a) Emulate IPFE decryption $\mathsf{Dec}(\mathsf{sk}_{\mathbf{v}_j}, c_i)$ using all colluded keys for all $j \in [t]$; record in Q the Q-A pairs made on the IHWE oracle. Moreover, for each d-query of form $d_{\mathsf{V}}(w, \mathsf{lb})$, if the instance a has been learned in Q, emulate the verification $\mathsf{V}(a, w)$ and record in Q the Q-A pairs made on the IHWE oracle.

Simulation phase: Exhaustively enumerate each string $\mathsf{sk}'_{\mathbf{v}}$ and two lists Q'_{kgen} and T', where

- Q'_{kgen} is the list of Q-A pairs that occurred when generating $\mathsf{sk}'_{\mathbf{v}}$, and
- T' is the list of Q-A pairs that occurred when *compiling* $\mathsf{sk}'_{\mathbf{v}}$ *using the compilation ciphertexts* $C_{\mathrm{com}}{}^{\dagger}$.

For each enumerated tuple $(\mathsf{sk}'_{\mathbf{v}}, Q'_{\mathsf{kgen}}, T')$, and for each challenge c_i, simulate IPFE decryption, $\mathsf{Dec}(\mathsf{sk}'_{\mathbf{v}}, c_i)$, using the Simulator of IHWE that is initialized with $(Q, Q'_{\mathsf{kgen}}, T')$ (Algorithm 4). If there is a tuple $(\mathsf{sk}'_{\mathbf{v}}, Q'_{\mathsf{kgen}}, T')$ such that functionally decrypts c_i to the challenge product z_i for all challenge ciphertexts, then the attacker outputs 'Yes' in the security game; otherwise, it outputs 'No'.

To see how the analysis works, we go through the cases of the query $d_{\mathsf{V}}(w, \mathsf{lb})$ as provided in the example. Recall that since lb is from $\mathsf{sk}'_{\mathbf{v}}$, which the attacker enumerated, the attacker knows the preimage of lb is (a, x) from Q'_{kgen} but needs to know whether to answer $\perp$ or not.

- If the answer is the preimage, by concentration bound and the same distribution of challenge c_i and compilation c', there exists a $c' \in C_{\mathrm{com}}$ such that also inverts the kth lb, which implies that $F(\mathbf{v}) \neq \{\perp\}$. By HLOW Lemma, $F(\mathbf{v})$ is covered by the union of colluded vectors, i.e., each query in $F(\mathbf{v})$ is in $F(\mathbf{v}_1) \neq \{\perp\}$ for some colluded vector $\mathbf{v}_1$. By construction of F and the learning phase, each $F(\mathbf{v}_1) \neq \{\perp\}$ is learned, which implies that $F(\mathbf{v})$ is learned. The attacker can simulate the verification w.h.p.
- If the answer is $\perp$ and all compilation ciphertexts fail to invert the kth lb, then the attacker simply answers $\perp$ as the concentration of ciphertexts suggests. The answer $\perp$ is correct w.h.p. (HLOW Lemma is unused).
- If the answer is $\perp$ but some compilation ciphertexts succeed to invert the kth lb, then $F(\mathbf{v}) \neq \{\perp\}$. By the same argument as the preimage case, $F(\mathbf{v})$ is covered and learned. The attacker can simulate the verification w.h.p.

Notice that the concentration bound relies on the compilation of the enumerated key, marked † in Algorithm 3. The Simulator of IHWE is given in Algorithm 4, which closely follows the above analysis.

Algorithm 4. Simulator of IHWE oracle, informal.

Initialization: Given $(Q, Q'_{\mathsf{kgen}}, T')$ as input, a state S_Q is initialized to the Q-A list $Q \cup Q'_{\mathsf{kgen}}$, while the lists (Q'_{kgen}, T') are kept unchanged.
Answer a query q: Once initialized, the simulator answers to q and appends the current Q-A pair to S_Q; it is similar to that of IRHWE, where the only differences are (a) there is no $eval$ query, and (b) the procedure to answer d-queries differs and is described below.

1. (The two cases, queries are already in S_Q and e-queries, are identical to Algorithm 2.)
2. If q is a d-query of the form $d_{\mathsf{V}}(w, \mathsf{lb})$:[a]
   (a) If lb is the output of an e-query in Q'_{kgen} and q is a direct query from IPFE Dec,
      i. If lb is never inverted in the compilation T', answer $\perp$.
      ii. If lb is inverted successfully in the compilation T', use the preimage (a, x) of lb in Q'_{kgen}, simulate $\mathsf{V}^{(\cdot)}(a, w)$, where every query asked by V is recursively answered by this procedure using the current state S_Q. If V outputs 0, answer $\perp$. Otherwise, answer (a, x).
   (b) Else if the preimage (a, x) of lb is found in S_Q, simulate $\mathsf{V}^{(\cdot)}(a, w)$ recursively, and answer the preimage or $\perp$ accordingly.
   (c) Finally, the preimage is not found S_Q, answer $\perp$.

[a]We abuse notation and implicitly simulate d^0_{V}-queries similar to d_{V}; the only difference is Item 2a, where "direct from IPFE" distinguishes d_{V} from d^0_{V}.

Challenge 6: Too many enumerations due to the compilation. The compilation solution to Challenge 5 works when the challenger indeed sends the inner products of the challenge key and challenge ciphertexts. However, when the challenger sends freshly independent and random values, the attacker shall not false-positively hit the random values in the simulation phase. Recall that in the framework (Section 2.2), such false positives are suppressed by increasing the number of challenge ciphertexts, t_c. Roughly speaking, if the attacker enumerates all key-query tuples, where each tuple is l-bit, then we need $t_c \gg l$, so that all enumerations do not hit t_c random values except with a negligible probability.

It is too much to enumerate all the potential queries in the compilation, as per the †-marked step in Algorithm 3. To see why, let $\tilde{t_c} := |C_{\text{com}}|$ be the number of compilation ciphertexts. Recall that t is the number of colluded keys. Our probabilistic analysis requires the following between parameters $t, t_c, \tilde{t_c}$, but it is impossible to satisfy all of them.

- $t_c \gg \tilde{t_c}$: We want to suppress false positives, but each enumerated string is longer than $\tilde{t_c}$.
- $\tilde{t_c} \gg t$: We want to ensure the concentration bound holds for each colluded key, and we take a union bound over all t keys.
- $t \gg t_c$: We want to ensure HLOW Lemma holds for all ciphertexts, and we take a union bound over all t_c ciphertexts.

Solution 6: Enumerate precisely for the simulation. To address the issue, our observation is that there is no need to enumerate the compilation for the enumerated functional key—the simulator only needs the *implication* of the compiled Q-A list, much shorter than the list. Namely, in Algorithm 3, the attacker enumerates the list T', but the list T' is only used by the simulator to answer $\perp$ or not in Algorithm 4—such $\perp$-or-not is just 1-bit information for each d-query in Dec! Particularly, it is independent of the number of compilation, $\tilde{t_c}$.

2.5

Informally, we replace T' with an indicator string I' of length polynomial in ℓ, where $\ell = \mathrm{poly}(\lambda)$ is the number of direct and indirect queries in the IPFE scheme. For each d-query q that the simulator may need to decide $\perp$ or not, $I'[q]$ is the indicator that simply gives the decision to the simulator. That is, in Algorithm 4, between the two Cases 2(a)i and 2(a)ii, if $I'[q] = 0$, the simulator goes into Case 2(a)i, otherwise it goes into Case 2(a)ii. Because there are at most $\mathrm{poly}(\ell)$ labels in Q'_{kgen} and the simulator only needs a decision for the Q'_{kgen} cases, enumerating $\mathrm{poly}(\ell)$ bits is sufficient. The above $t_c \gg \tilde{t_c}$ is now replaced by $t_c \gg \mathrm{poly}(\ell)$, yielding satisfiable parameters.

Details, summary and novelty. We omitted many details and case analysis in the separation of IHWE. Particularly, the indexing of indicators in the algorithm and in the analysis is skipped, which should be precise and formal—the algorithm looks for the index of the label in the Q-A list of key generation, Q'_{kgen}, and then uses the indexed indicator. The details are deferred to Section 5.

Overall, our attacker cross-learns the hidden instances for labels created when the challenge ciphertext is encrypted. Also, we utilize compilation ciphertexts to collect queries and thus the associated hidden instances for the colluded keys; we also make concise guesses that are sufficient for the attacker to simulate the functional decryption correctly. Cross learning is novel in the literature of separations to the best of our knowledge. Also, this is the first work to compose the technique of Hajiabadi et al. and the older compilation techniques.

2.6 Technical Open Questions

Our attacker breaks IPFE in the two-layer IHWE oracle world. An open question is whether IPFE exists relative to IHWE oracle with more layers. We did not find an IPFE scheme that fools our attacker, but our probabilistic analysis does not work for some examples. We illustrate the challenge in this subsection.

We begin with viewing an IPFE algorithm as an oracle-query tree, where the tree topology is fixed for all same-length input. To see why, represent an IPFE algorithm, such as Dec, as an oracle-aided circuit for a fixed input size. Clearly, the Dec circuit consists of pre-determined oracle-query gates with a fixed topology. Moreover, each oracle-query gate may invoke oracle queries in the monolithic model, and all invocations are pre-determined by the oracle-aided universal circuits, such as V. Hence, any Dec circuit defines a tree, the root is Dec circuit, and each child node is a query invoked by its parent node. We remark that each node is also associated with a pre-determined type of suboracle.

When considering IPFE in the monolithic IHWE oracle world, the number of layers and thus the tree depth is arbitrary. Recall that ℓ is the total number of queries of IPFE. The tree depth can be $O(\ell)$ at most. Also, notice that when the same label is inverted (by d-queries) at different nodes of the oracle-query tree, the answer can be totally different in terms of $\perp$ or preimage.

When we apply HLOW Lemma to bound the probability that our attacker fails to learn some queries, the probability grows exponentially with the number of layers. To see why, consider an IPFE such that the oracle-query tree of Dec consists of a single-line path of k queries, i.e., Dec invokes $q_1 := d_{\mathsf{V}}(\star, \mathsf{lb}_1)$, which invokes $q_2 := d_{\mathsf{V}}(\star, \mathsf{lb}_2)$, and so on, and then $q_{k-1} := d_{\mathsf{V}}(\star, \mathsf{lb}_{k-1})$ invokes $q_k := e(a, x)$. All labels lb_i are obtained during IPFE Enc, and the e-query $e(a, x)$ occurred in Enc as well. We do not know how to employ HLOW Lemma when k is large, e.g., $k = \omega(\log \lambda)$. It is because in Solution 4, when we argue that the children queries invoked by a parent d-query can be learned, we must condition on the *same label* of the parent; the conditioning makes the HLOW function F consistent with the children learned in the learning phase. Conditioning on $k = \omega(\log \lambda)$ ancestors is too many, incurring a useless union bound.

Notice that we left the IPFE to choose the labels on the path arbitrarily. The above example can be easily randomized in many ways, e.g., mixing labels from KGen and Enc, where the randomness is obtained from an e-query of IHWE and is seeded by both the whole input of Dec. One can narrow down the choice of labels to a uniformly random subset of labels from Enc. However, in that case, our attacker can actually learn all the instances with a good probability. This is because each uniformly chosen label and its preimage can be learned with a polynomial number of colluded keys, and then cross learning ultimately solves all the preimages. The issue is to rule out all cases, which remains unclear.

Other open questions. Our attacker takes exponential time in the simulation phase due to the enumeration of all possible functional decryption keys, regardless of whether the oracle is IRHWE or IHWE. That is inherited from the framework of Hajiabadi et al. [HLOW24]. If we could achieve an efficient simulation phase, potentially at the cost of more oracle queries in the learning phase, then it seemed that our attacker would be a “compiler” that transforms an IPFE in IRHWE-oracle model into another IPFE in the RO model. The other direction, from compilation to efficient attacker, might also hold. Both directions are desirable properties, e.g., some compilation gives black-box uselessness [CFM21]. We leave them as open questions.

3 Preliminaries

Define $\mathbb{N} = \{0, 1, 2, \ldots\}$ to be the set of natural numbers and $\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$ to be the set of integers. For any integer $n \geq 1$, define $[n] = \{1, \ldots, n\}$. By default, all our logarithms are base 2 and $\log n$ stands for $\log_2 n$. For any $p \in \mathbb{N}$, let $\mathbb{Z}_p$ be the ring $\mathbb{Z}/p\mathbb{Z}$. A function $\nu : \mathbb{N} \to \mathbb{N}$ is said to be negligible, denoted $\nu(n) = \mathrm{negl}(n)$, if for every positive polynomial $p(\cdot)$ and all sufficiently large n it holds that $\nu(n) < 1/p(n)$. We use the abbreviation PPT for probabilistic polynomial time. For a finite set S, we write $a \leftarrow S$ to mean a is sampled uniformly randomly from S. For a randomized algorithm A, we let $a \leftarrow A(\cdot)$ denote the process of running $A(\cdot)$ and assigning the outcome to a; when A is deterministic, we write $a := A(\cdot)$ instead. We denote the security parameter by λ. For two distributions X, Y parameterized by λ we say that they are computationally indistinguishable, denoted by $X \approx_c Y$ if for every non-uniform PPT distinguisher D, we have $|\Pr[D(X) = 1] - \Pr[D(Y) = 1]| = \mathrm{negl}(\lambda)$.

3.1 Black-Box Constructions

The black-box constructions of cryptographic primitives is defined in the standard way.

Definition 3.1 (Cryptographic primitives [RTV04])**.** *A* primitive *$\mathcal{P} = (\mathcal{F}, \mathcal{R})$ is defined as set of functions $\mathcal{F}$ and a relation $\mathcal{R}$ between functions. A (possibly inefficient) function $F \in \{0,1\}^* \to \{0,1\}^*$ is a correct implementation of $\mathcal{P}$ if $F \in \mathcal{F}$, and a (possibly inefficient) adversary A breaks an implementation $F \in \mathcal{F}$ if $(A, F) \in \mathcal{R}$.*

Definition 3.2 (Black-box constructions [RTV04])**.** *A* black-box construction *of a primitive $\mathcal{Q}$ from a primitive $\mathcal{P}$ consists of two PPT algorithms (Q, S):*

1. *Implementation: For any oracle P that implements $\mathcal{P}$, Q^P implements $\mathcal{Q}$.*
2. *Security reduction: for any oracle P implementing $\mathcal{P}$ and for any (computationally unbounded) oracle adversary A breaking the security of Q^P , it holds that $S^{P,A}$ breaks the security of P.*

3.2 Monolithic Constructions and Separations

In many modern cryptographic primitives, the input are circuits, the primitive executes the circuit, but the execution may call the primitives themselves. To model such primitives, the circuits are often restricted to oracle-aided circuits, which can only call the primitive recursively in an oracle way. We borrow the definitions from Garg, Mahmoody, and Mohammed (GMM) [GMM17]. We begin with universal circuit evaluators.

Definition 3.3 (Universal circuit evaluator [GMM17, Definition 4.2])**.** *By oracle-aided algorithm $\mathsf{Univ}^{(\cdot)}$ we denote the* oracle-aided universal circuit evaluator *which accepts a pair of inputs (C, x) where C is an oracle-aided circuit and x is a string in the domain of* C*, and $\mathsf{Univ}^{(\cdot)}(C, x)$ outputs $C^{(\cdot)}(x)$ by forwarding all of C's oracle queries to its own oracle. When it is clear from the context, we sometimes abuse notation and omit the superscript of Univ (particularly in nested subscript and/or superscript).*

To model the recursive nature of such primitives, GMM proposed the framework of monolithic constructions. They began with defining a family of primitives that is indexed by functions. Looking forward, each index will describe a way to evaluate circuits. With the indexing through functions, they defined a recursive indexing to a primitive in the family.

Definition 3.4 (Family of indexed primitives [GMM17, Definition 4.1])**.** *Let $\mathcal{W}$ be a set of (possibly inefficient) functions. A $\mathcal{W}$-*indexed family of primitives *$\mathcal{P}[\mathcal{W}]$ is a set of primitives $\{\mathcal{P}[w]\}_{w \in \mathcal{W}}$ each indexed by some $w \in \mathcal{W}$ where, for each $w \in \mathcal{W}$, $\mathcal{P}[w] = (\mathcal{F}[w], \mathcal{R}[w])$ is a primitive according to Definition 3.1. We call the primitive $\mathcal{P}[w]$ the w-*indexed member *of the family $\mathcal{P}[\mathcal{W}]$.*

Definition 3.5 ($V^{(\cdot)}$-indexed primitives [GMM17, Definition 4.3])**.** *For a $\mathcal{W}$-indexed family $\mathcal{P}[\mathcal{W}] = \{(\mathcal{F}[w], \mathcal{R}[w])\}_{w \in \mathcal{W}}$, and an oracle algorithm $V^{(\cdot)}$, define the primitive $\mathcal{P}[V^{(\cdot)}] = (\mathcal{F}', \mathcal{R}')$ as follows. For all $w \in \mathcal{W}$ and $F \in \mathcal{F}[w]$, if $w = V^F$, then we include $F \in \mathcal{F}'$, and for any such (w, F), if $(F, A) \in \mathcal{R}[w]$, then we include (F, A) in $\mathcal{R}'$ as well. We call $\mathcal{P}[V^{(\cdot)}]$ the $V^{(\cdot)}$-*indexed primitive *defined by (or, from) $\mathcal{P}[\mathcal{W}]$.*

In the above definition, given any $V^{(\cdot)}$, we find $w \in \mathcal{W}$ such that satisfies $w = V^F$ for some $F \in \mathcal{F}[w]$. That models the recursive nature, where V may call the primitive itself, modeled by oracle calls to F. Notice that the notation $\mathcal{P}[V^{(\cdot)}]$ is abused as it might not be identical to $\mathcal{P}[w]$ for any $w \in \mathcal{W}$. Next, GMM defined the "monolithic" extension of a primitive by allowing recursive calls in $V^{(\cdot)}$.

Definition 3.6 (Monolithic extension of indexed primitives [GMM17, Definition 4.4])**.** *For any indexed family $\mathcal{P}[\mathcal{W}]$ and $V \in \mathcal{W}$, we call the $V^{(\cdot)}$-indexed primitive $\mathcal{P}[V^{(\cdot)}]$ the* monolithic extension *(or simply the extension) of the V-indexed primitive $\mathcal{P}[V]$ (both with respect to the family $\mathcal{P}[\mathcal{W}]$). Whenever $\mathcal{P}[\mathcal{W}]$ is clear form the context, we simply call $\mathcal{P}[V^{(\cdot)}]$ the monolithic extension of $\mathcal{P}[V]$.*

Throughout this work, we consider only $V = \mathsf{Univ}$, that is, the monolithic extension $\mathcal{P}[\mathsf{Univ}^{(\cdot)}]$ of $\mathcal{P}[\mathsf{Univ}]$. Next, monolithic constructions are defined by black-box constructions (Definition 3.2) from monolithic extensions.

Definition 3.7 (Monolithic constructions [GMM17, Definition 4.8])**.** *Suppose $\mathcal{Q}$ is a primitive and $\widetilde{\mathcal{P}}$ is the monolithic extension of the primitive $\mathcal{P}$. Any fully black-box construction for $\mathcal{Q}$ from $\widetilde{\mathcal{P}}$ (i.e. the monolithic extension of P) is called a* monolithic construction *of $\mathcal{Q}$ from $\mathcal{P}$.*

With the definition of monolithic constructions, GMM derived the following lemma. We state the contrapositive, which will help proving separation between primitives

Lemma 3.8 (Separation of monolithic constructions [GMM17, Definition 4.10])**.** *Suppose $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$ are cryptographic primitives. If there is a monolithic construction of the monolithic extension $\widetilde{\mathcal{Q}}$ from $\mathcal{P}$ and if there is a monolithic construction of $\mathcal{R}$ from $\mathcal{Q}$, then there is a monolithic construction of $\mathcal{R}$ from $\mathcal{P}$.*

By contrapositive, if one proves: (a) There is no *monolithic construction of $\mathcal{R}$ from $\mathcal{P}$ and (b) there* is *a monolithic construction of the monolithic extension $\widetilde{\mathcal{Q}}$ from $\mathcal{P}$, then there is* no *monolithic construction of $\mathcal{R}$ from $\mathcal{Q}$.*[4]

3.3 Primitives

We define selective-secure secret-key functional encryption for inner-product functions below.

Definition 3.9 (Secret-Key Inner-Product Functional Encryption)**.** *Let λ be the security parameter and $n := n(\lambda), q := q(\lambda)$, a secret-key inner-product functional encryption (SK-IPFE) for vectors in $\mathbb{Z}_p^n$ consists of the following algorithms.*

- $\mathsf{sk}[\mathbf{v}] \leftarrow \mathsf{KGen}(\mathsf{msk}, \mathbf{v})$ *takes as input a master secret key msk and a vector $\mathbf{v} \in \mathbb{Z}_q^n$, outputs a secret key* $\mathsf{sk}[\mathbf{v}]$.[5] *Without loss of generality, we assume that* msk *is uniformly chosen from $\{0,1\}^\omega$ for some polynomial $\omega := \omega(\lambda)$.*
- $c \leftarrow \mathsf{Enc}(\mathsf{msk}, \mathbf{m})$ *takes as input a master secret key* msk *and a message vector $\mathbf{m} \in \mathbb{Z}_q^n$, outputs the ciphertext c.*
- $z \leftarrow \mathsf{Dec}(\mathsf{sk}[\mathbf{v}], c)$ *takes as input a secret key* $\mathsf{sk}[\mathbf{v}]$ *and a ciphertext c, outputs $z \in \mathbb{Z}_q$.*

The algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *should satisfy the following correctness and security requirements.*
Correctness. *A SK-IPFE is $(1-\epsilon)$-correct for dimension n if for any master secret key* $\mathsf{msk} \in \{0,1\}^\omega$, *any key vector $\mathbf{v} \in \mathbb{Z}_p^n$, and any message $\mathbf{m} \in \mathbb{Z}_p^n$,*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}[\mathbf{v}], c) = \langle \mathbf{v}, \mathbf{m} \rangle : \begin{array}{r} \mathsf{sk}[\mathbf{v}] \leftarrow \mathsf{KGen}(\mathsf{msk}, \mathbf{v}); \\ c \leftarrow \mathsf{Enc}(\mathsf{msk}, \mathbf{m}) \end{array}\right] \geq 1 - \epsilon.$$

[4] GMM stated another variant, but the contrapositive is sufficient for this work.
[5] We abuse notation of square brackets only for functional secret keys.

Security. *We focus on the selective-secure IPFE and follow the definition in [HLOW24], which is strictly weaker than the standard one [ABDP15]. The game between a challenger and an adversary $\mathcal{A}$ is presented below.*

1. *Selective queries. $\mathcal{A}$ selects and sends the following to the challenger simultaneously.*
   - *The challenge key vector $\mathbf{v} \in \mathbb{Z}_p^n$.*
   - *Key queries. $\mathbf{v}_1, \ldots, \mathbf{v}_t \in \mathbb{Z}_p^n$ for some polynomial $t := t(\lambda)$.*
   - *The number of challenge ciphertexts $t_c \in \mathbb{N}$, where $t_c := t_c(\lambda)$ is a polynomial.*
2. *The challenger samples a master secret key msk and a bit $b \in \{0, 1\}$. Then, challenger responds to $\mathcal{A}$ with the following.*
   - *Functional keys. If $\mathbf{v} \notin \mathsf{Span}(\mathbf{v}_1, \ldots, \mathbf{v}_t)$, the challenger responses with $\mathsf{sk}[\mathbf{v}_i] \leftarrow \mathsf{KGen}(\mathsf{msk}, \mathbf{v}_i)$ for all $i \in [t]$. Otherwise, the challenger aborts.*
   - *Challenge ciphertexts. For each $j \in [t_c]$, the challenger uniformly chooses $\mathbf{m}_j \in \mathbb{Z}_p^n$, computes $c_j \leftarrow \mathsf{Enc}(\mathsf{msk}, \mathbf{m}_j)$, and sends (c_j, m'_j) to $\mathcal{A}$, where $m'_j = \langle \mathbf{v}, \mathbf{m}_i \rangle$ if $b = 0$ and $m'_j \leftarrow \mathbb{Z}_q$ if $b = 1$.*
3. *Guess. $\mathcal{A}$ outputs a bit b'.*

A SK-IPFE is selective-secure for dimension n if for every non-uniform PPT adversary $\mathcal{A}$ and for every polynomial t, there exists a negligible function negl such that the advantage of $\mathcal{A}$ in the above security game is $|\Pr[b' = b] - 1/2| \leq \mathrm{negl}(\lambda)$.

We also define the SK-IPFE relative to an oracle as follows.

Definition 3.10 (Secret-Key Inner-Product Functional Encryption Relative to Oracle)**.** *A SK-IPFE relative to an oracle $\mathcal{O}$ is a tuple of oracle-aided algorithms $(\mathsf{KGen}^{\mathcal{O}}, \mathsf{Enc}^{\mathcal{O}}, \mathsf{Dec}^{\mathcal{O}})$ with syntax and correctness as in Definition 3.9. Such a SK-IPFE is secure iff for any unbounded $\mathcal{O}$-oracle aided adversary $\mathcal{A}$ that makes polynomially-many queries to $\mathcal{O}$, $\mathcal{A}$ has negligible advantage in the security game in Definition 3.9.*

Next, we define three variants of witness encryption. The first is the standard notion of witness encryption.

Definition 3.11 (Witness Encryption (WE) Indexed by Verifier V [GMM17, Definition 3.3])**.** *Let L be an NP language with corresponding efficient relation verifier V, which takes as input a witness w as well as an instance a and either outputs 1 for acceptance or 0 for rejection, a witness encryption (WE) for relation defined by V consists of the following two algorithms.*

- *$c \leftarrow \mathsf{Enc}(1^\lambda, a, m)$ takes as input a security parameter 1^λ, an instance $a \in \{0, 1\}^*$, and a message $m \in \mathcal{M}$, outputs a ciphertext c.*
- *$m/\perp \leftarrow \mathsf{Dec}_{\mathsf{V}}(w, c)$ takes as input a witness w and a ciphertext c, and outputs the underlying message m or $\perp$.*

The algorithms $(\mathsf{Enc}, \mathsf{Dec})$ should satisfy the following correctness and security requirements.

Correctness. *A WE is $(1 - \epsilon)$-correct if for any $m \in \mathcal{M}$, any $a \in L$,*

$$\Pr\left[\mathsf{Dec}_{\mathsf{V}}(w, c) = m : \begin{array}{r} c \leftarrow \mathsf{Enc}(1^\lambda, a, m); \\ \mathsf{V}(a, w) = 1 \end{array}\right] \geq 1 - \epsilon.$$

Security. *The standard security requirement of WE is soundness security, which requires a ciphertext to hide the underlying message. Formally, for any $a \notin L$, any messages $m_0, m_1 \in \mathcal{M}$ such that $|m_0| = |m_1|$, and any PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that*

$$|\Pr[\mathcal{A}(\mathsf{Enc}(1^\lambda, a, m_0)) = 1] - \Pr[\mathcal{A}(\mathsf{Enc}(1^\lambda, a, m_1)) = 1]| \leq \mathrm{negl}(\lambda).$$

The second WE variant augmented with an Eval algorithm that performs homomorphic evaluation on ciphertexts, but it reveals the instances of all ciphertexts. One may view it as FHE, where the public encryption and evaluation key is the instance, and the secret decryption key is the witness. Alternatively, one may view it as ABE, where the the public attribute is the instance, and the decryption keys linked to an access policy is the witness. Also, notice that it works with *oracle-aided circuits* that may call the scheme itself in the oracle way.

Definition 3.12 (Extended Extractable Instance-Revealing Homomorphic Witness Encryption [GMM17, Definition 7.2])**.** *Let $\mathsf{V}^{(\cdot)}$ and $f^{(\cdot)}$ be oracle-aided universal circuit-evaluators, where $\mathsf{V}^{(\cdot)}$ takes as input an instance a as well as a witness w and outputs 1 for acceptance or 0 for rejection, and $f^{(\cdot)}$ takes as input a sequence of messages $m_1, ..., m_l$ and outputs some value $\hat{m}$. An extended extractable instance-revealing homomorphic witness encryption (ex-EIRHWE) scheme defined for $(\mathsf{V}^{(\cdot)}, f^{(\cdot)})$ consists of three PPT algorithms $P = (\mathsf{Enc}, \mathsf{Eval}_f, \mathsf{Dec}_\mathsf{V})$.*

- $\mathsf{lb} \leftarrow \mathsf{Enc}(a, m, 1^\lambda)$ *takes as input an instance $a \in \{0,1\}^*$, a message $m \in \{0,1\}^*$, and a security parameter 1^λ as input. It outputs a label $\mathsf{lb} \in \{0,1\}^*$.*
- $\hat{\mathsf{lb}} \leftarrow \mathsf{Eval}_f(\mathsf{lb}_1, ..., \mathsf{lb}_l)$ *takes as a tuple of labels $\mathsf{lb}_1, ..., \mathsf{lb}_l$ and outputs $\hat{\mathsf{lb}}$.*
- $m' \leftarrow \mathsf{Dec}_\mathsf{V}(w, \mathsf{lb})$ *takes as input a witness $w \in \{0,1\}^*$ as well as a label lb and outputs a message $m' \in \{0,1\}^*$.*

An ex-EIRHWE scheme satisfies the following completeness and security properties.

Decryption correctness. *For any security parameter λ, the following two conditions are satisfied:*

- *For any (w, a) such that $\mathsf{V}^P(w, a) = 1$, and any m*
$$\Pr_{\mathsf{Enc},\mathsf{Dec}}[\mathsf{Dec}(w, \mathsf{Enc}(a, m, 1^\lambda)) = m] = 1.$$
- *For any (w, a) such that $\mathsf{V}^P(w, a) = 1$, and any $m_1, ..., m_l$,*
$$\Pr_{\mathsf{Enc},\mathsf{Eval},\mathsf{Dec}}\left[\mathsf{Dec}(w, \mathsf{Eval}_f(\mathsf{lb}_1, ..., \mathsf{lb}_l)) = f^P(m_1, ..., m_l) \quad : \quad m_i \leftarrow \mathsf{Dec}(w, \mathsf{lb}_i)\ \right] = 1,$$
where $\mathsf{lb}_i = \mathsf{Enc}(a, m_i, 1^\lambda)$ for all $i \in [l]$.

Instance-revealing correctness. *For any security parameter $\lambda \in \mathbb{N}$ and any (a, m), there exists a PPT algorithm* Rev *such that*

$$\Pr[\mathsf{Rev}(\mathsf{Enc}(a, m, 1^\lambda)) = a] = 1.$$

Extractability. *For any PPT adversary $\mathcal{A}$ and polynomial $p_1(\cdot)$, there exists a PPT straight-line extractor* Ext[6] *and a polynomial $p_2(\cdot)$ such that the following holds. For any security parameter $\lambda \in \mathbb{N}$, any instance a, and any messages m_0, m_1 subject to $|m_0| = |m_1| \wedge m_0 \neq m_1$, if*

$$\Pr\left[\mathcal{A}(1^\lambda, c) = b \quad : \quad \begin{array}{l} b \leftarrow \{0,1\}; \\ c \leftarrow \mathsf{Enc}(1^\lambda, a, m_b, 1^\lambda) \end{array}\right] \geq \frac{1}{2} + \frac{1}{p_1(\lambda)},$$

[6]Straight-line extractor is an extractor without rewinding the adversary.

then,

$$\Pr[\mathsf{Ext}^{\mathcal{A}}(a) = w \wedge \mathsf{V}^{P}(a, w) = 1] \geq \frac{1}{p_2(\lambda)}.$$

The third WE variant is required to hide the instance of each ciphertext, but it does not provide homomorphic evaluation. One may view it as a predicate encryption, where the hidden attribute is the instance, and the decryption key, associated with a predicate, is the witness. Similar to the second variant, it works with *oracle-aided circuits* that may call the scheme itself in the oracle way. We follow the definition of GMM but lightly restrict it to call the scheme itself at most *two layers* in an oracle way.

Definition 3.13 (Extended or two-layer Extractable Instance-Hiding Witness Encryption [GMM17, Definition 6.2])**.** *The definition of extended extractable instance-*hiding *witness encryption (ex-EIHWE) is similar to witness encryption (Definition 3.11) except for the following.*

- *The decryption* $\mathsf{Dec}_{\mathsf{V}^P}$ *performs the* oracle-aided verifier V^P*, where* P *is the EIHWE scheme.*
- *Soundness security is replaced by instance-hiding extractability, given below.*

Instance-hiding extractability. *For any PPT adversary* $\mathcal{A}$ *and polynomial* $p_1(\cdot)$*, there exists a PPT (black-box) straight-line extractor* Ext *and a polynomial* $p_2(\cdot)$ *such that the following holds. For any security parameter* $\lambda \in \mathbb{N}$*, any instances* a_0, a_1 *such that* $|a_0| = |a_1|$*, any messages* $m_0, m_1 \in \mathcal{M}$ *such that* $|m_0| = |m_1| \wedge m_0 \neq m_1$*, let* P *denote the ex-EIHWE scheme, if*

$$\Pr\left[\mathcal{A}(1^\lambda, c) = b \quad : \quad \begin{array}{r} b \leftarrow \{0,1\}; \\ c \leftarrow \mathsf{Enc}(a_b, m_b, 1^\lambda) \end{array}\right] \geq \frac{1}{2} + \frac{1}{p_1(\lambda)},$$

then,

$$\Pr[\mathsf{Ext}^{\mathcal{A}}(a_1, a_1) = w \wedge \exists b \in \{0,1\} \text{ s.t. } \mathsf{V}^{P}(a_b, w) = 1] \geq \frac{1}{p_2(\lambda)}.$$

We restrict the extended EIHWE to two-layer EIHWE *by defining the decryption to be* $\mathsf{Dec}_{\mathsf{V}^{P'}}$*, which performs a (more restricted) oracle-aided verifier* $\mathsf{V}^{P'}$*, where* $P' := (\mathsf{Enc}, \mathsf{Dec}_{\mathsf{V}'})$ *is the* plain *EIHWE scheme, and* P' *is instantiated with the* plain *circuit verifier* V' *(without oracle aid). The extractability is modified to* $\mathsf{V}^{P'}$ *correspondingly.*

3.4 Instance-Revealing Homomorphic Witness Encryption Oracle

Our IRHWE oracle is a specialization of the IRHWE of Garg, Mahmoody and Mohammed [GMM17, Definition 7.4]. A main difference is that their oracle supports homomorphic evaluation on ciphertexts that are encrypted using different instances, while our homomorphic evaluation oracle is restricted to the ciphertext of the same instance.

We adapt a minimal amount of "syntax sugar" to the IRHWE oracle of GMM as follows.

- In GMM, the encryption suboracle e is implemented using a single random injective function with long output for different instances, i.e., $e(\cdot)$ itself is an injective function. We append a specific string to the input plaintext and use a distinct random permutation for encryptions under each unique instance. With this adaptation, the injective function's output without a premiage (i.e., the fake labels in GMM) corresponds to the permutation's output with an illegitimate preimage.

- Moreover, GMM incorporates a dedicated suboracle Rev for instance-revealing functionality. It takes a ciphertext as input and outputs the corresponding instance. Our oracle does not rely on it and instead explicitly includes the instance in each ciphertext for simplicity.

These minor modifications do not hurt the functionality or security that the oracle can provide, and these two oracles are equivalent in the single-instance case. Finally, whereas the IRHWE oracle in GMM utilizes a public random oracle, we incorporate it directly as a suboracle that only the IRHWE oracle can ask which also don't hurt the functionality (we just encapsulated the steps for calculating the hash value within the oracle).

The IRHWE oracle is formalized in Definition 3.14, which is similar to the informal Definition 2.1 in the overview. The main difference is the additional zero-string 0^λ in the preimage, and this is needed to formally ensure that only the legitimate labels obtained from e and $eval$ queries can be successfully inverted by d_{V}-queries. Hence, the length of the permutation is extended from 2λ to 3λ. Such labels are called *valid* by Definition 3.15.

Definition 3.14 (Random Instance-Revealing Homomorphic Witness Encryption Oracle)**.** *Let $\mathsf{V}^{(\cdot)}$ be an oracle-aided universal circuit-evaluator that takes an instance a as well as a witness w and outputs 1 for acceptance or 0 for rejection. Let $f^{(\cdot)}$ be an oracle-aided universal circuit-evaluator that takes as input a sequence of messages $m_1, ..., m_l$ to output some value $\hat{m}$. Then, the random instance-revealing homomorphic witness encryption (IRHWE) oracle $\mathcal{O}$ is defined as follows. Below we detail $\mathcal{O}_\lambda$, and $\mathcal{O} = \{\mathcal{O}_\lambda\}_{\lambda\in\mathbb{N}}$.*

- $\mathsf{lb} \leftarrow e(a, m)$ *takes as input an instance $a \in \{0,1\}^{\kappa_1}$ and a message $m \in \{0,1\}^\lambda$, where $\kappa_1 = \mathrm{poly}(\lambda)$. For each a, it uses a random permutation $\pi_a : \{0,1\}^{3\lambda} \to \{0,1\}^{3\lambda}$ to compute $y \leftarrow \pi_a\big(m\|H(a\|m)\|0^\lambda\big)$ and outputs $\mathsf{lb} = (a, y)$, where $H : \{0,1\}^* \to \{0,1\}^\lambda$ is a random oracle.*

- $\hat{\mathsf{lb}}/\bot \leftarrow eval_f(\mathsf{lb}_1, ..., \mathsf{lb}_l)$ *takes a tuple of labels $\mathsf{lb}_1, ..., \mathsf{lb}_l$ and works as follows.*

  1. *Parse $\mathsf{lb}_i = (a_i, y_i)$ for each $i \in [l]$. If $\exists i, j \in [l]$ such that $a_i \neq a_j$, output $\bot$ and abort. Otherwise, continue to the next step. Henceforth, we use a to denote the instance in all labels.*
  2. *For every $i \in [l]$, find (m_i, h_i, s_i) such that $y_i = \pi_a(m_i\|h_i\|s_i)$, where $m_i, h_i, s_i \in \{0,1\}^\lambda$.*
  3. *If $\exists i \in [l]$ such that $s_i \neq 0^\lambda$, uniformly sample $\hat{m} \in \{0,1\}^\lambda$. Otherwise, compute $\hat{m} = f^{\mathcal{O}}(m_1, ..., m_l)$.*
  4. *Compute $\hat{h} = H(f\|a\|y_1\|\cdots\|y_l)$, $\hat{s} = H(s_1\|...\|s_l) \oplus H(0^{l\lambda})$. Output $\hat{\mathsf{lb}} = (a, \hat{y})$, where $\hat{y} = \pi_a(\hat{m}\|\hat{h}\|\hat{s})$.*

- $m/\bot \leftarrow d_{\mathsf{V}}(w, \mathsf{lb})$ *takes as input a witness $w \in \{0,1\}^{\kappa_2}$ and a label lb, where $\kappa_2 = \mathrm{poly}(\lambda)$, outputs m. It works as follows.*

  1. *Parse $\mathsf{lb} = (a, y)$, if $\mathsf{V}^{\mathcal{O}}(a, w) = 0$, output $\bot$.*
  2. *Find (m, h, s) such that $y = \pi_a(m\|h\|s)$ where $m, h, s \in \{0,1\}^\lambda$. If $s \neq 0^\lambda$, output $\bot$. Otherwise, output m.*

Definition 3.15 (Valid labels)**.** *In our IRHWE oracle, a label is generated by applying a permutation to the underlying plaintext (along with a hash value) concatenated with 0^λ. We say a label is* valid *w.r.t. an oracle $\mathcal{O}$ if and only if the last λ bits of its preimage are 0^λ. Notice that when the evaluation oracle $eval$ is given an invalid label, it proceeds with outputting another invalid label.*

Definition 3.16 (Direct and indirect queries)**.** *For any oracle-aided algorithm $A^{\mathcal{O}}$, we say a query is* direct *w.r.t.* $(A, \mathcal{O})$ *if the query is performed by A directly on $\mathcal{O}$. We say a query is* indirect *if the query is performed by $\mathcal{O}$ to itself; that includes the queries performed by computing $f^{\mathcal{O}}$ in an eval_f-query and computing $\mathsf{V}^{\mathcal{O}}$ in a d_{V}-query.*

In the IRHWE oracle world, several cryptographic primitives, including ABE, FHE, and spooky encryptions, exist unconditionally against polynomial-query but unbounded-time adversaries. The primitives are captured by extended extractable IRHWE, and the existence are stated below. We skip the proofs and the definitions of ABFHE and spooky encryptions, which are formally described by GMM.

Lemma 3.17 (Existence of extended extractable IRHWE relative to $\mathcal{O}$ [GMM17, Lemma 7.9])**.** *The random IRHWE oracle defined in Definition 3.14 implements a correct and subexponentially-secure ex-EIRHWE defined in Definition 3.12.*

Corollary 3.18 (Existence of extended AB-FHE and extended spooky encryption relative to IRHWE oracle [GMM17, Lemmas 8.2 and 8.3])**.** *Both extended Attribute-Based FHE and extended spooky encryption are implied by extended extractable IRHWE and one-way functions, and thus they exist in the IRHWE oracle world.*

Remark 3.19 (Indirect queries are necessary for oracle separation)**.** In the IRHWE oracle world, allowing the d-queries to evaluate oracle-aided circuits is crucial. Otherwise, there is a simple PSPACE attacker that inverts any label $\mathsf{lb} = (a, y)$: just to find a witness w for the instance a (taking PSPACE but no query) and then to perform a d-query on the label using w, which would be even weaker than the random oracle.

3.5 Instance-Hiding Witness Encryption Oracle

In this subsection, we introduce the IHWE oracle $\mathcal{O}$ which consists of two suboracles: e and d_{V}. Similar to IRHWE, the suboracle e uses a random permutation. To achieve instance hiding, we use a single permutation for all instance-message pairs, incorporating the instance as part of the permutation input while the output of suboracle will not include the instance. The suboracle d_{V} is almost identical to the counterpart in the IRHWE oracle. The only difference is that the d_{V} will output the message along with instance. The IHWE oracle follows from GMM [GMM17, Definition 6.4], but we also define the *two-layer* restriction.

The following IHWE oracle is similar to the informal Definition 2.2 in the overview. The main difference is the additional zero-string 0^λ in the preimage, which ensures that valid labels are sparse and lengthens the label by λ bits additively.

Definition 3.20 (Random (Two-layer) Instance-Hiding Witness Encryption Oracles)**.** *Let $\mathsf{V}^{(\cdot)}$ be an oracle-aided universal circuit-evaluator that takes an instance a as well as a witness w and outputs 1 for acceptance or 0 for rejection, and let V' be the corresponding (plain) universal circuit evaluator. The* random instance-hiding witness encryption (IHWE) oracle *$\mathcal{O}$ is defined as follows. We specify the sub-oracle $\mathcal{O}_\lambda$ whose inputs are parameterized by λ, and the actual oracle will be $\mathcal{O} = \{\mathcal{O}_\lambda\}_{\lambda \in \mathbb{N}}$.*

- $\mathsf{lb} \leftarrow e(a, m)$ *takes as input an instance $a \in \{0,1\}^\alpha$ and a message $m \in \{0,1\}^\lambda$ where $\alpha := \alpha(\lambda)$ is a fixed polynomial. The suboracle e use a random permutation $\pi : \{0,1\}^{\alpha+2\lambda} \to \{0,1\}^{\alpha+2\lambda}$ and output* $\mathsf{lb} = \pi(a \| m \| 0^\lambda)$.

- $x/\bot \leftarrow d_{\mathsf{V}}(w, \mathsf{lb})$ *takes a ciphertext* $\mathsf{lb} \in \{0,1\}^{\alpha+2\lambda}$ *as input and outputs* x*. It works as follows.*
    1. *Find the preimage of the ciphertext, denoted as* (a, m, h) *where* $h \in \{0,1\}^{\lambda}$*, such that* $\pi(a\|m\|h) = \mathsf{lb}$.
    2. *If* $h \neq 0^{\lambda}$*, output* $\bot$*. Otherwise continue to the next step.*
    3. *If* $\mathsf{V}^{\mathcal{O}}(a, w) = 1$*, output* $x = (a, m)$*. Otherwise, output* $\bot$*.*

The IHWE oracle is restricted to two-layer *if the oracle-aided verification* $\mathsf{V}^{\mathcal{O}}(a, w)$ *is restricted to* $\mathsf{V}^{\mathcal{O}'}(a, w)$*, where* $\mathcal{O}' := \{\mathcal{O}'_{\lambda} := (e, d_{\mathsf{V}'})\}_{\lambda\in\mathbb{N}}$ *is the* plain *IHWE oracle. That is, the two-layer IHWE oracle* $\mathcal{O}$ *consists of the following* three *suboracles.*

- $\mathsf{lb} \leftarrow e(a, m)$ *is identical to the above extended IHWE.*
- $x/\bot \leftarrow d_{\mathsf{V}^{\mathcal{O}'}}(w, \mathsf{lb})$ *is similar to* d_{V} *in the above extended IHWE, but the only difference is that the oracle queries of* $\mathsf{V}^{(\cdot)}$ *is answered by* $\mathcal{O}'$*.*
- $x/\bot \leftarrow d_{\mathsf{V}'}(w, \mathsf{lb})$ *is similar to* d_{V}*, but the only difference is that the oracle-aided* $\mathsf{V}^{(\cdot)}$ *is replaced by the plain* V'*.*

The valid labels, direct and indirect queries *are defined similarly to that of IRHWE (Definition 3.15 and Definition 3.16), and the only difference is that there is no* eval*-queries.*

The above oracle is parameterized by a fixed polynomial $\alpha(\lambda)$, and fixing λ, the oracle $\mathcal{O}_{\lambda}$ takes only α-bit instances for the fixed α. This is w.l.o.g. because shorter instances can be padded to α bits, and longer instances can be handled by scaling up λ.

GMM proved that extended extractable IHWE exists in the (extended) IHWE oracle world, stated below. Notice that the number of layers provided by the IHWE oracle is preserved.

Lemma 3.21 (Existence of two-layer extractable IHWE relative to $\mathcal{O}$ [GMM17, Lemma 6.6])**.** *The two-layer random IHWE oracle* $\mathcal{O}$ *defined in Definition 3.20 implements a correct and subexponentially-secure two-layer-EIHWE defined in Definition 3.13.*

GMM also proved that extended predicate encryption (PE) can be constructed from extended extractable IHWE and one-way functions; we observe that the reduction also preserves the number of layers. Hence, we state the existence below.

Corollary 3.22 (Existence of two-layer predicate encryptions relative to IHWE oracle [GMM17, Lemma 8.1])**.** *Two-layer (attribute-hiding) predicate encryption is implied by two-layer EIHWE and one-way functions, and thus such PE exists in the two-layer IHWE oracle world.*

The definition of two-layer PE is omitted, but it is similar to standard (attribute-hiding) PEs, e.g., [GMM17, Definition 3.6]. The difference is that the predicate is evaluated by an oracle-aided universal circuit evaluator, where the evaluator can call PE in the oracle way and using only plain predicates—such a difference is essentially shown in two-layer IHWE (Definition 3.13).

Unlike the instance-*revealing* oracle (Remark 3.19), due to the instance-hiding property, even one-layer IHWE oracle (disallowing the circuit evaluator to call the oracle itself) is non-trivial and strictly stronger than the random oracle. Looking forward, ruling out IPFE schemes from the two-layer IHWEs is novel and involved (Section 5).

3.6 Skeleton of the Attacker

Our attackers follow the framework of Hajiabadi et al. [HLOW24], as mentioned in Section 2.2. This subsection describes the framework's detailed steps. Also, we augment the framework so that the attacker learns more information than that of Hajiabadi et al.; the augmented steps are colored in blue. The ideal oracle, denoted by $\mathcal{O}$, is opaque and will be instantiated later in Section 4 and Section 5. Similarly, the simulator, denoted by Sim, will be tailored later for the corresponding oracles.

Colluding phase: Corrupting keys and setting up the challenge.

1. Uniformly sample $\mathbf{v} \leftarrow \mathbb{Z}_p^n$ as the challenge secret-key vector.
2. Let $t, t_\text{c}, \tilde{t}_\text{c} \in \mathbb{N}$ be sufficiently large numbers in $\mathrm{poly}(\lambda)$, to be chosen in the analysis. Sample $(\mathbf{v}_1, \ldots, \mathbf{v}_t) \leftarrow \mathsf{S}$ as colluded vectors, where S is a random $(n-1)$-dimensional subspace of $\mathbb{Z}_p^n$ subject to $\mathbf{v} \notin \mathsf{S}$.
3. For all $i \in [t]$, send $\mathbf{v}_i$ to the IPFE challenger (non-adaptively) and then receive the functional key $\mathsf{sk}[\mathbf{v}_i]$.
4. For all $j \in [t_\text{c}+\tilde{t}_\text{c}]$, receive from the IPFE challenger (non-adaptively) the challenge ciphertext and inner-product pairs, (c_j, m'_j), where $m'_j \in \mathbb{Z}_p$. Let C_com to denote the last $\tilde{t}_\text{c}$ ciphertexts.

Learning phase: Learning essential query-answer pairs.

1. For each $i \in [t]$, execute the algorithm $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}_i], \tilde{c})$ for each $\tilde{c}$ in C_com to get Q^i_com.
2. For each $i \in [t]$, for each $j \in [t_\text{c}]$, execute the algorithm $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}_i], c_j)$ and record all the Q-A pairs that are made to $\mathcal{O}$ in the execution; let $Q^{i,j}_\mathsf{dec}$ be the set of Q-A pairs.

Simulation phase: Using the learned Q-A pairs to decrypt. This phase uses the learned Q-A pairs $Q^{i,j}_\mathsf{dec}$ and Q^i_com, it is PSPACE, but it never queries the oracle $\mathcal{O}$. Let $\mu := \mu(\lambda)$ be the output length of $\mathsf{KGen}^{\mathcal{O}}(\cdot)$ and ρ be the bit-length of all Q-A pairs made during $\mathsf{KGen}^{\mathcal{O}}(\cdot)$. A polynomial-time simulator Sim is initialized with all learned Q-A pairs, $Q^{i,j}_\mathsf{dec}$ and Q^i_com for all i, j.

1. For each $Q \in \{0,1\}^\rho$ of Q-A pairs appearing in the key generation KGen on the challenging vector $\mathbf{v}$ and for each resulting functional key $\mathsf{sk}'[\mathbf{v}] \in \{0,1\}^\mu$, attempt to decrypt the challenge ciphertexts as follows, using the simulator Sim.
    (a) For each $j \in [t_\text{c}]$,
        i. Simulate $\tilde{m}_j \leftarrow \mathsf{Dec}^{\mathsf{Sim}}(\mathsf{sk}'[\mathbf{v}], c_j)$; whenever Dec queries q, answer it with $\mathsf{Sim}(q)$.
    (b) If $\tilde{m}_j = m'_j$ for all $j \in [t_\text{c}]$, return 0, indicating that m'_j is the inner product of $\mathbf{v}$.
2. Otherwise, all trials of $(Q, \mathsf{sk}'[\mathbf{v}])$ are unsuccessful, return 1, indicating that m'_j is randomly sampled. This concludes the attacker.

Remark. Compared to the original framework of Hajiabadi et al., our augmentation obtains more ciphertexts, C_{com}, which are exploited to learn more about the colluded keys. Notice that for $\tilde{c} \in C_{\text{com}}$, its corresponding m'_j (for some $j > t_c$) is completely unused. This is preparing for the compilation of key generation (Solution 5, Section 2.5) and will be elaborated and utilized later in Section 5.

The analysis will rely on the combinatorial lemma (Lemma 2.3), restated below.

Lemma 3.23 (HLOW Lemma [HLOW24, Lemma 4.1])**.** *For any $\lambda \in \mathbb{N}$, let $n := n(\lambda) \geq 3$ be a polynomial of λ, p be a prime such that p^{-n} is negligible in λ. For any polynomial $\ell := \ell(\lambda)$, any function $F : \mathbb{Z}_p^n \to 2^{[\ell]}$, where $2^{[\ell]}$ is the power set of $[\ell]$, and any constant τ, there exists $t = \text{poly}(\lambda)$ such that*

$$\Pr\left[F(\mathbf{v}) \subseteq \bigcup_{i \in [t]} F(\mathbf{v}_i)\right] \geq 1 - \lambda^{-\tau},$$

where random variables $\mathbf{v}, \mathbf{v}_1, \ldots, \mathbf{v}_t$ are sampled by: uniformly sample vector $\mathbf{v} \leftarrow \mathbb{Z}_p^n$, sample a random $(n-1)$-dimensional subspace S such that $\mathbf{v} \notin \mathsf{S}$, and then sample vectors $\mathbf{v}_1, \ldots, \mathbf{v}_t \in \mathsf{S}$ uniformly at random.

Remark 3.24 (Applying HLOW Lemma on randomized IPFEs)**.** In the HLOW Lemma, F is a deterministic (mathematical) function, but we will compose F from the potentially randomized IPFE algorithms, $\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}$. This issue is inherited from Hajiabadi et al. and addressed below.

By standard technique, we assume Dec is deterministic w.l.o.g. For Enc, we always ensure that $F(\mathbf{u})$ invokes Enc independent of the input $\mathbf{u}$, so that the randomness of Enc can be hardwired in F. We will compose $F(\mathbf{u})$ that runs $\mathsf{KGen}(\ldots, \mathbf{u})$, so a randomized KGen needs more randomness. To this end, we hardwire a random function r in F, and then when $F(\mathbf{u})$ runs $\mathsf{KGen}(\ldots, \mathbf{u})$, we use $r(\mathbf{u})$ as the randomness. The randomness $r(\mathbf{u})$ is distributed identical to a uniform random string as long as distinct inputs $\mathbf{u}$ are given to F. In HLOW Lemma, the vectors $\mathbf{v}$ and $\mathbf{v}_1, \ldots, \mathbf{v}_t$ are distinct except with negligible probability by p^{-n} is negligible. Hence, the lemma holds for the above composition of F.

Henceforth in this paper, we assume w.l.o.g. that the IPFE is deterministic for readability.

4 Separating SK-IPFE from IRHWE

In this section, we prove that there is no black-box construction of SK-IPFE from IRHWE. Let $\mathcal{O}$ be an IRHWE oracle defined in Section 3.4. In this section, $\mathcal{E}^{\mathcal{O}} := (\mathsf{KGen}^{\mathcal{O}}, \mathsf{Enc}^{\mathcal{O}}, \mathsf{Dec}^{\mathcal{O}})$ denotes an oracle-aided IPFE for vectors in $\mathbb{Z}_p^n$. We demonstrate that there exists a polynomial-query attacker $\mathsf{Brk}^{\mathcal{O}}$ which breaks the security of the given IPFE scheme in Section 4.2. The main theorem for this section is stated as follows.

Theorem 4.1. *Let λ be the security parameter, and suppose $n := n(\lambda) \geq 3$ and p^{-n} is negligible. There is no $\mathbb{Z}_p^n$-SK-IPFE construction (Definition 3.9) relative to the IRHWE oracle in the monolithic model.*

Putting together Theorem 4.1, Lemma 3.17, and Lemma 3.8, we separate IPFE from IRHWE as the following corollary, which implies our main result, Theorem 1.1, given that ABE, FHE, and AB-FHE are implied by IRHWE (Corollary 3.18).

Corollary 4.2. *Let $\lambda \in \mathbb{N}$ be the security parameter. For any dimension $n = n(\lambda) \geq 3$ and any prime $p = p(\lambda)$ such that p^{-n} is negligible in λ, there is no monolithic construction of $\mathbb{Z}_p^n$-SK-IPFE from extended extractable IRHWE.*

While the theorem statement holds only for $n \geq 3$, here is a brief discussion on $n < 3$. When $n = 0$, there is no message, and the encryption is trivial. When $n = 1$, the IPFE is equivalent to a secret-key encryption because colluding any functional key associated with a non-zero vector is sufficient to decrypt all messages; w.l.o.g., the zero-vector functional key is always an all-zero string. When $n = 2$, the adversary may collude at most p functional keys on the same line while being unable to learn all messages, and thus p-collusion security is possible.

Summary of challenges. Recall that in the security game of IPFE (Definition 3.9), the attacker receives the challenge ciphertexts and "messages" from the challenger. The goal of the attacker is to distinguish whether the messages are inner-products ($b = 0$) or randomly chosen values ($b = 1$). Hence, the goal is to show that $\mathsf{Brk}^{\mathcal{O}}$ will guess the value b correctly with probability much greater than $1/2$ for each case b. To this end, we construct Brk and prove that $\mathsf{Brk}^{\mathcal{O}}$ can correctly "perform" the functional decryption on the challenge ciphertexts with high probability. The framework of our $\mathsf{Brk}^{\mathcal{O}}$ follows the attacker of Hajiabadi et al. [HLOW24], which consists of three phases, also illustrated in Section 3.6. Firstly, the attacker samples the challenge and colluded vectors, interacting with the challenger to obtain colluded functional keys. Secondly, in the learning phase, the attacker runs the decryption algorithm with the colluded functional keys and learns those "essential" query-answer (Q-A) pairs from the oracle. Thirdly, in the simulation phase, the attacker exhausts all potential functional keys as well as the Q-A pairs used to generate the functional key and then simulates the functional decryption.

Because the IRHWE oracle is much more powerful than the random oracle, as mentioned in Section 2.4, there are several challenges to resolve, and we summarize them below.

The first challenge is to learn the "essential" queries that are incurred during the encryption of the challenge ciphertext. We say a query q performed during the encryption is *essential* if the decryption queries some information that depends on q or its answer. The dependency was minimal in the random oracle world or even trapdoor permutation. For example, in the random oracle world, each query is completely independent. For trapdoor permutations, the dependent Q-A pairs were called "normal form" [BPR$^+$08] or "closure" [GKLM12], which includes only the inverse of each query, a constant number. The IRHWE world is totally different because $eval$-queries introduce dependency on many Q-A pairs. Below Example 4.3 gives a potential IPFE that incurs such dependency. Since the closure is too large, it is impossible to learn the essential queries in the learning phase. For these dependent Q-A pairs, we take a different approach: we consider the *labels and their instances* (but not queries) yielded during the encryption, and we aim to learn the witnesses that verify the instances successfully through d-queries during the decryption. The witness and the d-query may be performed on a different label, but as long as the *instance is the same*, the attacker can learn the preimage of the label using the same witness and an extra d-query to the real oracle. There are polynomially many instance-witness pairs, and we will show that all such pairs can be learned using HLOW Lemma.

Example 4.3. $\mathsf{IPFE.Enc}$ queries $e(a, x)$ for some (a, x) and gets lb_1, and then it queries $eval_f(\mathsf{lb}_1)$ and gets lb_2. Afterward, $\mathsf{IPFE.Dec}$ queries $eval_f(\mathsf{lb}_2, \mathsf{lb}_3)$ for some lb_3 and gets lb_4, and then it queries $d_{\mathsf{V}}(w, \mathsf{lb}_4)$ for some w. All above queries and potentially others depend on $((e, a, x), \mathsf{lb}_1)$.

The second challenge arises from the monolithic model, where an oracle query may trigger *indirect* oracle queries (Definition 3.16). Because indirect queries are performed in the oracle internally, the attacker observes only the direct queries in the learning phase. However, in the simulation phase, to correctly answer a direct query, the attacker needs to know the corresponding indirect queries because the answer depends on the indirect queries. Hence, learning only the direct queries is insufficient. See Example 4.4 below.

Example 4.4. Let a, x_2, x_4 be strings and lb_1 be a label. Consider the the following $\mathsf{IPFE.Dec}$. $\mathsf{IPFE.Dec}$ queries $eval_f$ on lb_1 and gets lb_3, but such f indirectly queries $e(a, x_2)$ and gets lb_2; then $\mathsf{IPFE.Dec}$ queries $d_{\mathsf{V}}(w, \mathsf{lb}_3)$ for some w, but such V indirectly queries $d_{\mathsf{V}}(w', \mathsf{lb}_2)$ for some w'. The indirect queries affect the output of the evaluator f and the verification V. Also, learning $\mathsf{V}(a, w')$ is tricky even for a fixed a because there can be many w' such that $\mathsf{V}(a, w') = 1$; e.g., $\mathsf{V}(a, w')$ is insensitive to the second half of w'.

To resolve it, we borrow the idea and terminology of "canonicalization" from Garg, Mahmoody, and Mohammed [GMM17]. The high-level idea is to compile an oracle-aided algorithm (of IPFE), so that the compiled algorithm performs all indirect queries that the compiler "knows." That is, using the previous queries, if the compiler knows all inputs of f associated with an $eval$-query or V associated with a d-query, then the compiler emulates the oracle-aided f or V and forwards the queries to the oracle. Although the high-level idea is similar, our compilation is tailored to our attacker and analysis in the following ways. Firstly, whenever a d-query is answered successfully by the oracle, our compiler learns a legitimate witness to the instance, and it performs extra d-queries on all other labels of the same instance. Such extra d-queries give all plaintexts of the same-instance labels and address the previous challenge of dependent labels. Secondly, after the additional d-queries, our compiler knows more inputs of f and thus emulates such f, so that it learns more indirect queries in running f. Thirdly, our compiler ensures that all queries it makes follow a proper ordering. That is, each query should be performed after all the queries it depends on. The ordering is beneficial for the hybrid argument of our analysis later: simulating the answer to a query only needs its preceding queries.

The third challenge comes from a subtle case that an $eval$-query can introduce dependencies among labels that were *unrelated if the $eval$-query was not performed*. Even worse, if the $eval$-query is performed only in the generation of the challenge functional key (KGen), then there is no way to learn the introduced dependency from the colluded keys; see Example 4.5 below. Also, unlike the first challenge with dependent queries, the attacker cannot perform any extra d-query to resolve this issue. It is because the attacker exhausts each secret key and its Q-A pairs in the simulation phase, so it would be too many extra oracle queries. Fortunately, such dependency is introduced only when an arbitrary string coincidentally is a valid label. Because the valid labels are random and sparse, it happens with only negligible probability over the randomness of IRHWE oracles.

Example 4.5. Fix a challenge vector $\mathbf{v}$ in KGen and thus Dec below. Suppose that $\mathsf{IPFE.KGen}$ on $\mathbf{v}$ queries $eval_f(\mathsf{lb}_1)$ and gets lb_2 for some string lb_1, where lb_1 depends on $\mathbf{v}$, but lb_1 is *without* making any oracle query. Also suppose that $\mathsf{IPFE.Enc}$ queries $e(a, x)$ for some a, x and gets lb_1. Afterward, $\mathsf{IPFE.Dec}$ queries $d_{\mathsf{V}}(w, \mathsf{lb}_2)$ for some w.

In Section 4.1, we provide a detailed description of our canonicalization and explain how it resolves the first two challenges. After that, we give our attack Brk, which utilizes the canonicalization, followed by the proof of separating IPFE from IRHWE.

4.1 Canonical Execution of Oracle-Aided Algorithms

We describe how to compile any oracle-aided algorithm $A^{\mathcal{O}}$ relative to IRHWE oracle $\mathcal{O}$ to a "canonical" form $\bar{A}^{\mathcal{O}}$. Recall that the IRHWE oracle evaluates oracle-aided circuits, which may call the IRHWE itself (see $eval_f$ and d_{V} in Definition 3.14). Conceptually, the canonical form is to additionally evaluate the oracle-aided circuits *directly from* $\bar{A}$. Of course, some circuit evaluation performed by the IRHWE oracle can be totally unknown to the original algorithm A, and in that case, the canonical form simply skips the circuit. Hence, the compiler expands the original by reusing the information known to the original, and the compiled functionality is identical. Looking forward, we will compile any IPFE into its canonical form, which preserves the security. The purpose will be to simplify the attacker and the analysis.

The canonical compilation is conceptually simple. While executing the original algorithm, we maintain all previous Q-A pairs to the oracle. Using the list of Q-A pairs, we infer the circuit evaluation in the next oracle query, to the best we know, and performs the indirect oracle queries yielded during the evaluation. We expand recursively when the indirect query further evaluates another circuit.

In the following, we begin with the algorithm Entail, which finds the corresponding plaintext of a given label from a given Q-A list. Each Q-A pair is denoted as a tuple of suboracle, query, and answer, e.g., $((eval_f, \mathsf{lb}_1, \mathsf{lb}_2, \ldots, \mathsf{lb}_l), \mathsf{lb})$ is an $eval$ query on input $(\mathsf{lb}_1, \mathsf{lb}_2, \ldots, \mathsf{lb}_l)$ and gets an answer lb. When a given label is in an e-type or d-type Q-A pair, the plaintext is always found in the same pair. When the label is the answer of a $eval$-query, we repeatedly find the plaintexts of the input labels, and we use the input plaintext to infer the output plaintext. Entail is formalized below.

Definition 4.6 (Queries in a Q-A pair set)**.** *Let S be a set of query-answer (Q-A) pairs. Define* $\mathsf{Query}(S)$ *to be the set of all queries appearing in S, that is,* $\mathsf{Query}(S) := \{q : (q, a) \in S \text{ for some } a\}$.

Algorithm 5. $\mathsf{Entail}(Q, \mathsf{lb}_{\text{in}})$

Input: A set Q consisting of Q-A pairs of IRHWE oracle, and a label $\mathsf{lb}_{\text{in}} \in \{0,1\}^*$
Procedure: Initialize L as an empty list. For each Q-A pair $q \in Q$, perform the following:

1. If q is d-type, $q = ((d_{\mathsf{V}}, w, \mathsf{lb}), m)$, add (lb, m) into L.
2. If q is e-type, $q = ((e, a, m), \mathsf{lb})$, add (lb, m) into L.
3. If q is $eval$-type of the form $q = ((eval_f, \mathsf{lb}_1, ..., \mathsf{lb}_l), \mathsf{lb})$, where $\mathsf{lb}_i = (a_i, y_i)$ for each $i \in [l]$ and $\mathsf{lb} = (a, y)$, perform:
   (a) If $a_i \neq a$ for some i, skip such q.
   (b) For each $i \in [l]$, find $(\mathsf{lb}_i, m_i) \in \mathsf{L}$ for some m_i. If any m_i is not found, skip such q.
   (c) Evaluate the oracle-aided circuit $f^{(\cdot)}(m_1, \ldots, m_l)$; whenever the circuit queries the oracle, try to find the corresponding answer in Q. If an answer is not found, skip such q.
   (d) Let m be the evaluation of $f^{(\cdot)}$. Add (lb, m) into L.

Finally, find and return m_{out} if there exists $(\mathsf{lb}_{\text{in}}, m_{\text{out}}) \in \mathsf{L}$; otherwise, return $\mathsf{NotFound}$.

In Entail, each attempt to evaluate $f^{(\cdot)}$ takes polynomial time. Also, each $q \in Q$ adds at most one new element into L. Hence, the number of repetitions and thus the running time of Entail is polynomial. Note that Entail never queries any oracle.

Next, we formalize the canonical form by the compilation. Although we borrowed the idea and terminology of "canonicalization" from Garg, Mahmoody, and Mohammed [GMM17], our compilation is tailored to our attacker and analysis. Specifically, in our canonical form, for each d-query q, we want q to be performed *after* performing all "known" queries that q depends on. Recall that a d-query, $d_{\mathsf{V}}(w, \mathsf{lb} = (a, y))$, is answered by two steps, verifying $\mathsf{V}(a, w) = 1$ and then finding the preimage of y. The d-query depends both on the indirect queries incurred during $\mathsf{V}(a, w)$ and on the queries that output lb directly or indirectly. Our compilation makes such precursor queries directly to the oracle whenever they can be inferred from the history of queries. Looking forward, making a precursor query direct and earlier alleviates the hybrid argument in our analysis—it is easier to simulate a subsequent query when all of its precursors are provided.

The canonicalization is formally given in Algorithm 6, which is similar to the informal Algorithm 1 in the overview. The main differences are the detailed steps of inferring the preimage in a d-query.

Algorithm 6. Canonicalize$^{\mathcal{O}}(A)$

Input: An oracle-aided algorithm $A^{(\cdot)}$, which may include the input to A implicitly.
Oracle: An oracle $\mathcal{O}$ is given.
Procedure: Initialize an empty list Q. Emulate $A^{(\cdot)}$ and respond to every (direct or indirect) query q as follows:

1. If q is e-type, query oracle $a \leftarrow \mathcal{O}(q)$, and add the Q-A pair (q, a) into Q.
2. If q is $eval$-type of the form $eval_f(\mathsf{lb}_1, \cdots, \mathsf{lb}_l)$ for some l:
   (a) Before querying $\mathcal{O}$, run $m_i \leftarrow \mathsf{Entail}(Q, \mathsf{lb}_i)$ for every $i \in [l]$. If $m_i \notin \{\mathsf{NotFound}, \perp\}$ for all $i \in [l]$, emulate $f^{(\cdot)}(m_1, ..., m_l)$; respond to all queries recursively using this procedure and the list Q.
   (b) Query oracle $a \leftarrow \mathcal{O}(q)$, and add the Q-A pair (q, a) into Q.
3. If q is d-type of the form $d_{\mathsf{V}}(w, \mathsf{lb} = (a, y))$:
   (a) Before querying $\mathcal{O}$, emulate $\mathsf{V}^{(\cdot)}(a, w)$; respond to all queries recursively using this procedure and the list Q. Let v be the result.
   (b) (*Perform DFS.*) If $v = 1$, and $\mathsf{Entail}(Q, \mathsf{lb}) = \mathsf{NotFound}$, but there exists a Q-A pair $((eval, \mathsf{lb}_1, ..., \mathsf{lb}_l), \mathsf{lb}) \in Q$ that outputs lb, perform a depth-first search (DFS) as follows (otherwise, continue to the next step).[a] Let G be a directed graph induced by Q, where each label in any Q-A pair in Q is a vertex; for any pair of vertices $\mathsf{lb}_a, \mathsf{lb}_b$, if there is a $eval$-query in Q such that lb_b is an input and lb_a is an output label of the $eval$, add the arrow $(\mathsf{lb}_a, \mathsf{lb}_b)$ to G. That is, each edge goes from the output label to its input labels. The DFS traverses all the successors of lb on G, performing a d-query and an $eval$-query when visiting a label. That is, starting from $\mathsf{lb}' = \mathsf{lb}$, perform below:

i. Visit each child of lb' using this DFS, 3(b)i–3(b)iii (skip if there is no child).

ii. If lb' is the output of an $eval$-query $((eval_f, \mathsf{lb}'_1, \ldots, \mathsf{lb}'_{l'}), \mathsf{lb}') \in Q$ (equivalently, lb' has children vertices), recursively invoke Step 2 on the query $eval_f(\mathsf{lb}'_1, ..., \mathsf{lb}'_{l'})$.

iii. Query oracle $a \leftarrow \mathcal{O}(d_{\mathsf{V}}, w, \mathsf{lb}')$, and add the Q-A pair $((d_{\mathsf{V}}, w, \mathsf{lb}'), a)$ into Q.

(c) Query oracle $a \leftarrow \mathcal{O}(q)$, and add the Q-A pair (q, a) into Q.

Finally, the canonicalization outputs the output of the emulated $A^{(\cdot)}$.

[a]For readability, suppose that $eval$-query is always single-hop, i.e., an output of any $eval$ is never the input to any $eval$. Then, the depth of DFS is simplified to one. The simplification is without loss of generality because we can compile from multi-hop to single-hop—simply encode the former $eval$ into the circuit of the latter $eval$ by invoking extra d-queries.

We argue that for any $A^{\mathcal{O}}$ that runs in polynomial time and thus performs a polynomial number queries, the canonicalization $\overline{A}^{\mathcal{O}} := \mathsf{Canonicalize}^{\mathcal{O}}(A)$ also runs in polynomial time. We begin with a few terminologies. Recall that a query is *direct* if it is performed by A, and that query is *indirect* if it is performed by $\mathcal{O}$ through a circuit evaluation in a d-type or $eval$-type query (Definition 3.16). An indirect query may invoke another indirect query. Due to the constraint of circuit sizes, the input length of a caller query must be longer than the total input length of its callee queries. Hence, during the execution of $A^{\mathcal{O}}$, the total number of oracle queries, including direct and indirect, is polynomial in the number of direct queries performed by A, as claimed by [GMM17]. This implies that the total number of labels in all queries is also polynomial. Henceforth, we omit the number of queries.

To analyze the canonicalization $\overline{A}^{\mathcal{O}}$, we claim that, if we omit Step 3b, then the running time is polynomial. The challenge is to include Step 3b, DFS, which differs from [GMM17]. It is because in Step 3, the $eval$-query that outputs lb can be longer than the d-query since the two queries are unrelated in terms of caller-callee. Moreover, in Step 3(b)ii, we may recursively enter the DFS step again when $eval$ emulates f that in turn invokes another d-query. We use the following invariants.

- Invariant 0: If $\mathsf{Canonicalize}$ never enters DFS, it finishes in polynomial time.
- Invariant 1: If $\mathsf{Canonicalize}$ entered DFS with a label lb and then finished, then $\mathsf{Canonicalize}$ will never enter DFS again with the same lb.
- Invariant 2: If $\mathsf{Canonicalize}$ entered DFS with a label lb and then recursively entered another DFS with label lb' before finishing lb, then it holds that $\mathsf{lb}' \neq \mathsf{lb}$.

The Invariants imply that the running time is polynomial. To see why, consider any DFS that recursively entered (another) DFS. By Invariant 2, the labels are distinct. Since DFS always started from a label in Q, and there are at most a polynomial number of distinct labels, the recursive depth of any DFS must be polynomial. By Invariant 1, once we leave a DFS on a label, we never perform DFS again on the same label. Hence, every label in G invokes DFS at most once through $\mathsf{Canonicalize}$, no matter how it is invoked. By Invariant 0, the total time is polynomial.

It remains to prove the Invariants. Invariant 0 has already been argued as above. Invariant 1 follows as $\mathsf{Canonicalize}$ always queried oracle $\mathcal{O}$ on lb and added the query to Q before finishing

DFS at Step 3(b)iii; henceforth, $\mathsf{Entail}(Q, \mathsf{lb})$ will no longer be $\mathsf{NotFound}$, so DFS can never enter the same lb again.

For Invariant 2, we define and quantify a undesirable event Derive-Hit as follows. For any two labels $\mathsf{lb}_\mathrm{a}, \mathsf{lb}_\mathrm{b}$ and any algorithm A, we say that lb_a *derives* lb_b w.r.t. all the queries made during $A^{\mathcal{O}}$ recursively as below.

- For some $eval$ query on $\mathcal{O}$, lb_a is one of the input labels, and lb_b is the output label.
- For some $eval$ query q, the homomorphic evaluation f of q directly or indirectly invokes a query $d_{\mathsf{V}}(\cdot, \mathsf{lb}_\mathrm{a})$, and lb_b is the output label of q.
- For some label $\mathsf{lb} \neq \mathsf{lb}_\mathrm{a}$ and $\mathsf{lb} \neq \mathsf{lb}_\mathrm{b}$, it holds that lb_a derives lb and lb derives lb_b.

We define Derive-Hit to be the event that there exists a label lb_a derives itself in the queries made during $A^{\mathcal{O}}$. The intuition is that if lb_a derives lb_b, then lb_b is sampled uniformly at random by $\mathcal{O}$ and thus $\mathsf{lb}_\mathrm{b} \neq \mathsf{lb}_\mathrm{a}$. Notice that the ordering of queries does not matter as long as A learns only a polynomial number of queries.

Claim 4.6.1 (Derive-Hit is a negligible event)**.** *Let A be an oracle-aided uniform PPT algorithm hardwired with an oracle-dependent input, where the input is generated by an uniform PPT algorithm $A_0^{\mathcal{O}}(1^\lambda)$. For any such A, for any labels $(\mathsf{lb}_\mathrm{a}, \mathsf{lb}_\mathrm{b})$, if lb_a derives lb_b w.r.t. $A^{\mathcal{O}}$, then $\mathsf{lb}_\mathrm{a} \neq \mathsf{lb}_\mathrm{b}$ except with negligible probability in λ over the randomness of $\mathcal{O}$.*

Proof sketch. Observe that in the two base cases where lb_a derives lb_b in one $eval$ query, every distinct lb_a requires a distinct input to the $eval$, which implies that lb_b is sampled randomly by the oracle (Definition 3.20). Moreover, if there is a label derives itself, there is a cycle of labels that derive each other. Considering the chronological ordering of the labels in such a cycle, there must be an $eval$ query outputs a label identical to another chronologically earlier label. Using a polynomial number of queries, that happens with exponentially small probability over the random choice of the output label. Because A_0 gives A an oracle-dependent input, we consider all queries made by both A_0 and A. Still, there are at most a polynomial number of queries. Hence, there is no self-deriving label in the execution of $A^{\mathcal{O}}$ except with negligible probability. □

We next prove Invariant 2 conditioned on that Eval-Hit never happen in the original $A^{\mathcal{O}}$. We trace the oracle queries closely as follows. Suppose that a d-query q_1 entered DFS with input lb_1. Let query q_2 on lb_2 be the first d-query entered DFS recursively after q_1. Such q_2 must be triggered (remotely) by Step 3(b)ii in an $eval$-query recursively; let q_3 be the $eval$, and let lb_3 be the output of the $eval$. Because in the execution of the original $A^{\mathcal{O}}$, q_2 is invoked by q_3 directly or indirectly, we have that lb_2 *derives* lb_3. Moreover on G, lb_3 is either lb_1 or a successor of lb_1, and in either cases, we have that lb_2 *derives* lb_1. Conditioned on the no-Derive-Hit event, $\mathsf{lb}_2 \neq \mathsf{lb}_1$. The deeper recursive DFS invocations is argued inductively—e.g., if q_2 entered yet another DFS for some query q_4 and lb_4, then lb_4 derives lb_2 and thus lb_1, so that $\mathsf{lb}_4 \neq \mathsf{lb}_1$.

We conclude the canonicalization below. The exponentially small probability quantifies the Derive-Hit event, as per Claim 4.6.1. The Lemma additionally states two properties on the queries, where Property 1 below follows by Step 3a, and Property 2 follows by Step 3(b)ii.

Lemma 4.7. *Let $\mathcal{O}$ be the IRHWE oracle. For any uniform PPT oracle-aided algorithm A_0, any polynomial-time oracle-aided algorithm $A^{\mathcal{O}}$ that is uniform but hardwired an input computed by $A_0^{\mathcal{O}}(1^\lambda)$, the canonical* *form $\overline{A}^{\mathcal{O}} := \mathsf{Canonicalize}^{\mathcal{O}}(A)$ (Algorithm 6) is functionally equivalent to $A^{\mathcal{O}}$ and runs in polynomial-time except with a negligible probability over the choice of the IRHWE oracle $\mathcal{O}$. Moreover, the list of Q-A pairs Q performed by $\overline{A}^{\mathcal{O}}$ and any d-query $(q, \cdot) \in Q$ satisfies the properties below.*

1. *Represent $q = d_{\mathsf{V}}(w, \mathsf{lb} = (a, y))$ for some a, y. The verification $V^{(\cdot)}(a, w)$ can be evaluated solely using Q (without additional oracle query).*

2. *If label lb is successfully inverted w.r.t. $\mathcal{O}$ and there exists an $eval$-query q' that outputs lb in Q, then the circuit $f^{(\cdot)}(\cdots)$ associated with q' can be evaluated solely using $Q^{[-q]}$ where $Q^{[-q]}$ is the subset of Q that consists of all queries that chronologically precede q.*

Remark 4.8 (Canonical form is easier to simulate)**.** Looking forward, we will consider the canonicalization of all IPFE algorithms, $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, and our attacker will simulate the oracle for the canonicalized Dec. Hence, A is Dec and its corresponding input, and A_0 is the security game where the attacker and challenger jointly prepared the oracle-dependent input.

The attacker wants to correctly simulate the answers to d-queries in Dec. To do so, for each d-query of the form $d_{\mathsf{V}}(w, \mathsf{lb} = (a, y))$, we need to know the verification $V(a, w)$ and the plaintext of lb. Hence, when lb comes from $eval_f(\cdots)$, we need to know the evaluation of such $f(\cdots)$. That is why we want Property 2. Next, learning $V(a, w)$ is tricky because obtaining only the output for the (a, w) pair is insufficient. To see why, for a fixed lb and thus a fixed instance a, there can be many distinct w such that $V(a, w) = 1$; e.g., $V(a, w)$ is insensitive to some part of w. Hence, it is better to learn also the indirect queries of V, as per Property 1, so that $V(a, w)$ can be simulated for another w.

To correctly simulate the answers to d-queries in our attacker, we not only need the Q-A pairs but also a good ordering of the pairs in Q. It is because of the hybrid argument in our analysis. With a good ordering of queries in the canonical form, we will simulate a d-query while keeping its dependent queries (from the evaluation of V and f) as real oracle queries in the hybrid experiment. The Properties ensure the ordering between queries and thus the ordering of hybrids. The canonicalization simplifies the attacker later.

4.2 Detailed Attack, Brk

In this subsection, we describe how our attacker Brk can break an oracle-aided *canonical* IPFE scheme $(\mathsf{KGen}^{\mathcal{O}}, \mathsf{Enc}^{\mathcal{O}}, \mathsf{Dec}^{\mathcal{O}})$, where $\mathcal{O}$ is an IRHWE oracle. That is, we suppose that the IPFE algorithms are already canonicalized as per Algorithm 6 (Section 4.1). The attacker Brk is provided with all the public parameters of the given IPFE scheme, listed as follows. For an oracle-aided IPFE scheme, let λ denote the security parameter, p be the modulus, and n be the dimension of the vectors. Also in the IPFE, during the generation of any functional secret key, let $\mu := \mu(\lambda)$ be the bit-length of the functional secret key in the given IPFE scheme, and let $\rho := \rho(\lambda)$ be the bit-length of all queries and answers to/from the oracle $\mathcal{O}$. That is, $\mathsf{KGen}^{\mathcal{O}}$ outputs a μ-bit key, and the list of all queries and their answers is described using at most ρ bits. The attacker is given λ, p, n, μ, ρ as well as the oracle $\mathcal{O}$ in the security game against the challenger (Definition 3.9).

The attacker proceeds in three phases. In Colluding phase, the attacker interacts with the challenger to obtain colluded functional keys and challenge ciphertexts. In Learning phase, the attacker collects a sufficient set of Q-A of the oracle by running the decryption algorithm with the colluded functional keys and the challenge ciphertexts. In this phase, Brk learns the Q-A pairs by directly querying the oracle $\mathcal{O}$, where these Q-A pairs serve as the foundation for simulating the oracle in the next phase. Finally, in Simulation phase, the attacker attempts to decrypt the challenge ciphertexts. It takes PSPACE but never queries the real oracle $\mathcal{O}$ in this phase, and the answers to all queries are simulated by the attacker.

Colluding phase: Corrupting keys and setting up the challenge.

1. Uniformly sample $\mathbf{v} \leftarrow \mathbb{Z}_p^n$ as the challenge secret-key vector.
2. Let $t, t_c \in \mathbb{N}$ be sufficiently large numbers in $\mathrm{poly}(\lambda)$, to be chosen in the analysis. Sample $(\mathbf{v}_1, \ldots, \mathbf{v}_t) \leftarrow \mathsf{S}$ as colluded vectors, where S is a random $(n-1)$-dimensional subspace of $\mathbb{Z}_p^n$ subject to $\mathbf{v} \notin \mathsf{S}$.
3. Send $(\mathbf{v}_1, \ldots, \mathbf{v}_t, \mathbf{v})$ to the IPFE challenger (non-adaptively) and then receive the functional keys $(\mathsf{sk}[\mathbf{v}_i] : i \in [t])$.
4. For all $j \in [t_c]$, receive from the IPFE challenger (non-adaptively) the challenge ciphertext and inner-product pairs, (c_j, m'_j), where $m'_j \in \mathbb{Z}_p$.

Learning phase: Learning essential query-answer pairs.

1. For each $i \in [t], j \in [t_c]$, execute the algorithm $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}_i], c_j)$ and record all the query-answer (Q-A) pairs that are made to $\mathcal{O}$ in the execution; let $Q_{\mathsf{dec}}^{i,j}$ be the set of Q-A pairs.
2. Let $\bar{Q}_{\mathsf{dec}}^{i} := \bigcup_{j \in [t_c]} Q_{\mathsf{dec}}^{i,j}$ and let $\bar{Q}_{\mathsf{dec}} := \bigcup_{i \in [t]} \bar{Q}_{\mathsf{dec}}^{i}$ for short.

Simulation phase: Using the learned Q-A pairs to decrypt. This phase uses the learned Q-A pairs $\bar{Q}_{\mathsf{dec}}$ and runs in PSPACE, but it never queries the oracle $\mathcal{O}$. Recall that μ denotes the output length of $\mathsf{KGen}^{\mathcal{O}}(\cdot)$ and ρ denotes the bit-length of all Q-A pairs appearing during (the canonical form) $\mathsf{KGen}^{\mathcal{O}}(\cdot)$.

1. Try every possible set Q of Q-A pairs appearing in the key generation KGen on the challenging vector $\mathbf{v}$, and let $\mathsf{sk}'[\mathbf{v}]$ be the resulting secret key for the given Q. That is, for each $\mathsf{sk}'[\mathbf{v}] \in \{0,1\}^{\mu}$ and each $Q \in \{0,1\}^{\rho}$, simulate the decryption on the ciphertexts using $\mathsf{sk}'[\mathbf{v}]$ as follows.
   (a) For each $j \in [t_c]$,
      i. Initialize $\mathsf{Sim} := \mathsf{Sim}_{\mathrm{IRHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q]$ with $\bar{Q}_{\mathsf{dec}}$ and Q, where $\mathsf{Sim}_{\mathrm{IRHWE}}$ is a stateful algorithm, given later in Algorithm 7.
      ii. Simulate $\tilde{m}_j \leftarrow \mathsf{Dec}^{\mathsf{Sim}}(\mathsf{sk}'[\mathbf{v}], c_j)$ (whenever Dec queries q, answer it with $\mathsf{Sim}(q)$).
   (b) If $\tilde{m}_j = m'_j$ for all $j \in [t_c]$, return 0.
2. Otherwise, all trials of $(Q, \mathsf{sk}'[\mathbf{v}])$ are unsuccessful, return 1.

4.2.1 Simulated IRHWE Oracle, $\mathsf{Sim}_{\mathrm{IRHWE}}$

For any Q-A list Q, the simulated oracle $\mathsf{Sim}_{\mathrm{IRHWE}}[Q]$ is a stateful Turing Machine, and its internal state S_Q is initialized by Q and is updated after each query. The procedure is described in Algorithm 7. At a high level, answering an e-query or an $eval$-query is simply sampling an unused label. Hence, the simulator will use the set of used labels, which is computable in polynomial time.

Definition 4.9 (Labels of an instance)**.** *Given a list of Q-A pairs Q and any instance a, the set $\mathsf{Label}_a(Q)$ is defined as the set of y such that $\mathsf{lb} = (a, y)$ appears in any Q-A pair in Q.*

To answer a d-query on a label lb correctly, recall that the answer depends on both the verification of the witness and the underlying plaintext of lb. We would like our simulator to achieve the following informal properties.

1. The underlying plaintext of lb can be entailed from the state of the simulator, S_Q.
2. The answers to all queries of the verification can be simulated recursively using S_Q.

Since the $\mathsf{IPFE.Dec}$ is canonicalized, if all previous queries are answered correctly, then the above two properties are both satisfied for the current d-query; we will argue it formally later in Section 4.6 using Lemma 4.7.

Algorithm 7. $\mathsf{Sim}_{\mathrm{IRHWE}}[Q](q)$

Initialize the internal state: A state $S_Q := Q$ is initialized to the list Q.
Answer the query q**:** Once initialized, the simulator takes an input query q, simulates an answer to q, and then the current Q-A pair is always appended to the state S_Q, so that a future answer can be consistent with the current answer. The answer is simulated using the following procedure.

1. If q is identical to a query of a Q-A pair in S_Q, return the corresponding answer. Else, we describe below how to answer q when q is not a query in S_Q.
2. If q is an e-query of the form $e(a, m)$, uniformly sample $y \in \{0,1\}^{3\lambda} \setminus \mathsf{Label}_a(S_Q)$ and answer $\mathsf{lb} = (a, y)$.
3. If q is an $eval$-query of the form $eval_f(\mathsf{lb}_1 = (a_1, y_1), \ldots, \mathsf{lb}_l = (a_l, y_l))$ for some l:
   (a) If there are $i, j \in [l]$ such that $a_i \neq a_j$, return $\perp$ and abort.
   (b) Otherwise, let $a = a_1 = \cdots = a_l$, uniformly sample $y \in \{0,1\}^{3\lambda} \setminus \mathsf{Label}_a(S_Q)$, and answer $\mathsf{lb} = (a, y)$.
4. If q is a d-query of the form $d_{\mathsf{V}}(w, \mathsf{lb} = (a, y))$:
   (a) Let $x \leftarrow \mathsf{Entail}(S_Q, \mathsf{lb})$ If $x \in \{\mathsf{NotFound}, \perp\}$, answer $\perp$ (recall that Entail, Algorithm 5, outputs $\perp$ when lb is indeed invalid). Otherwise, $x \notin \{\mathsf{NotFound}, \perp\}$, execute the following steps.
   (b) Simulate the output of $\mathsf{V}^{(\cdot)}(a, w)$, where every query q' asked by V is recursively answered by this procedure $\mathsf{Sim}_{\mathrm{IRHWE}}$ with the current state S_Q. If V outputs 0, answer $\perp$. Otherwise, return x.

Notice that this algorithm is similar to the informal Algorithm 2, where the main difference is the formalization of Entail and Label_a.

4.2.2 Parameters

Setting parameters. We choose parameters n, p, t_{c}, t here.

- Recall that λ is the security parameter. For a $\mathbb{Z}_p^n$-SK-IPFE, $n := n(\lambda) \geq 3$ is a polynomial of λ. p is a prime number.
- $t_{\mathrm{c}} := t_{\mathrm{c}}(\lambda)$ is the number of ciphertext in the security experiment. Set $t_{\mathrm{c}} \geq \lambda + \mu + \rho$.
- Let $\ell := \ell(\lambda)$ be the number of queries made in each algorithm in the IPFE scheme (including all direct and indirect queries). Let τ be a constant such that $\lambda^\tau \geq 4 t_{\mathrm{c}} \lambda$. Choose t based on ℓ and 2τ as in the HLOW Lemma (Lemma 2.3).

Remark 4.10. The oracle we simulated has subtle differences from the real IRHWE oracle. For a new d-type query in the form of $d_{\mathsf{V}}(w, \mathsf{lb})$, if the preimage of lb is not entailed by S_Q (i.e., $\mathsf{NotFound} \leftarrow \mathsf{Entail}(S_Q, \mathsf{lb})$), the simulated answer is $\perp$ even if lb is a valid label. In the real IRHWE oracle, the answer may not be $\perp$. Another difference is that the label which is answered to a new e-type or $eval$-type query may appear in the Q-A lists, $\bar{Q}_{\mathsf{dec}} \cup Q$, with negligible probability in the real IRHWE oracle. However, in our simulation, the probability is 0 since we will sample labels from the space excluding all labels appearing in $\bar{Q}_{\mathsf{dec}} \cup Q$.

We claim that the above two cases, where the answers from the simulated oracle and the real oracle differ, will happen with only small probability. We will prove this claim in Section 4.3.

4.3 Proof for Separating IPFE from IRHWE

In this subsection, we show that $\mathsf{Brk}^{\mathcal{O}}$ correctly answers the queries performed in the IPFE decryption with the challenge functional decryption key. We will analyze the decryption of a single ciphertext and then use union bound to complete the proof. Fix a single ciphertext $c \in \{c_j : j \in [t_{\mathrm{c}}]\}$. Recall that $\mathbf{v}_1, \ldots, \mathbf{v}_t$ and $\mathbf{v}$ are the corrupted vectors and the challenge vector chosen by $\mathsf{Brk}^{\mathcal{O}}$. Let Q_{kgen} denote the list of the Q-A pairs appearing in $\mathsf{sk}[\mathbf{v}] \leftarrow \mathsf{KGen}^{\mathcal{O}}(\mathsf{msk}, \mathbf{v})$, Q_{enc} denote the list of Q-A pairs appearing in the encryption of challenge $c \leftarrow \mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$ for the plaintext $\mathbf{m}$. Recall that Q^i_{dec} denotes the Q-A pairs appearing in $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}_i], c)$ for each $i \in [t]$ and let $\bar{Q}_{\mathsf{dec}} = \bigcup_{i \in [t]} Q^i_{\mathsf{dec}}$. We have the following lemma to show that the probability of successful decryption using our simulated oracle $\mathsf{Sim}_{\mathrm{IRHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}}]$ is not far from that using $\mathcal{O}$.

Lemma 4.11 (Simulated decryption is correct)**.** *There exists a constant τ and a negligible function* $\mathrm{negl}(\cdot)$ *s.t.*

$$\left| \Pr[\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c) = \langle \mathbf{v}, \mathbf{m} \rangle] - \Pr[\mathsf{Dec}^{\mathsf{Sim}_{\mathrm{IRHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}}]}(\mathsf{sk}[\mathbf{v}], c) = \langle \mathbf{v}, \mathbf{m} \rangle] \right| \leq \lambda^{-\tau} + \mathrm{negl}(\lambda).$$

The above lemma is almost sufficient to prove Theorem 4.1, which claims that Brk wins the security game, Definition 3.9. To see why, consider that the challenger sends either the inner product $m'_j = \langle \mathbf{v}, \mathbf{m}_j \rangle$ or a uniform and independently random m'_j. If $m'_j = \langle \mathbf{v}, \mathbf{m}_j \rangle$, then the attacker gets

$\mathsf{Dec}^{\mathsf{Sim}_{\mathrm{IRHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}}]}(\mathsf{sk}[\mathbf{v}], c_j) = m'_j$ for all $j \in [t_c]$ with high probability. Otherwise, m'_j is independently random, then because t_c is sufficiently large, with high probability, for all $(\mathsf{sk}'[\mathbf{v}], Q)$, there exists j such that $\mathsf{Dec}^{\mathsf{Sim}_{\mathrm{IRHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q]}(\mathsf{sk}'[\mathbf{v}], c_j) \neq m'_j$. See Lemma 4.16 and Lemma 4.17 for the detailed probability.

Proof sketch of Lemma 4.11. Before delving into the detailed proof, we outline the high-level idea. For readability, the random function H, a subroutine in the IRHWE oracle (Definition 3.14), is assumed to never collide during the execution, i.e., for different inputs, the random oracle H always outputs differently. Since the number of queries to H is bounded by a polynomial in λ, the probability of a collision is negligible; we will remove the assumption and formally analyze the probability in Section 4.5. We begin with the following definition. For an oracle $\mathcal{O}$, a query q to $\mathcal{O}$, and a list of Q-A pairs Q of $\mathcal{O}$, we say that the answer to query q is *provided by* Q if one of the following conditions are satisfied:

1. There exists an answer ans such that $(q, \mathsf{ans}) \in Q$.

2. If q is a d-query of the form $d_{\mathsf{V}}(w, \mathsf{lb} = (a, y))$, then $\mathsf{Entail}(Q, \mathsf{lb}) \notin \{\mathsf{NotFound}, \perp\}$ and for every query q' asked by $\mathsf{V}^{\mathcal{O}}(a, w)$, the answer to query q' is provided by Q.

To consistently answer a query performed in $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$, the "essential" information about $\mathcal{O}$ is included in $Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}}$. Since Brk tries every possible Q_{kgen}, the main challenge arises from the hidden information in Q_{enc}, where some label-plaintext and labels-label pairs are associated in some e-type and $eval$-type Q-A pairs. To ensure the consistency of the simulated Q-A pairs, Brk needs to learn some Q-A pairs provided by Q_{enc}. Notice that Brk can only obtain the Q-A pairs in Q_{enc} by learning from the IPFE decryptions with the colluded functional keys. If a Q-A pair in Q_{enc} is correlated to $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$ but was not learned, Brk would fail to simulate some answers, which would then yield an incorrect inner product of $\mathbf{m}$ and $\mathbf{v}$.

There are three cases when such a failure can happen. In the following, we suppose that for some k, the answer to the kth query is the first one that Brk fails to simulate (while the previous $k-1$ answers are properly simulated).

Case 1: *The kth answer is provided by Q_{enc}.* A failure happens when the kth answer is provided only by Q_{enc} but not by $\bar{Q}_{\mathsf{dec}}$ (recall that Brk learned $\bar{Q}_{\mathsf{dec}}$ in the learning phase). Without Q_{enc}, Brk can guess correctly with only negligible probability. However, we claim that Brk can learn a large proportion of the Q-A pairs in Q_{enc} which appear in the IPFE decryption with the challenge functional decryption key in the learning Phase, and thus this case will not happen with high probability. To prove this claim, we define the event Bad to capture such a case: when the answer to be simulated is provided in $Q_{\mathsf{enc}} \setminus \bar{Q}_{\mathsf{dec}}$ and show Bad is unlikely to happen with high probability.

Case 2: *The kth answer is provided by $Q_{\mathsf{enc}} \cup Q_{\mathsf{kgen}}$.* A subtle issue remains even conditioning on $\neg\mathsf{Bad}$: the kth answer could be provided by the concatenation of $Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}}$ but not by the single Q_{kgen}. Because the answer depends on Q_{kgen}, it is unlikely to be provided by $\bar{Q}_{\mathsf{dec}}$. Fortunately, the algorithms $\mathsf{IPFE.KGen}$ and $\mathsf{IPFE.Enc}$ do not share any information about the oracle—indeed, they are independent except for sharing the master secret key, which is assumed to be independent of the oracle without loss of generality. This means that the concatenation of $Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}}$ could provide the answer with very low probability. We formally define the event Hit when the concatenation accidentally provides extra answers. Conditioned on $\neg\mathsf{Hit}$, the event Bad can capture all undesirable cases in which the answer is fully provided. Notice that both events Bad and Hit are defined on the (real) execution of IPFE over the real oracle $\mathcal{O}$. Also, the simulator is constructed under the $\neg\mathsf{Hit}$ assumption, as discussed in Remark 4.10.

Case 3: *The kth answer is not fully provided by Q_{enc} but depends partially on Q_{enc}.* In this case, Q_{enc} imposes some restrictions on a label because for the same instance, the label space is a permutation and the preimage of each label is unique. However, Brk does not have access to Q_{enc}, and it might reuse a label and cause an inconsistency. We define this failure event as Col_k and prove it occurs with negligible probability. Notice that Col_k is defined only in the simulation, which differs from Bad and Hit.

In the following proof, we formally define these undesirable events and show that each occurs with acceptably low probability. Conditioning on none of these events occurring, the simulated oracle behaves identically to the real oracle. This implies that Brk can perform decryption correctly with the same probability as when the decryption algorithm interacts with the real oracle.

Proof of Lemma 4.11. We begin by defining the undesirable events, Bad, Hit, and Col_k. The proof will utilize a sequence of hybrid experiments, where the sequence gradually replaces the kth query from the real IRHWE oracle $\mathcal{O}$ to the simulator. The hybrid argument shows that IPFE Dec achieves nearly the same correctness using the simulator $\mathsf{Sim}_{\mathrm{IRHWE}}$ as if the real IRHWE oracle $\mathcal{O}$ were used. We remark that Bad and Hit are defined with respect to the real IRHWE oracle $\mathcal{O}$, but Col_k is defined on a hybrid experiment, which uses both the real IRHWE oracle $\mathcal{O}$ and the simulator. After the definition of each event, we will claim that the event happens with a small or negligible probability, and then we will prove Lemma 4.11 using the claims. The proofs of some claims will be deferred.

The event Bad captures the case when there exists some Q-A pair in Q_{enc} which is essential but is not learned in $\bar{Q}_{\mathsf{dec}}$.

Definition 4.12 (Bad event w.r.t. $\mathcal{O}$)**.** *The event* Bad *is defined as the union of the following cases.*

1. *(Bad query.) There exists a Q-A pair in $Q_{\mathsf{dec}} \cap Q_{\mathsf{enc}}$ but not in $\bar{Q}_{\mathsf{dec}}$ (i.e., $\bigcup_{i\in[t]} Q^i_{\mathsf{dec}}$).*
2. *(Bad label.) There exists a label lb, a plaintext $m \neq \bot$, and a witness w such that lb is an answer in Q_{enc}, and m is the pre-image of lb w.r.t. $\mathcal{O}$, and the Q-A pair $((d_{\mathsf{V}}, w, \mathsf{lb}), m) \in Q_{\mathsf{dec}}$, but for any w', $((d_{\mathsf{V}}, w', \mathsf{lb}), m) \notin \bar{Q}_{\mathsf{dec}}$.*

We claim that Bad happens with a small probability below; the proof is provided later in Section 4.4.

Claim 4.12.1. *For a $\mathbb{Z}_p^n$-SK-IPFE where $n := n(\lambda)$ is a polynomial, $n \geq 3$, p is a prime number, and p^{-n} is negligible, there exists a constant τ s.t.* $\Pr[\mathsf{Bad}] \leq \lambda^{-\tau}$.

The event Hit captures the case when a string happens to be a *valid* label or not fresh; that is, the string is never an output of oracle queries but accidentally valid or has appeared before.

Definition 4.13 (Hit event w.r.t. $\mathcal{O}$)**.** *The event* Hit *is defined as the union of two cases. The first case happens when the oracle is queried with a* valid label *as an input, but the label is never an output of the oracle in the preceding queries. The second case happens when the oracle answered a query with a (valid or invalid) label such that the query is fresh, but the label existed identically in a preceding query. The "preceding" queries may be performed by the same algorithm or by another algorithm in the experiment.* Hit *is defined to be the union of all the following events, where the experiment is performed with the real IRHWE oracle $\mathcal{O}$ and the real functional key, as denoted by Q_{dec}.*

- *(Hit a valid input, same algorithm.) For some $Q \in \{Q_{\mathsf{kgen}}, Q_{\mathsf{enc}}\}$, there exist a query $q \in Q$ and* lb *such that* lb *is a* valid input label *of q, but for all query $q' \in \mathsf{Query}(Q)$ that is chronologically preceding q,* lb *is not an output label of q'.*
- *(Hit a valid input, across algorithms.) There exist a query $q \in Q_{\mathsf{dec}}$ and* lb *such that* lb *is a* valid input label *of q, but for all query $q' \in Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}}$ and for all $q' \in \mathsf{Query}(Q_{\mathsf{dec}})$ that is chronologically preceding q,* lb *is not an output label of q'.*
- *(Hit another label, across algorithms.) There exist a query $q \in Q_{\mathsf{dec}}$ and a label* lb *such that (1) the answer to q is* lb*, and (2) for all $q' \in \mathsf{Query}(Q)$, where $Q := Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}} \cup \bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{dec}}^{[-q]}$ and $Q_{\mathsf{dec}}^{[-q]}$ denotes the subset of Q_{dec} that consists of all queries that chronologically precede q, it holds that $q \neq q'$, but (3)* lb *is a label in a Q-A pair in Q.*

We claim that Hit happens with negligible probability; the proof is provided later in Section 4.5.

Claim 4.13.1. *For any IRHWE oracle $\mathcal{O}$ with parameter λ and any oracle-aided IPFE, the probability of* Hit *is bounded by $\Pr[\mathsf{Hit}] = 2^{-\Omega(\lambda)}$.*

We analyze the correctness of the simulated decryption by a sequence of hybrid experiments, Hyb_k, defined below.

Definition 4.14 (Hybrid experiment, Hyb_k)**.** *Let ℓ be the number of queries in each (canonical form) algorithm. For each $k \in \{0, \ldots, \ell\}$, the hybrid experiment Hyb_k answers the first k oracle queries using the real IRHWE oracle $\mathcal{O}$ and simulates the remaining $\ell - k$ answers. That is,*

$$\mathsf{Hyb}_k := \begin{cases} 1 & \text{if } \mathsf{Dec}^{\mathcal{O}_k}(\mathsf{sk}[\mathbf{v}], c) = \langle \mathbf{v}, \mathbf{m} \rangle \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathcal{O}_k$ is the hybrid oracle defined below.

- *The first k queries of $\mathsf{Dec}^{\mathcal{O}_k}(\mathsf{sk}[\mathbf{v}], c)$ are answered by $\mathcal{O}$; Denote the first k Q-A pairs as $Q_{\mathsf{dec},k}$.*
- *The remaining $\ell - k$ queries are answered by the simulated IRHWE oracle $\mathcal{O}'_k := \mathsf{Sim}_{\mathrm{IRHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{dec},k}]$.*

Notice that Hyb_0 indicates whether $\mathsf{Dec}^{\mathsf{Sim}_{\mathrm{IRHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}}]}(\mathsf{sk}[\mathbf{v}], c)$ is correct, and that Hyb_ℓ achieves the correctness of IPFE, $1 - \epsilon$ in probability, as it uses the real oracle $\mathcal{O}$. Now we define the event Col_k with respect to the simulated oracle on the hybrid experiment Hyb_{k-1}, where the simulated oracle outputs a *colliding* label.

Definition 4.15 (Col_k event for Hyb_{k-1})**.** *For the $(k-1)$th hybrid experiment $\mathsf{Dec}^{\mathcal{O}_{k-1}}(\mathsf{sk}[\mathbf{v}], c)$, we define the event Col_k as the following case.*

- *The kth simulated Q-A pair is (q, lb) for some e-type or $eval$-type query q, and lb appears in Q_{enc}, but (q, lb) does not appear in $\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{dec},k-1}$.*

By the definition, the probability of the event Col_k depends on the list $\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{dec},k-1}$ from the real oracle. Consequently, it depends on the events Bad and Hit, both defined with respect to the real oracle. Fortunately, we only need to bound the probability of Col_k occurring conditioned on that neither Bad nor Hit occurs. The following claim shows the required bound.

Claim 4.15.1. $\Pr[\mathsf{Col}_k | \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = 2^{-\Omega(\lambda)}$.

Proof. Let $\hat{l}$ be the upper bound of the number of inputs in any $eval$-type query. The event Col_k occurs only when the kth query is a new e-query or $eval$-query whose answer needs to be sampled by the simulator. In the simulation, the label is sampled uniformly from $\{0,1\}^{3\lambda}$, excluding all y values appearing in the labels associated with the target instance in $\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{dec},k-1}$ (Step 2 and 3b of Algorithm 7). For any fixed instance, the number of distinct labels in $\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{dec},k-1}$ is at most $((t+1)\ell + k - 1) \cdot (\hat{l} + 1)$. Meanwhile, for the same instance, the number of distinct labels in Q_{enc} is at most $\ell \cdot (\hat{l} + 1)$. Thus, the probability that the sampled y appears in Q_{enc} is at most $\ell \cdot (\hat{l}+1)/(2^{3\lambda} - ((t+1)\ell + k - 1)(\hat{l}+1)) = 2^{-\Omega(\lambda)}$. □

Conditioning on the good event that Bad, Hit, and Col_k never happen, we claim that Hyb_k and Hyb_{k-1} will be identical. The claim is formally stated below, and its proof is deferred to Section 4.6.

Claim 4.15.2. *For all $k \in \{1, \ldots, \ell\}$, it holds that*

$$\Pr[\mathsf{Hyb}_k = 1 \mid \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = \Pr[\mathsf{Hyb}_{k-1} = 1 \mid \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}].$$

Next, we bound the difference between the first and last hybrid experiments using the above claims. We begin by removing the condition $\neg\mathsf{Col}_k$ from the above claim so that the difference between hybrids is not conditioned on k. Let $T_1 := \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}$, and let $T_2 := \neg\mathsf{Col}_k$. We have that

$$\begin{aligned}
&\Pr[\mathsf{Hyb}_k = 1 \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] \\
=&\Pr[\mathsf{Hyb}_k = 1 \mid T_1 \wedge T_2] \cdot \Pr[T_2 \mid T_1] + \Pr[\mathsf{Hyb}_k = 1 \mid T_1 \wedge \neg T_2] \cdot \Pr[\neg T_2 \mid T_1] \\
\le&\Pr[\mathsf{Hyb}_k = 1 \mid T_1 \wedge T_2] + \Pr[\neg T_2 \mid T_1].
\end{aligned}$$

Similarly, we have that

$$\begin{aligned}
&\Pr[\mathsf{Hyb}_{k-1} = 1 \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] \\
=&\Pr[\mathsf{Hyb}_{k-1} = 1 \mid T_1 \wedge T_2] \cdot \Pr[T_2 \mid T_1] + \Pr[\mathsf{Hyb}_{k-1} = 1 \mid T_1 \wedge \neg T_2] \cdot \Pr[\neg T_2 \mid T_1] \\
\ge&\Pr[\mathsf{Hyb}_{k-1} = 1 \mid T_1 \wedge T_2] \cdot (1 - \Pr[\neg T_2 \mid T_1]).
\end{aligned}$$

Thus, using Claim 4.15.2 and then Claim 4.15.1, the difference can be bounded by

$$\begin{aligned}
&\Pr[\mathsf{Hyb}_k = 1 \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] - \Pr[\mathsf{Hyb}_{k-1} = 1 \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] \\
\le&\Pr[\mathsf{Hyb}_k = 1 \mid T_1 \wedge T_2] + \Pr[\neg T_2 \mid T_1] - \Pr[\mathsf{Hyb}_{k-1} = 1 \mid T_1 \wedge T_2] \cdot (1 - \Pr[\neg T_2 \mid T_1]) \\
\le&2\Pr[\mathsf{Col}_k | \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = 2^{-\Omega(\lambda)}. \qquad (1)
\end{aligned}$$

We now show the difference between the first and last hybrid experiments.

$$
\begin{aligned}
&\Pr[\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c) = \langle \mathbf{v}, \mathbf{m} \rangle] - \Pr[\mathsf{Dec}^{\mathsf{Sim}_{\mathrm{IRHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}}]}(\mathsf{sk}[\mathbf{v}], c) = \langle \mathbf{v}, \mathbf{m} \rangle] \\
=& \Pr[\mathsf{Hyb}_\ell = 1] - \Pr[\mathsf{Hyb}_0 = 1] \\
\le& \Pr[\mathsf{Hyb}_\ell = 1 \mid T_1] \Pr[T_1] + \Pr[\neg T_1] - \Pr[\mathsf{Hyb}_0 = 1 \mid T_1] \Pr[T_1] \\
\le& \Pr[\mathsf{Hyb}_\ell = 1 \mid T_1] - \Pr[\mathsf{Hyb}_0 = 1 \mid T_1] + \Pr[\neg T_1] \\
\le& \ell 2^{-\Omega(\lambda)} + \lambda^{-\tau} + 2^{-\Omega(\lambda)} = \lambda^{-\tau} + 2^{-\Omega(\lambda)}.
\end{aligned}
$$

The last inequality follows from Eq. (1) and from $\Pr[\mathsf{Bad} \vee \mathsf{Hit}] \le \Pr[\mathsf{Bad}] + \Pr[\mathsf{Hit}] \le \lambda^{-\tau} + 2^{-\Omega(\lambda)}$ by Claim 4.12.1 and Claim 4.13.1. This concludes the proof of Lemma 4.11. □

We next analyze the success probability of Brk in the security game. Recall that Brk aims to guess the challenge bit b. For each case of b, we show that the output of Brk equals to b with high probability, using union bound and Lemma 4.11.

Lemma 4.16. *Let b' be the output of $\mathsf{Brk}^{\mathcal{O}}$, for a $\mathbb{Z}_p^n$-SK-IPFE $\mathcal{E}^{\mathcal{O}}$ with correctness $1 - 1/2^\lambda$, where λ is the security parameter, we have*

$$\Pr[b' = 0 \mid b = 0] \ge 1 - O(\frac{1}{\lambda}).$$

Proof. When $Q = Q_{\mathsf{kgen}}$ and $\mathsf{sk}'[\mathbf{v}] = \mathsf{sk}[\mathbf{v}]$, by Lemma 4.11, there is a constant τ such that

$$\Pr[\mathsf{Dec}^{\mathsf{Sim}}(\mathsf{sk}'[\mathbf{v}], c_j) = \langle \mathbf{v}, \mathbf{m} \rangle] \ge 1 - 1/2^\lambda - \lambda^{-\tau} - 2^{-\Omega(\lambda)}$$

for each $c_j, j \in [t_c]$, where $\mathsf{Sim} = \mathsf{Sim}_{\mathrm{IRHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q]$. Since $\mathsf{Brk}^{\mathcal{O}}$ goes through all choices of Q and $\mathsf{sk}'[\mathbf{v}]$, the correct Q-A list Q_{kgen} and the challenge functional decryption key $\mathsf{sk}[\mathbf{v}]$ must be enumerated. By union bound, we have

$$\Pr[b' = 0 \mid b = 0] \ge 1 - t_c \left(1/2^\lambda + \lambda^{-\tau} + 2^{-\Omega(\lambda)}\right).$$

Because the parameter $\lambda^\tau \ge 4 t_c \lambda$, we get $\Pr[b' = 0 \mid b = 0] \ge 1 - O(1/\lambda)$ for sufficiently large λ. □

Lemma 4.17. *Let b' be the output of $\mathsf{Brk}^{\mathcal{O}}$, for a $\mathbb{Z}_p^n$-SK-IPFE $\mathcal{E}^{\mathcal{O}}$, we have $\Pr[b' = 1 \mid b = 1] \ge 1 - \frac{1}{2^\lambda}$.*

Proof. When $b = 1$ in the security game (Definition 3.9), each m'_j is uniformly chosen from $\mathbb{Z}_p$ for every $j \in [t_c]$. For any $\mathsf{sk}'[\mathbf{v}] \in \{0,1\}^\mu$ and $Q \in \{0,1\}^\rho$, the probability of $\mathsf{Dec}^{\mathsf{Sim}}(\mathsf{sk}'[\mathbf{v}], c_j) = m'_j$ is $1/p$. Furthermore, because every m'_j is independently chosen, the probability that $\mathsf{Dec}^{\mathsf{Sim}}(\mathsf{sk}'[\mathbf{v}], c_j) = m'_j$ holds for every $j \in [t_c]$ is $1/p^{t_c}$. Recall that we chose the parameter $t_c \ge \lambda + \mu + \rho$. By union bound over all $2^{\mu+\rho}$ choices of $\mathsf{sk}'[\mathbf{v}]$ and Q, it holds that

$$\Pr[b' = 1 \mid b = 1] \ge 1 - \frac{2^{\mu+\rho}}{p^{t_c}} \ge 1 - \frac{1}{2^\lambda}.$$

□

Putting together Lemma 4.16 and Lemma 4.17, we have Theorem 4.1.

4.4 Proof of Claim 4.12.1

Claim 4.12.1. *For a* $\mathbb{Z}_p^n$-SK-IPFE *where* $n := n(\lambda)$ *is a polynomial,* $n \geq 3$*,* p *is a prime number, and* p^{-n} *is negligible, there exists a constant* τ *s.t.* $\Pr[\mathsf{Bad}] \leq \lambda^{-\tau}$.

We recall the Bad event in Definition 4.12 and Claim 4.12.1 as the above. Let $2^{[\ell]}$ denote the set of subsets of $[\ell]$. Since at most ℓ queries are made in the canonical algorithm $\mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$, we can use $[\ell]$ to index these Q-A pairs. We define the functions F_1, F_2 below for any fixed oracle $\mathcal{O}$, master secret key msk, vector $\mathbf{m}$, and $c \leftarrow \mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$ for some fixed encryption randomness, where as $\mathsf{sk}[\mathbf{u}] \leftarrow \mathsf{KGen}^{\mathcal{O}}(\mathsf{msk}, \mathbf{u})$ is a function of a variable $\mathbf{u}$.

- (Queries.) $F_1(\mathbf{u})$: $\mathbb{Z}_p^n \to 2^{[\ell]}$ maps a vector $\mathbf{u} \in \mathbb{Z}_p^n$ to a subset $Z \subseteq [\ell]$ such that $j \in Z$ if and only if the jth Q-A pair of $\mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$ is also a Q-A pair in $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{u}], c)$.
- (Inverted labels.) $F_2(\mathbf{u})$: $\mathbb{Z}_p^n \to 2^{[\ell]}$ maps a vector $\mathbf{u} \in \mathbb{Z}_p^n$ to a subset $Z \subseteq [\ell]$ such that $j \in Z$ if and only if the jth Q-A pair of $\mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$ is $(\cdot, \mathsf{lb})$ for some label lb and there is a Q-A pair $((d_{\mathsf{V}}, w, \mathsf{lb}), m)$ in $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{u}], c)$ for some w, m.

By HLOW Lemma (Lemma 2.3), given any constant 2τ, there exists $t = \mathrm{poly}(\lambda)$ such that

$$\Pr\left[F_j(\mathbf{v}) \not\subseteq \bigcup_{i\in[t]} F_j(\mathbf{v}_i)\right] \leq \lambda^{-2\tau},$$

for each $j = 1, 2$, where $\mathbf{v}$ is the challenge key vector, and $\mathbf{v}_1, ..., \mathbf{v}_t$ are the colluded key vectors.

The "bad query" case of Bad implies that $F_1(\mathbf{v}) \not\subseteq \bigcup_{i\in[t]} F_1(\mathbf{v}_i)$ by the definition of F_1. Similarly, by the definition of F_2, the "bad label" case of Bad also implies that $F_2(\mathbf{v}) \not\subseteq \bigcup_{i\in[t]} F_2(\mathbf{v}_i)$. Therefore, we have

$$\Pr[\mathsf{Bad}] \leq \Pr[F_1(\mathbf{v}) \not\subseteq \bigcup_{i\in[t]} F_1(\mathbf{v}_i)] + \Pr[F_2(\mathbf{v}) \not\subseteq \bigcup_{i\in[t]} F_2(\mathbf{v}_i)] \leq 2\lambda^{-2\tau} \leq \lambda^{-\tau}.$$

4.5 Proof of Claim 4.13.1

Claim 4.13.1. *For any IRHWE oracle* $\mathcal{O}$ *with parameter* λ *and any oracle-aided IPFE, the probability of* Hit *is bounded by* $\Pr[\mathsf{Hit}] = 2^{-\Omega(\lambda)}$.

We recall the Hit event in Definition 4.13 and Claim 4.13.1 as the above. For any IPFE in the IRHWE oracle world, let $\hat{l} = \mathrm{poly}(\lambda)$ be the upper bound of the number of inputs in any *eval*-type query, where λ is the parameter of the oracle.

We will analyze the three cases of Hit separately below. By the security game of IPFE (Definition 3.9) and the construction of Brk, the master secret key msk, key vector $\mathbf{v}$, and message $\mathbf{m}$ are independent of IRHWE oracle $\mathcal{O}$. Thus, the IPFE scheme must obtain valid labels from the oracle but never from the inputs msk, $\mathbf{v}, \mathbf{m}$. With that, hitting the same label never happens except with exponentially small probability. Recall that ℓ denotes the number Q-A pairs in the canonical execution of IPFE algorithms and t denotes the number of colluded functional decryption keys.

Case 1, hit a valid input, same algorithm. Fix $Q \in \{Q_{\mathsf{kgen}}, Q_{\mathsf{enc}}\}$, and fix an instance a. There are $2^{3\lambda}$ distinct labels but only $2^{2\lambda}$ are valid. For a label appearing for the first time in a query, the probability that it is valid is no more than $2^{2\lambda}/(2^{3\lambda} - (\hat{l} + 1)\ell)$. This is because $\mathcal{O}$ is independent with $\mathsf{msk}, \mathbf{v}, \mathbf{m}$ which means that $\pi_a(\cdot)$ is a random permutation, and the preceding Q-A pairs of Q can rule out at most $(\hat{l}+1)\ell$ invalid labels. By union bound over at most $\hat{l} \cdot \ell$ labels, there exists such a valid label in Q_{kgen} (resp. Q_{enc}) happens with probability at most $\hat{l} \cdot \ell \cdot 2^{2\lambda}/(2^{3\lambda} - (\hat{l}+1)\ell)$.

Case 2, hit a valid input, across algorithms. Similarly, for a fixed instance a, there are $2^{2\lambda}$ valid labels and the whole label space is $2^{3\lambda}$. For an input label of a query, it happens with probability at most $2^{2\lambda}/(2^{3\lambda} - 3(\hat{l}+1)\ell)$ that the label is valid but it neither exists earlier in $Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}}$ nor has appeared as the answer to any preceding query in Q_{dec}. This is because $\pi_a(\cdot)$ is a random permutation for the same reason and the previous Q-A pairs can exclude at most $3(\hat{l}+1)\ell$ invalid labels. By union bound over all labels, the probability is at most $\hat{l} \cdot \ell \cdot 2^{2\lambda}/(2^{3\lambda} - 3(\hat{l}+1)\ell)$.

Case 3, hit another label, across algorithms. Fixing an e- or $eval$-query $q = (\cdot, \mathsf{lb}) \in Q_{\mathsf{dec}}$, the output label $\mathsf{lb} = (a, y)$ and thus the instance a are fixed. In the list $Q = \bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}} \cup Q_{\mathsf{dec}}^{[-q]}$, there are at most $(t+3) \cdot \ell$ number of distinct queries and thus $n_q := (t+3) \cdot \ell \cdot (\hat{l}+1)$ distinct labels. A label is either an input or an output of a query. By definition, each distinct query calls the random function H on a distinct input (Definition 3.14). If the outputs of all distinct calls to H are also distinct, then the query q gets a distinct $h \in \{0,1\}^\lambda$ from H, and then the answer consists of a distinct $y = \pi_a(\cdot \| h \| \cdot)$ by the permutation π_a, which implies that the output label is also distinct. By a union bound over at most n_q queries, the outputs of H are distinct except with probability at most $n_q^2/2^\lambda$, over the random space of H. Conditioning on the event that H does not collide, the permutation π_a samples a fresh random y from at least $2^{3\lambda} - n_q$ random strings because the independency between $\mathcal{O}$ and $\mathsf{msk}, \mathbf{v}, \mathbf{m}$. By union bound over at most n_q labels in Q, y differs from all labels in Q except with probability at most $n_q/(2^{3\lambda} - n_q)$. By union bound over all ℓ queries and the condition on H, the case happens with probability at most $\ell \cdot n_q/(2^{3\lambda} - n_q) + n_q^2/2^\lambda$.

By union bound and by $n_q = \mathrm{poly}(\lambda)$, we have that $\Pr[\mathsf{Hit}] \le \hat{l} \cdot \ell \cdot 2^{2\lambda+1}/(2^{3\lambda} - 3(\hat{l}+1)\ell) + \ell \cdot n_q/(2^{3\lambda} - n_q) + n_q^2/2^\lambda = 2^{-\Omega(\lambda)}$.

4.6 Proof of Claim 4.15.2

Claim 4.15.2. *For all $k \in \{1, \ldots, \ell\}$, it holds that*

$$\Pr[\mathsf{Hyb}_k = 1 \mid \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = \Pr[\mathsf{Hyb}_{k-1} = 1 \mid \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}].$$

The claim is restated above. Recall that by definition, the first $k-1$ Q-A pairs in the hybrids $\mathsf{Dec}^{\mathcal{O}_k}(\mathsf{sk}[\mathbf{v}], c)$ and $\mathsf{Dec}^{\mathcal{O}_{k-1}}(\mathsf{sk}[\mathbf{v}], c)$ are exactly the same and using the real IRHWE oracle $\mathcal{O}$, and thus the kth query appearing in both experiments is identical. Let $Q_{\mathsf{dec},k-1}$ denote the first $k-1$ Q-A pairs w.r.t. the real IRHWE oracle $\mathcal{O}$. Next, we argue that the answers to the kth query from $\mathcal{O}$ and $\mathcal{O}'_{k-1} := \mathsf{Sim}_{\mathrm{IRHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{dec},k-1}]$ follow the same distribution when Hit, Bad, and Col_k does not occur.

Recall the Col_k event (Definition 4.15) formalizes the case when the simulator accidentally sampled a label in Q_{enc} but not elsewhere. Also, when Hit does not happen, we have the following:

I. In Q_{kgen} or Q_{enc}, for each valid input label of any query, it must be an answer to an earlier query in the list. This is followed by the case "hit a valid input, same algorithm" of Hit.

II. In Q_{dec}, if an input label lb of a d-query q is successfully inverted (so that the label must be valid), then its preimage (i.e., underlying plaintext) is entailed by $Q_{\mathsf{kgen}} \cup \bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{dec}}^{[-q]}$.

We explain this implication below. Conditioned on the case "hit a valid input, across algorithms" of Hit not occurring, there exists a query $q' \in Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}} \cup Q_{\mathsf{dec}}^{[-q]}$ such that q' outputs label lb. If q' is a e-query in Q_{kgen} or $Q_{\mathsf{dec}}^{[-q]}$, it's preimage is directly entailed by Q_{kgen} or $Q_{\mathsf{dec}}^{[-q]}$ respectively. If q' is a e-query in Q_{enc}, then by the case "bad label" of Bad not occurring, there must exists a successful d-query inverting lb in $\bar{Q}_{\mathsf{dec}}$. Now we consider that q' is an $eval$-query. If q' is in Q_{kgen}, the preimage is entailed by Q_{kgen}, since the case "hit a valid input, same algorithm" of Hit not occurring. If q' is in Q_{enc}, because the case "bad label" of Bad not occurring, the preimage is entailed by $\bar{Q}_{\mathsf{dec}}$. Finally, if q' is in $Q_{\mathsf{dec}}^{[-q]}$, the preimage is entailed by $Q_{\mathsf{dec}}^{[-q]}$ because of the second item of Lemma 4.7. To conclude, conditioned on $\neg\mathsf{Bad} \wedge \mathsf{Hit}$, the preimage is entailed by $Q_{\mathsf{kgen}} \cup \bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{dec}}^{[-q]}$ as implied by Lemma 4.7.

We proceed with a case analysis to demonstrate the distribution difference between the kth answers from the real IRHWE oracle $\mathcal{O}$ and the simulation $\mathcal{O}'_{k-1}$.

Let ans denote the kth answer from $\mathcal{O}$, ans' denote the kth answer from $\mathcal{O}'_{k-1}$, and $Q := Q_{\mathsf{kgen}} \cup \bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{dec},k-1}$ which is the list of all Q-A pairs to/from the real oracle known to the simulator.

When the kth query also exists in $\mathsf{Query}(Q)$, both the real oracle and the simulator will answer the same string (recall that $\mathsf{Query}(Q)$ is the set of all queries appearing in Q). For the cases where the kth query does not exist in $\mathsf{Query}(Q)$, we categorize them according to the query type and give the analysis below, where the (distributions of) answers of the real oracle and the simulator are compared.

1. When kth query is an e-query of the form (e, a, m):

   (a) If $((e, a, m), \mathsf{lb}) \in Q_{\mathsf{enc}}$ for some $\mathsf{lb} = (a, y)$:
   Real oracle answers lb, and thus $\Pr[\mathsf{ans} = \mathsf{lb}] = 1$.
   When Conditioned on $\neg\mathsf{Bad}$, the "bad query" case does not happen, $((e, a, m), \mathsf{lb}) \in Q$, and thus the simulator answers lb, $\Pr[\mathsf{ans}' = \mathsf{lb} | \neg\mathsf{Bad}] = 1$.

   (b) Else if $((e, a, m), \mathsf{lb}) \notin Q_{\mathsf{enc}}$ for any lb:
   Real oracle is conditioned on $\neg\mathsf{Hit}$, the "hit another label, across algorithms" case does not happen, and thus lb is not in $Q \cup Q_{\mathsf{enc}}$; thus, the kth answer is uniformly distributed in the set $\mathcal{Y} = \{0,1\}^{3\lambda} \setminus \mathsf{lb}_a(Q \cup Q_{\mathsf{enc}})$.
   Conditioned on $\neg\mathsf{Col}_k$, the kth answer from the simulator is also uniformly distributed in $\mathcal{Y} = \{0,1\}^{3\lambda} \setminus \mathsf{lb}_a(Q \cup Q_{\mathsf{enc}})$ because $\neg\mathsf{Col}_k$ excludes the labels in Q_{enc}.

   Therefore, in both cases, the distributions of ans and ans' are identical when conditioned on $\neg\mathsf{Hit} \wedge \neg\mathsf{Col}_k$.

2. When kth query is an $eval$-query of the form $(eval, \mathsf{lb}_1, ..., \mathsf{lb}_l)$, where $\mathsf{lb}_i = (a, y_i \neq \perp)$ for each $i \in [l]$ (the cases are almost identical to those of e-query):

   (a) If $((eval, \mathsf{lb}_1, ..., \mathsf{lb}_l), \mathsf{lb}) \in Q_{\mathsf{enc}}$ for some $\mathsf{lb} = (a, y)$:
   Real oracle answers lb, and thus $\Pr[\mathsf{ans} = \mathsf{lb}] = 1$.
   Conditioned on $\neg\mathsf{Bad}$, the "bad query" case does not happen, and thus $((eval, \mathsf{lb}_1, ..., \mathsf{lb}_l), \mathsf{lb}) \in Q$. Simulator answers the same lb, thus $\Pr[\mathsf{ans}' = \mathsf{lb} | \neg\mathsf{Bad}] = 1$.

(b) Else if $((eval, \mathsf{lb}_1, ..., \mathsf{lb}_l), \mathsf{lb}) \notin Q_{\mathsf{enc}}$ for any lb:
Real oracle is conditioned on $\neg\mathsf{Hit}$, the "hit another label, across algorithms" case does not happen, and thus lb is not in $Q \cup Q_{\mathsf{enc}}$; the kth answer from the real oracle is uniformly distributed in the set $\mathcal{Y} = \{0,1\}^{3\lambda} \setminus \mathsf{lb}_a(Q \cup Q_{\mathsf{enc}})$.
Conditioned on $\neg\mathsf{Col}_k$, the kth answer from the simulator is also uniformly distributed in $\mathcal{Y} = \{0,1\}^{3\lambda} \setminus \mathsf{lb}_a(Q \cup Q_{\mathsf{enc}})$.

Therefore, in both cases, the distributions of ans and ans' are identical when conditioned on $\neg\mathsf{Hit} \wedge \neg\mathsf{Col}_k$.

A remaining case is that the input labels are mal-formed, i.e., there exist $i_1 \neq i_2 \in [l]$ such that $\mathsf{lb}_{i_1} = (a_{i_1}, y_{i_1}), \mathsf{lb}_{i_2} = (a_{i_2}, y_{i_2})$, but $a_{i_1} \neq a_{i_2}$. The real oracle and simulator both abort with $\perp$, so that they are identical.

3. When the kth query is a d-query of the form $(d_{\mathsf{V}}, w, \mathsf{lb} = (a, y))$:

(a) If $m \leftarrow \mathsf{Entail}(Q, \mathsf{lb})$, where $m \in \{\mathsf{NotFound}, \perp\}$:
For real oracle, suppose for contradiction, the label lb is inverted successfully. By the implication Item II, the preimage is entailed by Q. However, the algorithm $\mathsf{Entail}(Q, \mathsf{lb})$ would had outputted the preimage of lb, contradicting $m \in \{\mathsf{NotFound}, \perp\}$. Therefore, the label lb is not inverted successfully, and the real oracle answers to the query with $\mathsf{ans} = \perp$.
The simulator answers $\perp$ by construction.

(b) If $m \leftarrow \mathsf{Entail}(Q, \mathsf{lb})$, where $m \notin \{\mathsf{NotFound}, \perp\}$:
Real oracle runs $\mathsf{V}^{\mathcal{O}}(a, w)$, then answers m if $1 = \mathsf{V}^{\mathcal{O}}(a, w)$ and $\perp$ otherwise.
The simulator evaluates $\mathsf{V}^{\mathcal{O}'_{k-1}}(a, w)$, where all queries are answered using $\mathcal{O}'_{k-1}$ and Q. If the verification is correct, then it will answer m, and $\perp$ otherwise. By the first item of Lemma 4.7, all Q-A pairs needed in $\mathsf{V}^{(\cdot)}(a, w)$ are already in Q. Since the Q-A pairs in Q are all identical to the real oracle, the answer ans' is identical to the real ans.

Therefore, in both cases, ans and ans' are identically distributed when conditioned on $\neg\mathsf{Bad} \wedge \neg\mathsf{Hit}$.

In conclusion, conditioned on none of the events $\mathsf{Bad}, \mathsf{Hit}, \mathsf{Col}_k$ happens, the distributions of the answers given by $\mathcal{O}$ and $\mathcal{O}'_{k-1}$ to the kth query are identical. Specifically, both oracles either 1) provide a previously determined answer, or 2) answer it by uniformly sampling from all possible answers. By construction, the distributions of the remaining $\ell - k$ Q-A pairs in $\mathsf{Dec}^{\mathcal{O}_{k-1}}(\mathsf{sk}[\mathbf{v}], c)$ and $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$ are identical. Therefore, we have $\Pr[\mathsf{Hyb}_k = 1 | \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = \Pr[\mathsf{Hyb}_{k-1} = 1 | \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}]$.

5 Separating SK-IPFE from two-layer IHWE

In this section, we prove that there is no SK-IPFE exists in the two-layer IHWE oracle world. Let $\mathcal{O}$ be a two-layer IHWE oracle, Definition 3.20. Let $\mathcal{E}^{\mathcal{O}} := (\mathsf{KGen}^{\mathcal{O}}, \mathsf{Enc}^{\mathcal{O}}, \mathsf{Dec}^{\mathcal{O}})$ be an oracle-aided IPFE for vectors in $\mathbb{Z}_p^n$. We show that there exists a polynomial-query attacker $\mathsf{Brk}^{\mathcal{O}}$ which breaks the security of the given IPFE scheme in Section 5.3. The main theorem for this section is stated as follows.

Theorem 5.1. *Let λ be the security parameter, and suppose $n := n(\lambda) \geq 3$ and p^{-n} is negligible, then there is no $\mathbb{Z}_p^n$-$\mathsf{SK\text{-}IPFE}$ (Definition 3.9) relative to the two-layer IHWE oracle.*

Putting together Theorem 5.1, Lemma 3.21, and Lemma 3.8, we separate IPFE from two-layer IHWE as the following corollary, which implies our main result, Theorem 1.2, given that two-layer predicate encryptions are implied by two-layer IHWE (Corollary 3.22).

Corollary 5.2. *Let $\lambda \in \mathbb{N}$ be the security parameter. For any dimension $n = n(\lambda) \geq 3$ and any prime $p = p(\lambda)$ such that p^{-n} is negligible in λ, there is no black-box construction of $\mathbb{Z}_p^n$-$\mathsf{SK\text{-}IPFE}$ from two-layer extractable IHWE.*

Algorithms in the two-layer IHWE oracle model, and abused notations. Because we consider oracle-aid IPFE schemes using a two-layer IHWE oracle, throughout this section, we suppose that the IPFE algorithms always invoke the IHWE oracle in the well-formed two-layer fashion. That is, an algorithm can query the suboracle $d_{\mathsf{V}^{\mathcal{O}'}}$ that can query the suboracle $d_{\mathsf{V}'}$, but $d_{\mathsf{V}'}$ and its V' cannot perform further oracle queries (see Definition 3.13 for the suboracles). Well-formedness can be syntactically verified by checking the algorithm's circuit. We assume that IPFE never queries $d_{\mathsf{V}'}$ suboracle directly; this is w.l.o.g. because any algorithm can instead query $d_{\mathsf{V}^{\mathcal{O}'}}$ and ignore $\mathcal{O}'$ in the extended (or two-layer) IHWE oracle world.

Our attacker queries both suboracles $d_{\mathsf{V}^{\mathcal{O}'}}$ and $d_{\mathsf{V}'}$ for readability. The two suboracles are often treated analogously, and we abuse notation and write d_{V} for both; when we use a corresponding oracle-aided $\mathsf{V}^{(\cdot)}$ in the context, we exclusively use $d_{\mathsf{V}^{\mathcal{O}'}}$. By the same reason, we abuse notation and write V for both $\mathsf{V}^{\mathcal{O}'}$ and V'.

Summary of challenges. Our attacker $\mathsf{Brk}^{\mathcal{O}}$ follows the same framework as in Section 4. Recall that the attacker consists of three phases, colluding, learning, and simulation (Section 3.6).

Similar to IRHWE, an IHWE query may trigger *indirect* queries (see Definition 3.20). Namely, a d-query is associated with an oracle-aided verification circuit V and can trigger indirect queries when running V. Because the indirect queries are made by the oracle internally, they are unseen when running the IPFE algorithms. Thus, our attacker $\mathsf{Brk}^{\mathcal{O}}$ cannot learn them in the learning phase. Previously in the IRHWE oracle world, we handled these indirect queries by canonicalization: the IPFE algorithms are compiled to make all indirect queries direct, as long as the algorithm "knows" the queries. For IHWE oracle, we also canonicalize the IPFE algorithms: in the execution of an algorithm, whenever there is a d-query q on a witness w and a label lb, as long as the algorithm knows the instance a of lb from other queries made in this algorithm, it runs $\mathsf{V}^{\mathcal{O}}(a, w)$ and directly makes the queries to $\mathcal{O}$. By HLOW Lemma, our attacker *can* learn all queries directly performed during the canonical IPFE decryption. However, *due to the instance-hiding property, there are several challenges to learn the indirect queries*. In the following, unless otherwise specified, $\mathsf{IPFE.KGen}$ denotes the IPFE key generation for the challenge key vector, $\mathsf{IPFE.Enc}$ denotes the IPFE encryption of the challenge ciphertext, and $\mathsf{IPFE.Dec}$ denotes the IPFE decryption using the functional decryption key and the ciphertext.

First challenge: *To learn the indirect queries incurred by a direct d-queries on a "retardant" label in canonicalized* $\mathsf{IPFE.Dec}$*, where the label was created in* $\mathsf{IPFE.Enc}$. After the canonicalization, for any label lb output by an e-query in $\mathsf{IPFE.Enc}$, our attacker can learn all indirect queries triggered by the d-query on lb if it is successfully inverted *at least once* in the challenge $\mathsf{IPFE.Dec}$; it is because the instance of lb must be revealed in the learning phase by HLOW Lemma, and then the learning phase canonicalizes the indirect queries. However, if lb is never inverted successfully in $\mathsf{IPFE.Dec}$, which is called a *retardant label*, then its instance is hidden, and the canonicalization has no way to expose the indirect queries of the associated verification circuits. Meanwhile, our attacker may learn the instance of lb in the learning phase. In such a case, the attacker needs to learn the unexposed queries to correctly simulate the verification for d-queries on lb. Example 5.3 illustrates such case.

Example 5.3. Suppose that $\mathsf{IPFE.Enc}$ somehow queries $e(a, x)$ and gets lb, and then $\mathsf{IPFE.Enc}$ outputs a ciphertext c. For one colluded functional key, $\mathsf{IPFE.Dec}$ on c queries $d_{\mathsf{V}}(w, \mathsf{lb})$ and gets (a, x). For another colluded functional key, $\mathsf{IPFE.Dec}$ on c queries $d_{\mathsf{V}}(w', \mathsf{lb})$ with a different $w' \neq w$ but gets $\perp$ because w' failed the verification. Then, using the challenge functional key, $\mathsf{IPFE.Dec}$ on c queries $d_{\mathsf{V}}(w'', \mathsf{lb})$ with yet another witness w'' also gets $\perp$. An attacker needs to simulate the failed verification $\mathsf{V}^{(\cdot)}(a, w'')$ and the incurred indirect queries.

To address it, our attacker makes additional queries as follows. For every label lb that appears in the learning phase, if the preimage of lb is also learned and thus the corresponding instance a is learned, then for *every* d-query of the form $d_{\mathsf{V}}(w, \mathsf{lb})$, the attacker additionally runs $\mathsf{V}^{\mathcal{O}}(a, w)$ and learns the queries. We will show that by doing so, our attacker learns all essential queries made by d-queries on lb.

Second challenge: *To learn the indirect queries incurred by direct d-queries on a retardant label in canonicalized* $\mathsf{IPFE.Dec}$, *where the label was created in* $\mathsf{IPFE.KGen}$. Similar to the first challenge, for a retardant label lb output by an e-query in $\mathsf{IPFE.KGen}$, our attacker may not learn the indirect queries incurred by a d-query on lb. Also similarly, that is because the canonicalization does not know the instance of lb if lb is never inverted successfully in $\mathsf{IPFE.Dec}$. However, the solution to the previous challenge does not help. Because lb may depend on the challenge key vector, it may never appear in the learning phase. Even worse, the instance a of lb can be designed to let all witnesses fail. See Example 5.4.

Example 5.4. $\mathsf{IPFE.KGen}$ queries $e(a, x)$ for some a, x and gets lb, where a is designed to trigger an indirect query q. Also, q is triggered in $\mathsf{IPFE.Enc}$ for any message vectors, and q depends on the master secret key. In $\mathsf{IPFE.Dec}$, $d_{\mathsf{V}}(w, \mathsf{lb})$ is queried, but the verification fails due to q.

To address this issue, observe that for a label lb created in $\mathsf{IPFE.KGen}$, the d-queries on lb during $\mathsf{IPFE.Dec}$ for many ciphertexts falls in two cases: 1) the d-queries are sometimes successful, i.e., $\mathcal{O}$ responded with non-$\perp$ answers, or 2) the d-queries are always answered by $\perp$. Our attacker needs to identify the two cases with high probability. To this end, we use the idea of compilation as follows. The attacker utilizes a sufficiently large set of ciphertexts (provided by challenger) for compilation. For each functional decryption key $\mathsf{sk}[\mathbf{u}]$, our attacker emulates the decryption of the compilation ciphertexts, and records all queries in the emulation; the queries are called heavy queries and denote as $Q^{\mathbf{u}}_{\text{com}}$.

The attacker will take $Q^{\mathbf{u}}_{\text{com}}$ as additional input to perform the decryption, which can be done by anyone who has $\mathsf{sk}[\mathbf{u}]$. Now, the canonicalized IPFE decryption utilizes $Q^{\mathbf{u}}_{\text{com}}$ and further exposes the indirect queries incurred by the d-queries. This is applied to both the challenge functional key and the colluded keys. That resolves case 1) because the canonicalization is able to expose the indirect queries; case 2) is resolved because our attacker can identify them and always output $\perp$ without simulating the verifications.

In Section 5.1, we describe our canonicalization algorithm in the IHWE oracle world and show canonicalization does not hurt the functionality of an algorithm. Then, in Section 5.2, we describe the compilation process. After that, we give our attacker Brk based on canonicalized IPFE followed by the proof of separating IPFE from two-layer IHWE oracle.

5.1 Canonical Execution of Oracle-Aided Algorithms in the IHWE oracle world

We describe how to compile any algorithm $A^{\mathcal{O}}$ relative to IHWE oracle into a canonical form $\bar{A}^{\mathcal{O}}$. Recall that the IHWE oracle evaluates oracle-aided circuits which may call the IHWE itself (see d_{V} in Definition 3.20). Similar to IRHWE, a d-query of the form $d_{\mathsf{V}}(w, \mathsf{lb})$ to IHWE oralce is answered in two steps: firstly, find the preimage of lb, which reveals the corresponding instance a, and secondly, run the verification $\mathsf{V}^{(\cdot)}(a, w)$ to determine the output. Our canonicalization makes the indirect queries in the verification direct, as long as it "knows" the preimage of lb. There are two ways that our canonicalization can know the preimage of a label. The first is to find the preimage from an earlier Q-A pair in the earlier canonical execution. The second is that the canonicalization $\bar{A}^{(\cdot)}[Q] := \mathsf{Canonicalize}^{(\cdot)}(A, Q)$ will take an additional list Q of Q-A pairs as input, and thus the preimage may be found in Q. Notice that the list Q is a major difference compared to that of Section 4, where Q is always empty. For readability, we write $\bar{A}^{(\cdot)}$ instead of $\bar{A}^{(\cdot)}[Q]$ when Q is empty.

Algorithm 8. $\mathsf{Canonicalize}^{\mathcal{O}}(A, Q)$

Input: An oracle-aided algorithm $A^{(\cdot)}$ that may implicitly include the input to A, and an additional list Q of Q-A pairs of the oracle; Q can be empty.
Oracle: An oracle $\mathcal{O}$ is given.
Procedure: Initialize a list $Q' := Q$. Then, repeat the following until no new element is added into Q'.
Emulate $A^{(\cdot)}$ and respond to every (direct/indirect) query q as follows:

```
1. If $q$ is $e$-type, query oracle $\mathsf{ans} \leftarrow \mathcal{O}(q)$, and add the Q-A pair $(q, \mathsf{ans})$ into $Q'$.
2. If $q$ is $d$-type of the form $d_{\mathsf{V}}(w, \mathsf{lb})$:
   (a) Query oracle $\mathsf{ans} \leftarrow \mathcal{O}(q)$ and add the Q-A pair $(q, \mathsf{ans})$ into $Q'$.
   (b) If $\mathsf{ans} \neq \perp$, parse $\mathsf{ans} = (a, m)$.
       i. Query the oracle $\mathsf{lb} \leftarrow e(a, m)$.
       ii. For every $((d_{\mathsf{V}}, w', \mathsf{lb}), x) \in Q'$, where $x$ is either $\perp$ or $(a, m)$, emulate
           $\mathsf{V}^{(\cdot)}(a, w')$ and respond to all queries using this procedure.
```

Finally, output the same as $\overline{A}^{(\cdot)}$.

5.2 Compilation

For each functional decryption key, The compilation process takes an additional list of "compilation ciphertexts" and then outputs an additional Q-A list. Looking forward, this compilation helps our attacker to simulate the answers of d-queries on labels created in $\mathsf{IPFE.KGen}$. Note that for every label lb created in $\mathsf{IPFE.KGen}$, the instance associated with this label lb is revealed to the attacker because the attacker exhausts all possible (direct and indirect) Q-A pairs in $\mathsf{IPFE.KGen}$.

However, when $\mathsf{IPFE.Dec}$ makes d-queries on lb, the queries triggered by their verifications are not directly performed by $\mathsf{IPFE.Dec}$ if all d-queries on the same label lb fail. There is no guarantee that these Q-A pairs can be learned through decryption using colluded functional keys because they can be specific to the challenge key vector.

Denoting queries as tree nodes. We observe that for all oracle-aided circuits A, the whole oracle querying topology of $A^{\mathcal{O}}$, including the indirect queries, are fully described by A and can be represented as a tree, where $\mathcal{O}$ is the extended IHWE, two-layer IHWE, or extended IRHWE oracles. This holds for all inputs as long as A is a circuit, so that the input length is fixed, potentially with fixed-length randomness. That observation follows because everything is circuit in the evaluation of $A^{\mathcal{O}}$, and thus the topology depends only on the input length. We define of oracle query tree of $\mathsf{IPFE.Dec}$ below, where Dec is an oracle-aided circuit, and the tree describes the structure of oracle queries made by Dec.

Definition 5.5 (Oracle query tree of Dec)**.** *For any IPFE decryption procedure* $\mathsf{Dec}^{\mathcal{O}}$ *w.r.t. extended (or two-layer) IHWE oracle $\mathcal{O}$, we suppose that* Dec *is an oracle-aided circuit of size polynomial in the security parameter λ; this is w.l.o.g. because* Dec *is a deterministic polynomial time Turing machine. Given any circuit* Dec*, the* oracle query tree $\mathsf{T}_{\mathsf{Dec}}$ *is defined below.*

- *The root node is associated with the circuit* Dec*. Each non-root node is an oracle query made by* $\mathsf{Dec}^{\mathcal{O}}$*, no matter directly or indirectly.*
- *The children of* Dec *are the direct queries from* Dec *to $\mathcal{O}$. Each non-root internal node u is a d-query, and the children of u are the queries made by (the verification performed by) the d-query. The ordering of the children nodes (from left to right) is defined as the ordering of circuit evaluation (from former to latter).*

Each node of $\mathsf{T}_{\mathsf{Dec}}$ *is* identified *in the standard way, i.e., by the sequence $(i_1, i_2, \dots)$ that denotes a path taking the i_1th child of the root, the i_2th child at the first layer, and so on. When we say the* ordering of tree nodes, *we mean the ordering of the corresponding queries in* Dec, *which coincide with the post-order traversal of* $\mathsf{T}_{\mathsf{Dec}}$.

We remark that $\mathsf{T}_{\mathsf{Dec}}$ is well-defined *without input*. Particularly, Dec is a circuit, and its children queries are fixed in e- or d-type and in the input lengths for all inputs of Dec. Also, fixing the input length of any d-query of the form $d_{\mathsf{V}}(w, \mathsf{lb})$, for any (w, lb) and the corresponding instance a, the verification $\mathsf{V}^{(\cdot)}(a, w)$ makes a child query in the same type and length because $\mathsf{V}^{(\cdot)}$ is a circuit and the length $|(a, w)|$ is the same. Also, notice that the number of tree nodes and the lengths of identifiers are all polynomial, as the total number of queries is polynomial.

Recall that we supposed that IPFE makes only $d_{\mathsf{V}^{\mathcal{O}'}}$-queries to invert labels. To meet the supposition, some leaf nodes may change from $d_{\mathsf{V}'}$-queries to $d_{\mathsf{V}^{\mathcal{O}'}}$-queries. Because that syntactically modifies the circuit Dec, the topology of $\mathsf{T}_{\mathsf{Dec}}$ is lightly changed but remains in polynomial size.

The compilation is formalized next, and Lemma 5.6 summarizes the probabilistic event of discovering the queries in the compilation; the proof of Lemma 5.6 is part of proof of Claim 5.13.1 and is deferred to Section 5.5.

Algorithm 9. $\mathsf{Compile}^{\mathcal{O}}(\mathsf{Dec}, \mathsf{sk}[\mathbf{v}], C_{\mathrm{com}})$

Input: An oracle-aided algorithm $\mathsf{Dec}^{(\cdot)}$ of IPFE, a functional decryption key $\mathsf{sk}[\mathbf{v}]$, and a set C_{com} consisting of $\tilde{t}_{\mathrm{c}}$ ciphertexts for some $\tilde{t}_{\mathrm{c}} \in \mathbb{N}$.
Output: A list of Q-A pairs Q_{com}.
Oracle: An oracle $\mathcal{O}$ is given.
Procedure: Let $\overline{\mathsf{Dec}}^{\mathcal{O}} := \mathsf{Canonicalize}^{\mathcal{O}}(\mathsf{Dec})$ be the canonicalized algorithm. (Recall that the input and output remain identical after canonicalization.) Perform the following.

```
1. Initialize $Q_{\mathrm{com}}$ as an empty list.

2. For each $\tilde{c}_j \in C_{\mathrm{com}}$, run $\overline{\mathsf{Dec}}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], \tilde{c}_j)$. For every query $q$ incurred in the execution,
   add $(q, \mathcal{O}(q))$ into $Q_{\mathrm{com}}$.

Finally, output the list $Q_{\mathrm{com}}$.
```

Lemma 5.6. *Let $\mathcal{O}$ be a two-layer IHWE oracle. For any oracle-aided $\mathbb{Z}_p^n$-SK-IPFE scheme with three algorithms $\mathcal{E} = (\mathsf{KGen}^{\mathcal{O}}, \mathsf{Enc}^{\mathcal{O}}, \mathsf{Dec}^{\mathcal{O}})$ as defined in Definition 3.9, let $\mathsf{T}_{\mathsf{Dec}}$ be the oracle query tree of* Dec. *For any master secret key* $\mathsf{msk} \in \{0,1\}^{\omega}$, *any key vector* $\mathbf{v} \in \mathbb{Z}_p^n$, *and any uniformly chosen message vector* $\mathbf{m} \in \mathbb{Z}_p^n$, *let* $\mathsf{sk}[\mathbf{v}] \leftarrow \mathsf{KGen}^{\mathcal{O}}(\mathsf{msk}, \mathbf{v})$, *and let c be the ciphertext obtained from* $\mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$. *Let* $\{\tilde{\mathbf{m}}_j\}_{j \in [\tilde{t}_{\mathrm{c}}]}$ *be a set of message vector uniformly sampled from $\mathbb{Z}_p^n$ and* $C_{\mathrm{com}} := \{\tilde{c}_j | \tilde{c}_j \leftarrow \mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \tilde{\mathbf{m}}_j)\}_{j \in [\tilde{t}_{\mathrm{c}}]}$. *For each d-query* $q = d_{\mathsf{V}}(\star, \mathsf{lb})$ *performed during* $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$, *i.e., q corresponds to a child of* Dec *on* $\mathsf{T}_{\mathsf{Dec}}$, *where* lb *is a label created in* KGen, *the following holds.*

- *If for all ciphertexts $\tilde{c} \in C_{\mathrm{com}}$, during* $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], \tilde{c}_j)$, *the answers to all d-queries on* lb *are $\bot$, then the answer to q is $\bot$ w.h.p. (over the randomness of $\mathbf{m}, \tilde{\mathbf{m}}_j$, and* Enc*).*

5.3 Detailed Attack, Brk

In this subsection, we describe how our attacker $\mathsf{Brk}^{\mathcal{O}}$ can break an oracle-aided canonicalized IPFE scheme $\mathcal{E} := (\mathsf{KGen}^{\mathcal{O}}, \mathsf{Enc}^{\mathcal{O}}, \mathsf{Dec}^{\mathcal{O}})$, where $\mathcal{O}$ is a two-layer IHWE oracle. The attacker Brk is provided with the oracle $\mathcal{O}$ and the public parameters of the IPFE scheme $\mathcal{E}$ and, listed as follows. For an oracle-aided IPFE scheme, let λ denote the security parameter, p be the modulus, and n be the dimension of the vectors. Also in the IPFE, let $\mu := \mu(\lambda)$ be the bit-length of the functional decryption key output by the key generation algorithm $\mathsf{KGen}^{\mathcal{O}}$, and let $\rho := \rho(\lambda)$ be the bit-length of *all* queries (i.e., queries associated with all nodes on the oracle query tree $\mathsf{T}_{\mathsf{Dec}}$) and answers to/from the oracle $\mathcal{O}$ in $\mathsf{KGen}^{\mathcal{O}}$. The number $\ell := \ell(\lambda)$ is the number of queries made in each algorithm of the IPFE. The attacker is given $\lambda, p, n, \mu, \rho, \ell$ as well as the oracle $\mathcal{O}$ in the security game against the challenger (Definition 3.9). In addition to the security parameter λ, the oracle $\mathcal{O}$ is parameterized by $\alpha(\lambda)$, which is the bit-length of instances (Definition 3.20).

Precise enumeration of key generation. Recall that our Brk follows the HLOW framework, which enumerates all possible challenge functional keys $\mathsf{sk}'[\mathbf{v}]$ as well as the oracle queries (Section 3.6). Following the framework, the challenge key $\mathsf{sk}'[\mathbf{v}]$ shall be generated using the same procedure as the colluded keys, $\mathsf{sk}[\mathbf{v}_i]$. However, as mentioned in Challenge 6 (Section 2.5), compilation introduces too many oracle queries, making the enumeration grow too large and defeating the framework. We briefly mentioned the idea in Solution 6. We formalize the idea by identifying the invertible labels below, and we will show that enumerating the indicators of invertible labels is sufficient.

Definition 5.7 (Invertible labels and their indicators). *Fix the IPFE scheme, fix a key vector $\mathbf{v}$, for any* msk, *and for any set of ciphertexts C_{com}. The* invertible labels *are the labels such that*

- *In* $\mathsf{sk}[\mathbf{v}] \leftarrow \mathsf{KGen}^{\mathcal{O}}(\mathsf{msk}, \mathbf{v})$, *the ith query is an e-query that outputs label* lb, *and*
- *Label* lb *is inverted successfully at least once in* $\mathsf{Compile}^{\mathcal{O}}(\mathsf{Dec}, \mathsf{sk}[\mathbf{v}], C_{\text{com}})$.

We will use a string of indicators $s \in \{0,1\}^{\ell}$ such that $s_i = 1$ if and only if the ith query of KGen *outputs an invertible label.*

Closure of a query-answer pair set. Notice that if there exists a known query-answer (Q-A) pair $((d_{\mathsf{V}}, w, \mathsf{lb}), (a, m))$ from an IHWE oracle, then the answer to the query $e(a, m)$ is also known to be lb. For a Q-A pair set Q, we define its *closure* as the set containing all Q-A pairs in Q together with all such implied Q-A pairs. We formally define this as follows.

For convenience of exposition, we introduce the "closure." Informally, whenever we see a d-type query-answer of the form $((d_{\mathsf{V}}, w, \mathsf{lb}), \mathsf{ans})$, if the answer $\mathsf{ans} \neq \perp$, then we infer that $((e, x), \mathsf{lb})$ is also a query-answer pair in the same oracle $\mathcal{O}$, called *closure*. The closure is defined below. We will not explicitly mention this henceforth.

Definition 5.8 (Partial IHWE oracle and the closure.). *A set S of query-answer (Q-A) pairs from an IHWE oracle is call a* partial IHWE oracle. *For any partial IHWE oracle S, the* closure *of S is defined to be the smallest superset of S such that if a Q-A pair in the form of $((d_{\mathsf{V}}, w, \mathsf{lb}), x = (a, m)) \in S$ for some $(a, m) \neq \perp$, then the Q-A pair $((e, a, m), \mathsf{lb})$ is in the closure of S.*

Remark 5.9. Recall Step 2(b)i of the $\mathsf{Canonicalize}$ algorithm (Algorithm 8). If a set consists of the Q-A pairs made by a canonicalized algorithm, then its closure is itself.

Colluding phase: Corrupting keys and setting up the challenge.

1. Uniformly sample $\mathbf{v} \leftarrow \mathbb{Z}_p^n$ as the challenge key vector.
2. Let $t, t_{\text{c}}, \tilde{t}_{\text{c}} \in \mathbb{N}$ be sufficiently large numbers in $\text{poly}(\lambda)$, to be chosen in the analysis. Sample $(\mathbf{v}_1, \ldots, \mathbf{v}_t) \leftarrow \mathsf{S}$ as colluded vectors, where S is a random $(n-1)$-dimensional subspace of $\mathbb{Z}_p^n$ subject to $\mathbf{v} \notin \mathsf{S}$.
3. Send $(\mathbf{v}_1, \ldots, \mathbf{v}_t, \mathbf{v})$ to the IPFE challenger (non-adaptively) and then receive the functional key $\mathsf{sk}[\mathbf{v}_i]$, which is the output of the key generation algorithm.
4. For all $j \in [t_{\text{c}} + \tilde{t}_{\text{c}}]$, receive from the IPFE challenger (non-adaptively) the challenge ciphertext and inner-product pairs, (c_j, m'_j), where $m'_j \in \mathbb{Z}_p$. The last $\tilde{t}_{\text{c}}$ ciphertexts will be used in compilation, and we use C_{com} to denote the set of these ciphertexts. (The last $\tilde{t}_{\text{c}}$ items of m'_j's are never used.)

Learning phase: Learning essential query-answer pairs.

1. For each $i \in [t]$, run the compilation $\mathsf{Compile}^{\mathcal{O}}(\mathsf{Dec}, \mathsf{sk}[\mathbf{v}_i], C_{\mathrm{com}})$ to get Q^i_{com}.
2. For each $j \in [t_{\mathrm{c}}]$:
   (a) Initialize an empty list $\bar{Q}^j_{\mathsf{dec}}$.
   (b) Repeat the following step until no new Q-A pair can be added into $\bar{Q}^j_{\mathsf{dec}}$:
      i. Execute the algorithm $\mathsf{Canonicalize}^{\mathcal{O}}\left(\mathsf{Dec}(\mathsf{sk}[\mathbf{v}_i], c_j), \bar{Q}^j_{\mathsf{dec}} \cup Q^i_{\mathrm{com}}\right)$ for each $i \in [t]$; For every query-answer (Q-A) pairs made during the execution, add it into $\bar{Q}^j_{\mathsf{dec}}$.

Simulation phase: Using the learned Q-A pairs to decrypt. This phase uses the learned Q-A pairs $\bar{Q}_{\mathsf{dec}} := \bigcup_{j\in[t_c]} \bar{Q}^j_{\mathsf{dec}}$, it is PSPACE, but it never queries the oracle $\mathcal{O}$. Recall that μ denotes the output length of the algorithm $\mathsf{KGen}^{\mathcal{O}}(\cdot)$, ρ denotes the bit-length of all direct and indirect Q-A pairs made by $\mathsf{KGen}^{\mathcal{O}}(\cdot)$, and ℓ denotes the maximal number of queries.

1. Try every possible list Q'_{kgen} of all direct and indirect Q-A pairs of KGen on the challenge key vector $\mathbf{v}$, every possible corresponding functional decryption key $\mathsf{sk}'[\mathbf{v}]$ generated by KGen, and every possible indicator of invertible labels s' (Definition 5.7). That is, for each $\mathsf{sk}'[\mathbf{v}] \in \{0,1\}^{\mu}$, each $Q'_{\mathsf{kgen}} \in \{0,1\}^{\rho}$, and each $s' \in \{0,1\}^{\ell}$, decrypt the ciphertexts using $\mathsf{sk}'[\mathbf{v}]$, where the answers to all queries are simulated as follows.
   (a) For each $j \in [t_{\mathrm{c}}]$,
      i. Initialize $\mathsf{Sim} := \mathsf{Sim}_{\mathrm{IHWE}}[\bar{Q}_{\mathsf{dec}}, Q'_{\mathsf{kgen}}, s']$ with $\bar{Q}_{\mathsf{dec}}$, Q'_{kgen}, and s', where $\mathsf{Sim}_{\mathrm{IHWE}}$ is a stateful algorithm, given later in Algorithm 10.
      ii. Simulate $\tilde{m}_j \leftarrow \mathsf{Dec}^{\mathsf{Sim}}(\mathsf{sk}'[\mathbf{v}], c_j)$; whenever Dec queries q, answer it with $\mathsf{ans} \leftarrow \mathsf{Sim}(q)$.
   (b) If $\tilde{m}_j = m'_j$ for all $j \in [t_{\mathrm{c}}]$, return 0.
2. Otherwise, all trials of $(Q'_{\mathsf{kgen}}, \mathsf{sk}'[\mathbf{v}], s')$ are unsuccessful, return 1.

5.3.1 Simulated IHWE Oracle, $\mathsf{Sim}_{\mathrm{IHWE}}$

For any Q-A lists $\bar{Q}_{\mathsf{dec}}, Q'_{\mathsf{kgen}}$ and indicator s', the simulated oracle $\mathsf{Sim}_{\mathrm{IHWE}}[\bar{Q}_{\mathsf{dec}}, Q'_{\mathsf{kgen}}, s']$ is a stateful Turing Machine, and its state S_Q is initialized by $\bar{Q}_{\mathsf{dec}} \cup Q'_{\mathsf{kgen}}$ and is updated after each query. The procedure is described in Algorithm 10.

Definition 5.10. *Given a list of Q-A pairs Q, the set $\mathsf{Label}(Q)$ is defined as the set of labels that appear in any Q-A pair in Q. This differs from Definition 4.9 because the IHWE instances are hidden.*

Algorithm 10. $\mathsf{Sim}_{\mathrm{IHWE}}[\bar{Q}_{\mathsf{dec}}, Q'_{\mathsf{kgen}}, s'](q)$

Initialize the state: State S_Q is initialized to the list $\bar{Q}_{\mathsf{dec}} \cup Q'_{\mathsf{kgen}}$, and state S_{kgen} is initialized to Q'_{kgen}.
Input and output: Given a query q as input, output an answer ans.

Answer the query q: Once initialized, the simulator takes an input query q, simulates an answer to q, and then appends the current Q-A pair to the state S_Q, so that a future answer can be consistent with the current answer. The answer is simulated using the following procedure.

1. If q is identical to a query in $\mathsf{Query}(S_Q)$, return the corresponding answer. Below we describe how to answer q when $q \notin \mathsf{Query}(S_Q)$.
2. If q is an e-query of the form $e(a, m)$, uniformly sample $\mathsf{lb} \in \{0,1\}^{\alpha+2\lambda} \setminus \mathsf{Label}(S_Q)$. Set the answer as lb and output lb.
3. If q is a d-query of the form $d_{\mathsf{V}}(w, \mathsf{lb})$:
   (a) If $((e, a, m), \mathsf{lb}) \notin S_Q$ for any (a, m), output $\perp$ as the answer.
   (b) Else if $((e, a, m), \mathsf{lb}) \in S_Q$ for some (a, m), let u_q be the node corresponding to q on $\mathsf{T}_{\mathsf{Dec}}$ (u_q can be found by keeping track of every query so far w.r.t. the query tree $\mathsf{T}_{\mathsf{Dec}}$ and the $\mathsf{Canonicalize}$ procedure):
      i. If 1) u_q is an internal node on $\mathsf{T}_{\mathsf{Dec}}$, and 2) $((e, a, m), \mathsf{lb})$ is the ith query in S_{kgen}, and 3) $s'_i = 0$, then set the answer of $\mathsf{V}^{(\cdot)}(a, w)$ as 0 and return $\perp$.
      ii. Otherwise, simulate $\mathsf{V}^{(\cdot)}(a, w)$, where every query q' asked by $\mathsf{V}^{(\cdot)}$ is answered by this procedure $\mathsf{Sim}_{\mathrm{IHWE}}$ with the current state S_Q. If $\mathsf{V}^{(\cdot)}$ outputs 0, return $\perp$; Otherwise, return (a, m).

5.3.2 Parameters

Setting parameters. We revisit parameters here.

- λ is the security parameter. For a $\mathbb{Z}_p^n$-SK-IPFE, $n := n(\lambda) \geq 3$ is a polynomial of λ. p is a prime number.
- $t_{\mathrm{c}} := t_{\mathrm{c}}(\lambda)$ is the number of ciphertext in the security experiment. Set $t_{\mathrm{c}} \geq \lambda + \mu + \rho + \ell$.
- Let $\ell := \ell(\lambda)$ be the number of queries made in each algorithm in the IPFE scheme (including all direct and indirect queries). Choose constants τ, τ' such that $\lambda^{\tau} \geq 4t_{\mathrm{c}}\lambda, \tau' > \log_{\lambda}(\ell) + \tau$. Choose $t \geq 4\ell\lambda^{\tau'}$ as in HLOW Lemma.
- $\tilde{t}_{\mathrm{c}} := \tilde{t}_{\mathrm{c}}(\lambda)$ is the number of ciphertexts used in compilation. Set $\tilde{t}_{\mathrm{c}} \geq t\ell\lambda^{2\tau}$.

Remark 5.11. The oracle we simulated has subtle differences from the real two-layer IHWE oracle. When simulating the answer to a d-query q on a label lb, the simulated answer is always $\perp$ in the following cases. However in the real IHWE oracle, the answer may not be $\perp$.

1. $((e, a, m), \mathsf{lb}) \notin S_Q$ for any (a, m).
2. q is in an internal node of $\mathsf{T}_{\mathsf{Dec}}$, $((e, a, m), \mathsf{lb})$ is the ith query in Q'_{kgen}, and $s'_i = 0$.

Another difference is that the label used to answer a new e-query may appear in the Q-A lists, $\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}}$ in the real IHWE oracle. However, in our simulation, the probability is 0 since we will sample labels from the space excluding all labels appearing in $\bar{Q}_{\mathsf{dec}} \cup Q'_{\mathsf{kgen}}$.

When Q'_{kgen} is the Q-A list of generating the real challenge functional decryption key $\mathsf{sk}[\mathbf{v}]$, the above two cases where the answers from simulated oracle and the real oracle differ will happen with only small probability. We will prove this in Section 5.4.

5.4 Proof for Separating IPFE from IHWE

We consider the single ciphertext case and then use union bound to complete the proof. For a single ciphertext case, let $\mathbf{v}$ be the challenge vector chosen by $\mathsf{Brk}^{\mathcal{O}}$, $\mathbf{m}$ denote the underlying message vector.

Let Q_{kgen} denote *all* Q-A pairs incurred in $\mathsf{KGen}^{\mathcal{O}}(\mathsf{msk}, \mathbf{v})$, including those are not asked by KGen; Q_{enc} denote the list of *all* Q-A pairs in $c \leftarrow \mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$; $\bar{Q}_{\mathsf{dec}}$ denote the list of all learned Q-A pairs in Learning phase; Q_{dec} denote the Q-A pairs directly made by $\mathsf{Canonicalize}^{\mathcal{O}}(\mathsf{Dec}(\mathsf{sk}[\mathbf{v}], c), \bar{Q}_{\mathsf{dec}} \cup Q_{\mathrm{com}})$.

We have the following lemma to show that the probability of successful decryption using the simulated oracle is not far from the decryption using the real oracle as long as the attacker's trail $(Q'_{\mathsf{kgen}}, \mathsf{sk}'[\mathbf{v}], s')$ hits the real $(Q_{\mathsf{kgen}}, \mathsf{sk}[\mathbf{v}], s)$.

Lemma 5.12 (Simulated decryption is correct)**.** *There exists a constant τ and a negligible function* $\mathrm{negl}(\cdot)$ *such that*

$$\Pr[\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c) = \langle \mathbf{v}, \mathbf{m} \rangle] - \Pr[\mathsf{Dec}^{\mathsf{Sim}_{\mathrm{IHWE}}[\bar{Q}_{\mathsf{dec}}, Q_{\mathsf{kgen}}, s]}(\mathsf{sk}[\mathbf{v}], c) = \langle \mathbf{v}, \mathbf{m} \rangle] \leq \lambda^{-\tau} + \mathrm{negl}(\lambda).$$

Proof sketch of Lemma 5.12. Before delving into the detailed proof, we outline the high-level idea. We begin with the following definition. For an IHWE oracle $\mathcal{O}$, a query q to $\mathcal{O}$, and a list of Q-A pairs Q of $\mathcal{O}$, we say that the answer to query q is *provided by* Q if one of the following conditions are satisfied:

1. There exists an answer ans such that $(q, \mathsf{ans}) \in Q$.
2. If q is a d-query of the form $d_{\mathsf{V}}(w, \mathsf{lb})$, $((e, a, m), \mathsf{lb}) \in Q$ for some $(a, m \neq \perp)$, and for every query q' asked by $\mathsf{V}^{\mathcal{O}}(a, w)$, its answer ans' is provided by Q.

To consistently answer a query performed in $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$, the "essential" information about $\mathcal{O}$ is included in $Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}}$. Since Brk tries every possible Q_{kgen}, the main challenge arises from the hidden information in Q_{enc}. Notice that Brk can only obtain the Q-A pairs in Q_{enc} by learning from the IPFE decryptions with the colluded functional keys.

As in Section 4, if a Q-A pair in Q_{enc} is correlated with $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$ but was not learned, Brk would fail to simulate some answers. This failure causes an incorrect inner product of $\mathbf{m}$ and $\mathbf{v}$. The additional challenge in the IHWE setting is that, due to the instance-hiding property, for some queries in Q_{enc}, there is no guarantee that Brk can learn them.

There are two cases when such a failure can happen. In the following, we suppose that for some k, the answer to the kth query is the first one that Brk fails to simulate (while the previous $k-1$ answers are properly simulated).

Case 1: *The kth answer is provided by Q_{enc}.* As in Section 4, a failure occurs when the k-th answer is given by Q_{enc} but not by $\bar{Q}_{\mathsf{dec}}$. We define the event Bad to capture such cases. We outline each subcase of Bad here and give the formal definition later.

For queries in the children of the root of the oracle query tree w.r.t. IPFE Dec: Case 1 of Bad describes that some queries in $\mathsf{IPFE.Enc}$ are also made by $\mathsf{IPFE.Dec}$, but $\mathsf{Brk}^{\mathcal{O}}$ fails to learn them; Case 2 of Bad covers the situation where $\mathsf{Brk}^{\mathcal{O}}$ fails to learn the preimage of the input label of a d-query, which is created during $\mathsf{IPFE.Enc}$.

For a d-query on a label lb in an internal node, its answer depends on the result of the verification, which takes a witness and the instance of lb as input. When Brk has learned the instance, it would simulate the verification result. Thus, we also need to consider the queries asked in the verification, which correspond to leaf nodes on the oracle query tree w.r.t. IPFE Dec: Cases 4, 5, and 6 of Bad are for queries that $\mathsf{Brk}^{\mathcal{O}}$ fails to learn; Cases 7, 8, and 9 of Bad are for labels whose preimages $\mathsf{Brk}^{\mathcal{O}}$ fails to learn.

In addition, there is a special case related to the simulation strategy for d-queries. Recall that for some unsuccessful d-queries on labels created in $\mathsf{IPFE.KGen}$, $\mathsf{Brk}^{\mathcal{O}}$ has no guarantee of learning the queries asked by these d-queries. The intended strategy is that $\mathsf{Brk}^{\mathcal{O}}$ identifies these d-queries using the indicator of invertible labels s and directly simulates $\perp$ as their answers, without attempting verification. Case 3 of Bad captures the failure where $\mathsf{Brk}^{\mathcal{O}}$ misidentifies the correct strategy: it simulates $\perp$ as the answer to a d-query q, while the real oracle's answer is not $\perp$.

A further issue remains even when we condition on Bad not happening. Suppose a valid label lb is used as input to a d-query, but lb has not been output by any preceding e-query that is visible to the IPFE algorithms. In this case, $\mathsf{Brk}^{\mathcal{O}}$ cannot know the preimage of lb. Moreover, when sampling a label as the answer to an e-query, $\mathsf{Brk}^{\mathcal{O}}$ avoids reusing labels that appear in any Q–A pair it has already learned or simulated. By contrast, the real oracle may output a label that has appeared before. We define the event Hit to capture these situations. Case 1 and Case 2 of Hit describe failures where some labels used as inputs are not outputs of any previous queries. Case 3 further captures the situation where the real oracle samples a label that has already appeared.

Case 2: *The kth answer is not fully provided by Q_{enc} but depends partially on Q_{enc}.* In this case, Q_{enc} does not provide the kth answer but restricts answer candidates by preventing label reuse. However, Brk, which does not have access to Q_{enc}, might reuse a label, which causes an inconsistency. We define this failure event as Col_k and prove it occurs with negligible probability.

Proof. The proof follows the same structure with that in Section 4.3. We start the detailed proof by defining the undesirable events Bad, Hit, and Col_k. The proof will utilize a sequence of hybrid experiments, where the sequence gradually replaces the kth query from the real IHWE oracle $\mathcal{O}$ to the simulator $\mathsf{Sim}_{\mathrm{IHWE}}$. The hybrid argument shows that IPFE Dec achieves nearly the same correctness using the simulator $\mathsf{Sim}_{\mathrm{IHWE}}$ as if the real IHWE oracle $\mathcal{O}$ were used. We remark that Bad and Hit are defined with respect to the real IHWE oracle $\mathcal{O}$, but Col_k is defined on a hybrid experiment, which uses both the real IHWE oracle $\mathcal{O}$ and the simulator. After the definition of each event, we will claim that the event happens with a small or negligible probability, and then we will prove Lemma 5.12 using the claims. The proofs of some claims will be deferred.

Definition 5.13. *The event* Bad *is defined as the union of the following cases. For two queries q_1 and q_2 in* $\mathsf{Query}(Q_{\mathsf{dec}})$*, let u_1 and u_2 be their corresponding nodes on* $\mathsf{T}_{\mathsf{Dec}}$*. We say that q_2 is a child query of q_1 if u_2 is a child of u_1. The notion of a child Q-A pair is defined analogously.*

1. *(Queries in children of the root.) There exists a query in* $\mathsf{Query}(Q_{\mathsf{enc}})$ *that is also directly performed by* $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$*, which corresponds to a child of the root of* $\mathsf{T}_{\mathsf{Dec}}$*, but it is not in* $\bar{Q}_{\mathsf{dec}}$*.*

2. *(Labels in children of the root from Enc.) There exist a label* lb*, an instance* a*, a plaintext* m*, and a witness* w *such that* lb *is an answer in* Q_{enc}*,* $(a, m) \neq \bot$ *is the preimage of* lb *w.r.t.* $\mathcal{O}$*, and* $((d_{\mathsf{V}}, w, \mathsf{lb}), (a, m)) \in Q_{\mathsf{dec}}$ *corresponds to a child of the root on* $\mathsf{T}_{\mathsf{Dec}}$*, but* $((d_{\mathsf{V}}, w', \mathsf{lb}), (a, m)) \notin \bar{Q}_{\mathsf{dec}}$ *for any witness* w'*.*

3. *(Labels in children of the root from KGen.) For any vector* $\mathbf{u}$*, let* $Q^{\mathbf{u}}_{\mathsf{kgen}}$ *be the list of all direct and indirect Q-A pairs performed by* $\mathsf{sk}[\mathbf{u}] \leftarrow \mathsf{KGen}^{\mathcal{O}}(\mathsf{msk}, \mathbf{u})$*, let* $Q^{\mathbf{u}}_{\mathrm{com}}$ *be the output of* $\mathsf{Compile}^{\mathcal{O}}(\mathsf{Dec}, \mathsf{sk}[\mathbf{u}], C_{\mathrm{com}})$*,* $s^{\mathbf{u}}$ *be the status string w.r.t.* $\mathbf{u}$*. There exist a vector* $\mathbf{u}$*, a label* lb*, an instance* a*, a plaintext* m*, a witness* w *such that* $\mathbf{u} \in \{\mathbf{v}, \mathbf{v}_1, ..., \mathbf{v}_t\}$*,* lb *is the answer of the* i*th query in* $Q^{\mathbf{u}}_{\mathsf{kgen}}$*,* $(a, m) \neq \bot$ *is the preimage of* lb *w.r.t.* $\mathcal{O}$*,* (a, m) *is the answer to a query performed by* $\mathsf{Dec}(\mathsf{sk}[\mathbf{u}], c)$*, but* $s^{\mathbf{u}}_i = 0$*.*

4. *(Queries in leaf nodes due to labels from Enc.) There exist a label* lb*, an instance* a*, a plaintext* m*, a witness* w*, and a query* q*, such that* lb *is an answer in* Q_{enc}*,* $(a, m) \neq \bot$ *is the preimage of* lb *w.r.t.* $\mathcal{O}$*,* $(d_{\mathsf{V}}, w, \mathsf{lb}) \in \mathsf{Query}(Q_{\mathsf{dec}})$ *has a child query* q*,* q *is in* $\mathsf{Query}(Q_{\mathsf{enc}})$ *but not in* $\mathsf{Query}(\bar{Q}_{\mathsf{dec}})$*.*

5. *(Queries in leaf nodes due to labels from KGen.) There exist a label* lb*, an instance* a*, a plaintext* m*, a witness* w*, and a query* q*, such that* lb *is an answer in* Q_{kgen}*,* $(a, m) \neq \bot$ *is the preimage of* lb *w.r.t* $\mathcal{O}$*,* $s_i = 1$*,* $(d_{\mathsf{V}}, w, \mathsf{lb}) \in \mathsf{Query}(Q_{\mathsf{dec}})$ *has a child query* q*,* q *is in* $\mathsf{Query}(Q_{\mathsf{enc}})$ *but not in* $\mathsf{Query}(\bar{Q}_{\mathsf{dec}})$*.*

6. *(Queries in leaf nodes due to labels from Dec.) There exist a label* lb*, an instance* a*, a plaintext* m*, a witness* w*, and a query* q*, such that* lb *is the answer of a Q-A pair incurred in* $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$*,* $(a, m) \neq \bot$ *is the preimage of* lb *w.r.t* $\mathcal{O}$*,* $(d_{\mathsf{V}}, w, \mathsf{lb}) \in \mathsf{Query}(Q_{\mathsf{dec}})$ *has a child query* q*,* q *is in* $\mathsf{Query}(Q_{\mathsf{enc}})$ *but not in* $\mathsf{Query}(\bar{Q}_{\mathsf{dec}})$*.*

7. *(Labels in leaf nodes due to labels from Enc.) There exist labels* $\mathsf{lb}_1, \mathsf{lb}_2$*, instances* a_1, a_2*, plaintexts* m_1, m_2*, a witness* w_1*, such that* $\mathsf{lb}_1, \mathsf{lb}_2$ *are answers in* Q_{enc}*,* $(a_1, m_1) \neq \bot$ *(resp.* $(a_2, m_2) \neq \bot$*) is the preimage of* lb_1 *(resp.* lb_2*) w.r.t.* $\mathcal{O}$*,* $(d_{\mathsf{V}}, w_1, \mathsf{lb}_1) \in \mathsf{Query}(Q_{\mathsf{dec}})$ *has a child query with* (a_2, m_2) *as the answer, but* (a_2, m_2) *is not an answer in* $\bar{Q}_{\mathsf{dec}}$*.*

8. *(Labels in leaf nodes due to labels from KGen.) There exist labels* $\mathsf{lb}_1, \mathsf{lb}_2$*, instances* a_1, a_2*, plaintexts* m_1, m_2*, a witness* w_1*, such that* lb_1 *is the answer of the* i*th query in* KGen*,* lb_2 *is an answer in* Q_{enc}*,* $(a_1, m_1) \neq \bot$ *(resp.* $(a_2, m_2) \neq \bot$*) is the preimage of* lb_1 *(resp.* lb_2*) w.r.t.* $\mathcal{O}$*,* $s_i = 1$*,* $(d_{\mathsf{V}}, w_1, \mathsf{lb}_1) \in \mathsf{Query}(Q_{\mathsf{dec}})$ *has a child query with* (a_2, m_2) *as the answer, but* (a_2, m_2) *is not an answer in* $\bar{Q}_{\mathsf{dec}}$*.*

9. *(Labels in leaf nodes due to labels from Dec.) There exist labels* $\mathsf{lb}_1, \mathsf{lb}_2$*, instances* a_1, a_2*, plaintexts* m_1, m_2*, a witness* w_1*, such that* lb_1 *is an answer in a Q-A pair incurred in* $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$*,* lb_2 *is an answer in* Q_{enc}*,* $(a_1, m_1) \neq \bot$ *(resp.* $(a_2, m_2) \neq \bot$*) is the preimage of* lb_1 *(resp.* lb_2*) w.r.t.* $\mathcal{O}$*,* $(d_{\mathsf{V}}, w_1, \mathsf{lb}_1) \in \mathsf{Query}(Q_{\mathsf{dec}})$ *has a child query with* (a_2, m_2) *as the answer, but* (a_2, m_2) *is not an answer in* $\bar{Q}_{\mathsf{dec}}$*.*

We claim that Bad happens with a small probability below; the proof is provided later in Section 5.5.

Claim 5.13.1. *For a $\mathbb{Z}_p^n$-SK-IPFE where $n := n(\lambda)$ is a polynomial, $n \geq 3$, p is a prime number, and p^{-n} is negligible, there exists a constant τ such that $\Pr[\mathsf{Bad}] \leq \lambda^{-\tau}$.*

The event Hit is defined as the following cases: The first case happens when the oracle is queried with a valid label as an input, but the label is never an output of a preceding query. The second case happens when the oracle answered to a query with a (valid or invalid) label such that the query is fresh, but the label exists identically in a preceding query. The "preceding" queries may be performed by the same algorithm or by another algorithm in the experiment. We formalize two relevant cases below.

Definition 5.14. *For a query $q \in \mathsf{Query}(Q_{\mathsf{dec}})$, let u_q denote the corresponding node on the oracle query tree $\mathsf{T}_{\mathsf{Dec}}$, $Q_{\mathsf{dec}}^{[-q]}$ denote the subset of Q_{dec} that consists of all queries that chronologically precede q. The event* Hit *is the union of the following two cases:*

1. *(Hit a valid input, same algorithm.) There exists a d-query in $\mathsf{Query}(Q_{\mathsf{enc}})$ with the input label* lb *such that* lb *is valid, but for each query $q' \in \mathsf{Query}(Q_{\mathsf{enc}})$ that is chronologically preceding q,* lb *is not an output label of q'.*

2. *(Hit a valid input, across algorithms.) Let $Q := Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}} \cup \bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{dec}}^{[-q]}$; There exists a query $q \in \mathsf{Query}(Q_{\mathsf{dec}})$ with the input label* lb *such that for each query $q' \in \mathsf{Query}(Q)$ that is chronologically preceding q,* lb *is not an output label of q'.*

3. *(Hit another label, across algorithms.) Let $Q := Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}} \cup \bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{dec}}^{[-q]}$; There exist a query $q \in \mathsf{Query}(Q_{\mathsf{dec}})$ with the output label* lb *such that for every $q' \in \mathsf{Query}(Q)$, it holds that $q \neq q'$, but* lb *appears in Q.*

Claim 5.14.1. *For any IHWE oracle $\mathcal{O}$ with parameter λ and any oracle-aided IPFE, the probability of* Hit *is bounded by $\Pr[\mathsf{Hit}] = 2^{-\Omega(\lambda)}$.*

We analyze the correctness of the simulated decryption by a sequence by hybrid experiments $\{\mathsf{Hyb}\}_{k \in [0,\ell]}$, where ℓ is the number of queries made in each algorithm of IPFE (including all direct and indirect queries).

Definition 5.15. *Hybrid experiment, Hyb_k. Let ℓ be the number of queries in each (compiled) algorithm. For each $k \in [\ell]$, the hybrid experiment Hyb_k answers the first k oracle queries using the real IHWE oracle $\mathcal{O}$ and simulates the remaining $\ell - k$ answers. That is,*

$$\mathsf{Hyb}_k := \begin{cases} 1 & \textit{if } \mathsf{Sim}_k = \langle \mathbf{v}, \mathbf{m} \rangle \\ 0 & \textit{otherwise,} \end{cases}$$

where Sim_ℓ is described in Algorithm 11. For $k \in [0, \ell - 1]$, Sim_k follows the same procedure as Sim_ℓ, but the queries performed by them are answered using different oracles:

- *The first k queries are answered by $\mathcal{O}$. We denote these k Q-A pairs as $Q_{\mathsf{dec},k}$.*
- *The remaining $\ell - k$ queries are answered by the simulated IHWE oracle $\mathcal{O}'_k := \mathsf{Sim}_{\mathrm{IHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{dec},k}, Q_{\mathsf{kgen}}, s](q)$ described in Algorithm 10.*

Sim_ℓ**.** In Sim_ℓ (Algorithm 11), the procedure $\mathsf{IPFE.Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$ is executed, and the queries are forwarded to the real IHWE oracle $\mathcal{O}$. Compared with $\mathsf{IPFE.Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$, the simulator Sim_ℓ also takes $\bar{Q}_{\mathsf{dec}}$, Q_{kgen}, and s as input. It directly performs some of the indirect queries in $\mathsf{IPFE.Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$ and records all queries along with their answers into the state S_Q, in the order they are forwarded to $\mathcal{O}$. The state S_Q is initialized as the list $\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}}$.

Algorithm 11. $\mathsf{Sim}_\ell[\bar{Q}_{\mathsf{dec}}, Q_{\mathsf{kgen}}, s](\mathsf{sk}[\mathbf{v}], c)$

Input: Take the functional decryption key $\mathsf{sk}[\mathbf{v}]$, the ciphertext c, the Q-A lists $\bar{Q}_{\mathsf{dec}}, Q_{\mathsf{kgen}}$, and the status string s as input.
Oracle: The real IHWE oracle $\mathcal{O}$ is given.
Initialize the states: A state S_Q is initialized to the list $\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}}$.
Procedure: Run $\mathsf{Dec}^{(\cdot)}(\mathsf{sk}[\mathbf{v}], c)$, whenever there is a query q, process it using the following procedure. Once obtaining the answer $\mathsf{ans} \leftarrow \mathcal{O}(q)$, add (q, ans) into S_Q. Before processing the next query, S_Q always includes (q, ans).

1. If q is identical to a query in $\mathsf{Query}(S_Q)$, query oracle $\mathsf{ans} \leftarrow \mathcal{O}(q)$. Below we describe how to process q when $q \notin \mathsf{Query}(S_Q)$.
2. If q is an e-query of the form $e(a, m)$, query oracle $\mathsf{ans} \leftarrow \mathcal{O}(q)$.
3. If q is a d-query of the form $d_{\mathsf{V}}(w, \mathsf{lb})$:
   (a) If $((e, a, m), \mathsf{lb}) \notin S_Q$ for any a, m, query oracle $\mathsf{ans} \leftarrow \mathcal{O}(q)$.
   (b) Else if $((e, a, m), \mathsf{lb}) \in S_Q$, execute the following steps:
      i. If q corresponds to an internal node on $\mathsf{T}_{\mathsf{Dec}}$, $((e, a, m), \mathsf{lb})$ is the ith Q-A pair in Q_{kgen}, and $s_i = 0$, query oracle $\mathsf{ans} \leftarrow \mathcal{O}(q)$.
      ii. Otherwise, for every query q_1 asked by $\mathsf{V}^{(\cdot)}(a, w)$, emulate the answer using this procedure.

Finally, output whatever $\mathsf{Dec}^{(\cdot)}(\mathsf{sk}[\mathbf{v}], c)$ outputs.

All queries in Sim_ℓ are (direct or indirect) queries in $\mathsf{Dec}^{(\cdot)}(\mathsf{sk}[\mathbf{v}], c)$, and they are answered using the real oracle $\mathcal{O}$. Therefore, the decryption result in the output of Sim_ℓ is identical to that of $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c)$. We have

$$\Pr[\mathsf{Hyb}_\ell = 1] = \Pr[\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c) = \langle \mathbf{v}, \mathbf{m} \rangle].$$

In Sim_0, all queries are answered using $\mathcal{O}'_0$, which is exactly the same as that in the simulated oracle $\mathsf{Sim}_{\mathrm{IHWE}}[\bar{Q}_{\mathsf{dec}}, Q_{\mathsf{kgen}}, s]$. Thus we have

$$\Pr[\mathsf{Hyb}_0 = 1] = \Pr[\mathsf{Dec}^{\mathsf{Sim}_{\mathrm{IHWE}}[\bar{Q}_{\mathsf{dec}}, Q_{\mathsf{kgen}}, s]} = \langle \mathbf{v}, \mathbf{m} \rangle].$$

Lemma 5.16. *For each $k \in [0, \ell]$, the first k queries in Sim_k have the following properties:*

1. *For a d-query of the form $q = d_{\mathsf{V}}(w, \mathsf{lb})$, if the preimage of lb is not provided by the input lists or preceding queries, that is, $((e, x), \mathsf{lb}) \notin S_Q$ for any x, where S_Q includes all queries that have been forwarded to $\mathcal{O}$ before q and the Q-A pairs in the input, the queries in the verification triggered by q are not forwarded to $\mathcal{O}$.*

2. *For a d-query of the form $q = d_{\mathsf{V}}(w, \mathsf{lb})$, if $((e, a, m), \mathsf{lb})$ is in the input lists or chronologically preceding q for some (a, m):*
   - *If q corresponds to a internal node on $\mathsf{T}_{\mathsf{Dec}}$, $((e, a, m), \mathsf{lb})$ is the ith Q-A pair in Q_{kgen}, and $s_i = 0$, its children queries (i.e., queries asked by $\mathsf{V}^{(\cdot)}(a, w)$) are not forwarded to $\mathcal{O}$.*
   - *Otherwise, all of its children queries (if any) are chronologically preceding q. That is, the result of $\mathsf{V}^{(\cdot)}(a, w)$ can be simulated using solely S_Q.*

With these properties, for each $k \in [0, \ell - 1]$, the adjacent experiments Sim_k and Sim_{k+1} differ in exactly *one* Q–A pair. This makes it easier to work with hybrid argument.

Now we define the event Col_k w.r.t. the simulated oracle on the hybrid experiment Hyb_{k-1}, where the simulated oracle outputs a *colliding* label.

Definition 5.17 (Col_k event for Hyb_{k-1})**.** *For the $(k-1)$th hybrid experiment Hyb_{k-1}, we define the event Col_k as the union of the following cases.*

- *The kth simulated Q-A pair is $((e, a, m), \mathsf{lb})$ for some (a, m), lb appears in Q_{enc}, but $((e, a, m), \mathsf{lb}) \notin \bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{dec},k-1}$.*

By the definition, the probability of the event Col_k depends on the list $\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{dec},k-1}$ from the real oracle. Consequently, it depends on the events Bad and Hit, both defined with respect to the real oracle. Fortunately, we only need to bound the probability of Col_k occurring conditioned on that neither Bad nor Hit occurs. The following claim shows the required bound.

Claim 5.17.1. $\Pr[\mathsf{Col}_k | \neg\mathsf{Bad} \wedge \mathsf{Hit}] = 2^{-\Omega(\lambda)}$.

Proof. The event Col_k occurs only when the kth query is a new e-query whose answer needs to be sampled. In the simulation, the label used to answer such a query is sampled uniformly from $\{0,1\}^{\alpha+2\lambda}$ excluding all labels appearing in $\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{dec},k-1}$. The number of different labels in $\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{dec},k-1}$ is no more than $(t+2)\ell$. Meanwhile, the number of different labels appearing in Q_{enc} is no more than ℓ. Therefore, the probability that the sampled label appears in Q_{enc} is no more than $\ell/(2^{\alpha+2\lambda} - (t+2)\ell) = 2^{-\Omega(\lambda)}$. □

Conditioned on the good event that Bad, Hit, and Col_k never happen, we claim that Hyb_k and Hyb_{k-1} will be identical. The claim is formally stated below, and its proof is deferred to Section 5.7.

Claim 5.17.2. *For all $k \in [\ell]$, it holds that*

$$\Pr[\mathsf{Hyb}_k = 1 | \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = \Pr[\mathsf{Hyb}_{k-1} = 1 | \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}].$$

Next, we bound the difference between the first and the last hybrid experiments using the above claims. We begin by removing the condition $\neg\mathsf{Col}_k$ from the above claim so that the difference between hybrids is not conditioned on k. Let $T_1 := \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}$, and let $T_2 := \neg\mathsf{Col}_k$. We have that

$$\begin{aligned} &\Pr[\mathsf{Hyb}_k = 1 \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] \\ =&\Pr[\mathsf{Hyb}_k = 1 \mid T_1 \wedge T_2] \cdot \Pr[T_2 \mid T_1] + \Pr[\mathsf{Hyb}_k = 1 \mid T_1 \wedge \neg T_2] \cdot \Pr[\neg T_2 \mid T_1] \\ \le&\Pr[\mathsf{Hyb}_k = 1 \mid T_1 \wedge T_2] + \Pr[\neg T_2 \mid T_1]. \end{aligned}$$

Similarly, we have that

$$\begin{aligned}
&\Pr[\mathsf{Hyb}_{k-1} = 1 \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] \\
=&\Pr[\mathsf{Hyb}_{k-1} = 1 \mid T_1 \wedge T_2] \cdot \Pr[T_2 \mid T_1] + \Pr[\mathsf{Hyb}_{k-1} = 1 \mid T_1 \wedge \neg T_2] \cdot \Pr[\neg T_2 \mid T_1] \\
\geq&\Pr[\mathsf{Hyb}_{k-1} = 1 \mid T_1 \wedge T_2] \cdot (1 - \Pr[\neg T_2 \mid T_1]).
\end{aligned}$$

Thus, using Claim 5.17.2 and then Claim 5.17.1, the difference can be bounded by

$$\begin{aligned}
&\Pr[\mathsf{Hyb}_{k} = 1 \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] - \Pr[\mathsf{Hyb}_{k-1} = 1 \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] \\
\leq&\Pr[\mathsf{Hyb}_{k} = 1 \mid T_1 \wedge T_2] + \Pr[\neg T_2 \mid T_1] - \Pr[\mathsf{Hyb}_{k-1} = 1 \mid T_1 \wedge T_2] \cdot (1 - \Pr[\neg T_2 \mid T_1]) \\
\leq&2\Pr[\mathsf{Col}_k \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = 2^{-\Omega(\lambda)}. \qquad (2)
\end{aligned}$$

We now show the difference between the first and the last hybrid experiments.

$$\begin{aligned}
&\Pr[\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{v}], c) = \langle \mathbf{v}, \mathbf{m} \rangle] - \Pr[\mathsf{Dec}^{\mathsf{Sim}_{\mathrm{IHWE}}[\bar{Q}_{\mathsf{dec}}, Q_{\mathsf{kgen}}, Q_{\mathsf{enc}}, Q_{\mathrm{com}}, \mathsf{T}_{\mathrm{com}}]}(\mathsf{sk}[\mathbf{v}], c) = \langle \mathbf{v}, \mathbf{m} \rangle] \\
=&\Pr[\mathsf{Hyb}_{\ell} = 1] - \Pr[\mathsf{Hyb}_0 = 1] \\
\leq&\Pr[\mathsf{Hyb}_{\ell} = 1 \mid T_1]\Pr[T_1] + \Pr[\neg T_1] - \Pr[\mathsf{Hyb}_0 = 1 \mid T_1]\Pr[T_1] \\
\leq&\Pr[\mathsf{Hyb}_{\ell} = 1 \mid T_1] + \Pr[\neg T_1] - \Pr[\mathsf{Hyb}_0 = 1 \mid T_1] - \Pr[\neg T_1] \\
\leq&\ell \cdot 2^{\Omega(-\lambda)} + \lambda^{-\tau} + 2^{-\Omega(\lambda)} = \lambda^{-\tau} + 2^{-\Omega(\lambda)}.
\end{aligned}$$

The last inequality follows from Eq. (2) and from $\Pr[\mathsf{Bad} \vee \mathsf{Hit}] \leq \Pr[\mathsf{Bad}] + \Pr[\mathsf{Hit}] \leq \lambda^{-\tau} + 2^{-\Omega(\lambda)}$ by Claim 5.13.1 and Claim 5.14.1. This concludes the proof of Lemma 5.12. □

We next analyze the success probability of Brk in the security game. Recall that Brk aims to guess the challenge bit b. For each case of b, we show that the output of Brk equals to b with high probability, using union bound and Lemma 5.12.

Theorem 5.18. *Let b' be the output of $\mathsf{Brk}^{\mathcal{O}}$, for a $\mathbb{Z}_p^n$-SK-IPFE $\mathcal{E}^{\mathcal{O}}$ with correctness $1 - 1/2^{\lambda}$, where λ is the security parameter, we have*

$$\Pr[b' = 0 \mid b = 0] \geq 1 - \frac{1}{\lambda}.$$

Proof. When $Q'_{\mathsf{kgen}} = Q_{\mathsf{kgen}}$, $\mathsf{sk}'[\mathbf{v}] = \mathsf{sk}[\mathbf{v}]$, and $s' = s$, by Lemma 5.12, there is a constant τ such that

$$\Pr[\mathsf{Dec}^{\mathsf{Sim}}(\mathsf{sk}'[\mathbf{v}], c_j) = \langle \mathbf{v}, \mathbf{m} \rangle] \geq 1 - 2^{-\lambda} - 2^{-\Omega(\lambda)} - \lambda^{-\tau}$$

for each $c_j, j \in [t]$, where $\mathsf{Sim} = \mathsf{Sim}_{\mathrm{IHWE}}[\bar{Q}_{\mathsf{dec}}, Q'_{\mathsf{kgen}}, s']$. Since $\mathsf{Brk}^{\mathcal{O}}$ goes through all choices of Q'_{kgen}, $\mathsf{sk}'[\mathbf{v}]$, and s', it must hit the correct Q-A list Q_{kgen}, functional decryption key $\mathsf{sk}[\mathbf{v}]$, and status string s at some point. By union bound, we have

$$\Pr[b' = 0 \mid b = 0] \geq 1 - t_c\left(\frac{1}{2^{\lambda}} + \lambda^{-\tau} + 2^{-\Omega(\lambda)}\right).$$

Because $\lambda^{\tau} \geq 4t_c\lambda$, we get $\Pr[b' = 0 \mid b = 0] \geq 1 - O(1/\lambda)$ for sufficiently large λ.

□

Theorem 5.19. *Let b' be the output of $\mathsf{Brk}^{\mathcal{O}}$, for a $\mathbb{Z}_p^n$-SK-IPFE $\mathcal{E}^{\mathcal{O}}$, we have* $\Pr[b' = 1 | b = 1] \geq 1 - \frac{1}{2^{\lambda}}$.

Proof. When $b = 1$ in the security game (Definition 3.9), each m'_j is uniformly chosen from $\mathbb{Z}_p$ for every $j \in [t_c]$. Fix $\mathsf{sk}'[\mathbf{v}] \in \{0,1\}^\mu$, $Q'_{\mathsf{kgen}} \in \{0,1\}^\rho$, and $s' \in \{0,1\}^\ell$ the probability of $\mathsf{Dec}^{\mathsf{Sim}}(\mathsf{sk}'[\mathbf{v}], c_j) = m'_j$ is no more than $1/p$. Furthermore, since every m'_j is independently chosen, the probability that $\mathsf{Dec}^{\mathsf{Sim}}(\mathsf{sk}'[\mathbf{v}], c_j) = m'_j$ holds for every $j \in [t_c]$ is at most $1/p^{t_c}$. Given the parameters we set before, i.e., $t_c \geq \lambda + \mu + \rho + \ell$, applying the union bound over all possible choices of $\mathsf{sk}'[\mathbf{v}]$, Q'_{kgen}, and s' gives

$$\Pr[b' = 1 \mid b = 1] \geq 1 - \frac{2^{\mu+\rho+\ell}}{q^{t_c}} \geq 1 - \frac{1}{2^\lambda}.$$

□

Putting together Theorem 5.18 and Theorem 5.19, we have Theorem 5.1.

5.5 Proof of Claim 5.13.1

Claim 5.13.1. *For a $\mathbb{Z}_p^n$-SK-IPFE where $n := n(\lambda)$ is a polynomial, $n \geq 3$, p is a prime number, and p^{-n} is negligible, there exists a constant τ such that* $\Pr[\mathsf{Bad}] \leq \lambda^{-\tau}$.

We recall the Bad event in Definition 5.13 and Claim 5.13.1 as the above.

Let $2^{[\ell]}$ denote the set of subsets of $[\ell]$. Since at most ℓ queries are (directly or indirectly) performed by $\mathsf{Enc}^{\mathcal{O}}(\cdot)$ or $\mathsf{KGen}^{\mathcal{O}}(\cdot)$, we can use $[\ell]$ to index all of these Q-A pairs in an algorithm. We define the following functions for any fixed oracle $\mathcal{O}$, master secret key msk, vector $\mathbf{m}$, $c \leftarrow \mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$ for some fixed encryption randomness, $Q^{\mathbf{u}}_{\mathrm{com}} \leftarrow \mathsf{Compile}(\mathsf{Dec}, \mathsf{sk}[\mathbf{u}], C_{\mathrm{com}})$ for a set of ciphertexts C_{com} generated by $\mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \cdot)$, where $\mathsf{sk}[\mathbf{u}] \leftarrow \mathsf{KGen}^{\mathcal{O}}(\mathsf{msk}, \mathbf{u})$ is a function of a variable $\mathbf{u}$.

- (Cases 1, 5, and 6: Queries in children of the root and in leaf nodes due to labels from KGen and Dec.)

  $F^{(1)}(\mathbf{u}) : \mathbb{Z}_p^n \to 2^{[\ell]}$ maps a vector $\mathbf{u} \in \mathbb{Z}_p^n$ to a subset $Z \subseteq [\ell]$ such that $j \in Z$ if and only if the jth (direct or indirect) Q-A pair in $\mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$ is also a Q-A pair directly performed by $\mathsf{Canonicalize}^{\mathcal{O}}(\mathsf{Dec}(\mathsf{sk}[\mathbf{u}], c), Q^{\mathbf{u}}_{\mathrm{com}})$.

- (Cases 2, 8, and 9: Labels in children of the root from Enc and in leaf nodes due to labels from KGen and Dec.)

  $F^{(2)}(\mathbf{u}) : \mathbb{Z}_p^n \to 2^{[\ell]}$ maps a vector $\mathbf{u} \in \mathbb{Z}_p^n$ to a subset $Z \subseteq [\ell]$ such that $j \in Z$ if and only if the answer of the jth (direct or indirect) query in $\mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$ is a label lb with preimage x, and there exists a Q-A pair $((d_{\mathsf{V}}, w, \mathsf{lb}), x)$ directly performed by $\mathsf{Canonicalize}^{\mathcal{O}}(\mathsf{Dec}(\mathsf{sk}[\mathbf{u}], c), Q^{\mathbf{u}}_{\mathrm{com}})$.

- (Case 4: Queries in leaf nodes due to labels from Enc.)

  $F^{(1)}_{\mathsf{lb}}(\mathbf{u}) : \mathbb{Z}_p^n \to 2^{[\ell]}$ maps a vector $\mathbf{u} \in \mathbb{Z}_p^n$ to a subset $Z \subseteq [\ell]$ such that $j \in Z$ if and only if the jth (direct or indirect) query q in $\mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$ 1) is a (direct or indirect) query in $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{u}], c)$ and 2) is a child query of a d-query on label lb.

- (Case 7: Labels in leaf nodes due to labels from Enc.)

  $F^{(2)}_{\mathsf{lb}}(\mathbf{u}) : \mathbb{Z}_p^n \to 2^{[\ell]}$ maps a vector $\mathbf{u} \in \mathbb{Z}_p^n$ to a subset $Z \subseteq [\ell]$ such that $j \in Z$ if and only if 1) the jth (direct or indirect) query in $\mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$ outputs a label lb', 2) there exists a (direct or indirect) query q of the form $(d_{\mathsf{V}}, w, \mathsf{lb}')$ in $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{u}], c)$, and 3) q is a child query of a d-query with the input label lb.

By HLOW Lemma (Lemma 2.3), given any constant τ', there exists $t \in \mathrm{poly}(\lambda)$ such that

$$\Pr[F^{(j)}(\mathbf{v}) \not\subseteq \bigcup_{i\in[t]} F^{(j)}(\mathbf{v}_i)] \leq \lambda^{-\tau'}, \qquad \Pr[F^{(j)}_{\mathsf{lb}}(\mathbf{v}) \not\subseteq \bigcup_{i\in[t]} F^{(j)}_{\mathsf{lb}}(\mathbf{v}_i)] \leq \lambda^{-\tau'},$$

for each $j = 1, 2$ and any label lb that is the answer of a query in $\mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$, where $\mathbf{v}$ is the challenge key vector, and $\mathbf{v}_1, ..., \mathbf{v}_t$ are the colluded key vectors.

Since there are at most ℓ queries in $\mathsf{Enc}^{\mathcal{O}}(\mathsf{msk}, \mathbf{m})$, the probability of the above cases can be bounded by $(2 + 2\ell)\lambda^{-\tau'}$.

Next we consider the remaining Case 3:

Fix a vector $\mathbf{u}$, in the compilation $Q^{\mathbf{u}}_{\mathrm{com}} \leftarrow \mathsf{Compile}^{\mathcal{O}}(\mathsf{Dec}, \mathsf{sk}[\mathbf{u}], C_{\mathrm{com}})$, the underlying message vectors (denoted as $\{\tilde{\mathbf{m}}_j\}_{j\in[\tilde{t}_{\mathrm{c}}]}$) of C_{com} are *uniformly and independently* chosen. $Q^{\mathbf{u}}_{\mathrm{com}}$ includes all Q-A pairs directly performed by $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{u}], \tilde{c}_j)$.

For each e-query made by KGen, let i denote its index in Q_{kgen} and lb_i denote its answer. Define $\mathsf{E}_{i,j}$ as the event that a successful d-query on lb_i is directly asked in $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{u}], \tilde{c}_j)$, E_i denote the event that a successful d-query on lb_i is directly asked in $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}[\mathbf{u}], c)$, where c is the challenge ciphertext. Since all message vectors, including the challenge one $\mathbf{m}$, are uniformly and independently chosen. Fix $i \in [\ell]$, the oracle $\mathcal{O}$, and the functional decryption key $\mathsf{sk}[\mathbf{u}]$, we have

$$\Pr_{\tilde{\mathbf{m}}_1,...,\tilde{\mathbf{m}}_{\tilde{t}_{\mathrm{c}}},\mathbf{m}}[\mathsf{E}_{i,j}] = \Pr_{\tilde{\mathbf{m}}_1,...,\tilde{\mathbf{m}}_{\tilde{t}_{\mathrm{c}}},\mathbf{m}}[\mathsf{E}_i]$$

for every $j \in [\tilde{t}_{\mathrm{c}}]$. Since there are at most ℓ (direct and indirect) queries made by KGen, the probability of Case 3 for a vector $\mathbf{u} \in \{\mathbf{v}, \mathbf{v}_1, ..., \mathbf{v}_t\}$ can be bounded as

$$\begin{aligned}\sum_{i\in[\ell]} \Pr_{\tilde{\mathbf{m}}_1,...,\tilde{\mathbf{m}}_{\tilde{t}_{\mathrm{c}}},\mathbf{m}}[\mathsf{E}_i \wedge \neg\mathsf{E}_{i,1} \wedge \cdots \wedge \neg\mathsf{E}_{i,\tilde{t}_{\mathrm{c}}}] &\leq \sum_{i\in[\ell]} \Pr_{\tilde{\mathbf{m}}_1,...,\tilde{\mathbf{m}}_{\tilde{t}_{\mathrm{c}}},\mathbf{m}}[\mathsf{E}_i](1 - \Pr_{\tilde{\mathbf{m}}_1,...,\tilde{\mathbf{m}}_{\tilde{t}_{\mathrm{c}}},\mathbf{m}}[\mathsf{E}_i])^{\tilde{t}_{\mathrm{c}}} \\ &\leq \sum_{i\in[\ell]} \frac{1}{\tilde{t}_{\mathrm{c}}} \leq \frac{\ell}{\tilde{t}_{\mathrm{c}}}.\end{aligned}$$

Set $\tau' > \log_\lambda(\ell) + \tau$ and $\tilde{t}_{\mathrm{c}} \geq t\ell\lambda^{2\tau}$ as described in Section 5.3.2, then for constant τ, $\Pr[\mathsf{Bad}] \leq (2\ell + 2)\lambda^{-\tau'} + t\ell/\tilde{t}_{\mathrm{c}} \leq \lambda^{-\tau}$ when λ is sufficiently large.

5.6 Proof of Claim 5.14.1

Claim 5.14.1. *For any IHWE oracle $\mathcal{O}$ with parameter λ and any oracle-aided IPFE, the probability of* Hit *is bounded by* $\Pr[\mathsf{Hit}] = 2^{-\Omega(\lambda)}$.

We reall the Hit event in Definition 5.14 and Claim 5.14.1 as the above.

We will consider the cases of Hit separately below. According to the definition of IPFE in Definition 3.9, the master secret key msk, challenge vector $\mathbf{v}$, and message $\mathbf{m}$ are independent of the random IHWE oracle $\mathcal{O}$.

Case 1, hit a valid input, same algorithm. There are $2^{\alpha+2\lambda}$ distinct labels and $2^{\alpha+\lambda}$ valid ones in Q_{enc}. For a label appearing for the first time in a query, the probability that it is valid is no more than $2^{\alpha+\lambda}/(2^{\alpha+2\lambda}-\ell)$. This is because that $\pi(\cdot)$ is random permutation, and the preceding Q-A pairs of Q_{enc} can exclude at most ℓ invalid labels. By union bound over all labels, the probability is at most $\ell \cdot 2^{\alpha+\lambda}/(2^{\alpha+2\lambda}-\ell)$.

Case 2, hit a valid input, across algorithms. Similarly, there are $2^{\alpha+\lambda}$ valid labels and the whole label space is $2^{\alpha+2\lambda}$. A label is valid in a d-query, but it neither exists in $Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}}$ nor has appeared in the visible history of this query as an answer with probability no more than $2^{\alpha+\lambda}/(2^{\alpha+\lambda}-3\ell)$. This is because $\pi(\cdot)$ is a random permutation and the previous Q-A pairs can exclude only 3ℓ valid labels at most. By union bound over all labels, the probability is at most $\ell 2^{\alpha+\lambda}/(2^{\alpha+2\lambda}-3\ell)$.

Case 3, hit another label, across algorithms. Fix an e-query q with the output label lb. In the list $\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{kgen}} \cup Q_{\mathsf{enc}} \cup Q_{\mathsf{dec}}^{[-q]}$, there are at most $(t+3)\ell$ number of distinct queries and thus distinct labels. The permutation π samples a label from at least $2^{3\lambda}-(t+3)\ell$ labels. Hence, the sample is fresh except with probability at most $(t+3)\ell/(2^{3\lambda}-(t+3)\ell)$ and there exists such a sample with probability $(t+3)\ell^2/(2^{3\lambda}-(t+3)\ell)$.

Combining the three cases above, we have $\Pr[\mathsf{Hit}] \leq \ell 2^{\alpha+\lambda+1}/(2^{\alpha+2\lambda}-3\ell)+(t+3)\ell^2/(2^{3\lambda}-(t+3)\ell) = 2^{-\Omega(\lambda)}$.

5.7 Proof of Claim 5.17.2

Claim 5.17.2. *For all $k \in [\ell]$, it holds that*

$$\Pr[\mathsf{Hyb}_k = 1 | \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = \Pr[\mathsf{Hyb}_{k-1} = 1 | \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}].$$

The claim is restated above. Also recall the Col_k event in Definition 5.17. Recall that the first $k-1$ Q-A pairs forwarded by Sim_{k-1} and Sim_k are exactly the same and using the real IHWE oracle $\mathcal{O}$, and thus the kth query appearing in them is identical. Let $Q_{\mathsf{Dec},k-1}$ denote the first $k-1$ Q-A pairs w.r.t the real IHWE oracle $\mathcal{O}$. Now we demonstrate that the answers to the kth query from $\mathcal{O}$ and $\mathcal{O}'_{k-1} := \mathsf{Sim}_{\mathrm{IHWE}}[\bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{dec},k-1}, Q_{\mathsf{kgen}}, s]$ follow the same distribution when Hit, Bad, and Col_k does not occur.

We proceed with a case analysis to demonstrate the distribution difference between the kth answers from the real oralce $\mathcal{O}$ and $\mathcal{O}'_{k-1}$.

Recall that in Definition 5.15, $\mathcal{O}'_{k-1}$ is the hybrid oracle whose first $k-1$ queries are answered by $\mathcal{O}$ and the last $\ell-k+1$ queries are answered by the simulator $\mathcal{O}'_{k-1}$ in the hybrid experiment of Sim_{k-1}. Below, we consider the kth query-answer in the experiment of Sim_{k-1}, where the query is identical to the next hybrid experiment Sim_k (which is identical to the experiment with respect to real $\mathcal{O}$), but the answer is simulated by $\mathsf{Sim}_{\mathrm{IHWE}}$.

Let ans denote the kth answer from $\mathcal{O}$, ans' denote the kth answer from $\mathcal{O}'_{k-1}$, and $Q := Q_{\mathsf{kgen}} \cup \bar{Q}_{\mathsf{dec}} \cup Q_{\mathsf{dec},k-1}$ which is the list of all Q-A pairs to/from the real oracle known to the simulator.

When the kth query also exists in $\mathsf{Query}(Q)$ (recall that $\mathsf{Query}(Q)$ is the set of all queries appearing in Q), both the real oracle and the simulator will answer the same string. For the cases where the kth query does not exist in $\mathsf{Query}(Q)$, we categorize them according to according to the nodes it belongs to and provide the analysis below.

1. When the kth query is in a child node of the root of $\mathsf{T}_{\mathsf{Dec}}$:
   (a) The kth query is an e-query of the form (e,a,m):
      i. If $((e,a,m),lb) \in Q_{\mathsf{enc}}$ for some lb:

The real oracle will answer lb, and thus $\Pr[\mathsf{ans} = \mathsf{lb}] = 1$.
Conditioned on $\neg\mathsf{Bad}$ (Case 1: Queries in children nodes of root), $((e, a, m), \mathsf{lb}) \in Q$, and thus $\Pr[\mathsf{ans}' = \mathsf{lb} \mid \neg\mathsf{Bad}] = 1$.
Therefore, $\Pr[\mathsf{ans} = \mathsf{ans}' \mid \neg\mathsf{Bad}] = 1$ in this case.

ii. Else, $((e, a, m), \mathsf{lb}) \notin Q_{\mathsf{enc}}$ for any lb:
Conditioned on $\neg\mathsf{Hit}$ (Case 3: Hit another label, across algorithms), the kth answer from the real oracle is uniformly distributed in $\{0, 1\}^{\alpha+2\lambda} \setminus \mathsf{Label}(Q \cup Q_{\mathsf{enc}})$.
Conditioned on $\neg\mathsf{Col}_k$, the kth answer from the simulator is also uniformly distributed in $\{0, 1\}^{\alpha+2\lambda} \setminus \mathsf{Label}(Q \cup Q_{\mathsf{enc}})$.
Therefore, $\Pr[\mathsf{ans} = \mathsf{ans}' \mid \neg\mathsf{Hit} \wedge \neg\mathsf{Col}_k] = 1$ in this case.

(b) The kth query is an d-query of the form $(d_{\mathsf{V}}, w, \mathsf{lb})$:

i. If $((e, x), \mathsf{lb}) \notin Q$ for any x:
Conditioned on $\neg\mathsf{Bad}$ (Case 2: Labels in children nodes of root from Enc) and $\neg\mathsf{Hit}$ (Cases 1 and 2: Hit a valid input), lb is an invalid label. The real oracle will answer $\perp$, i.e., $\Pr[\mathsf{ans} = \perp \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = 1$.
The simulator will answer $\perp$. $\Pr[\mathsf{ans}' = \perp] = 1$.
Therefore, $\Pr[\mathsf{ans} = \mathsf{ans}' \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = 1$ in this case.

ii. If $((e, a, m), \mathsf{lb}) \in Q_{\mathsf{kgen}}$ for some (a, m), let i denote its index in Q_{kgen}, $s_i = 0$:
Conditioned on $\neg\mathsf{Bad}$ (Case 3: Labels in children nodes of root from KGen), the real oracle will answer $\perp$. $\Pr[\mathsf{ans} = \perp \mid \neg\mathsf{Bad}] = 1$.
The simulator will answer $\perp$.
Therefore, $\Pr[\mathsf{ans} = \mathsf{ans}' \mid \neg\mathsf{Bad}] = 1$.

iii. If $((e, a, m), \mathsf{lb}) \in Q_{\mathsf{kgen}}$ for some (a, m), let i denote its index in Q_{kgen}, $s_i = 0$; Or $((e, a, m), \mathsf{lb}) \in Q \setminus Q_{\mathsf{kgen}}$ for some (a, m):
The real oracle will run $\mathsf{V}^{\mathcal{O}}(a, w)$, then answer (a, m) if $1 \leftarrow \mathsf{V}^{\mathcal{O}}(a, w)$ and $\perp$ otherwise.
The simulator will simulate $\mathsf{V}^{\mathcal{O}}(a, w)$, where all queries are answered using Q. If the simulation of this verification is correct, then it will answer the kth query correctly. By the second item of Lemma 5.16, all Q-A pairs in $\mathsf{V}^{\mathcal{O}}(a, w)$ are already in Q. Since the Q-A pairs in Q are all to-from the real oracle, we have $\Pr[\mathsf{V}^{\mathcal{O}'_{k-1}}(a, w) = \mathsf{V}^{\mathcal{O}}(a, w)] = 1$, which ensures the correctness of the simulated kth answer.
Therefore, $\Pr[\mathsf{ans} = \mathsf{ans}'] = 1$ in this case.

2. When the kth query is in a leaf node on $\mathsf{T}_{\mathsf{Dec}}$:

(a) The kth query is an e-query of the form (e, a, m):

i. If $((e, a, m), lb) \in Q_{\mathsf{enc}}$ for some lb:
The real oracle will answer lb, and thus $\Pr[\mathsf{ans} = \mathsf{lb}] = 1$.
Conditioned on $\neg\mathsf{Bad}$ (Cases 4, 5, and 6: Queries in leaf nodes), $((e, a, m), \mathsf{lb}) \in Q$, and thus $\Pr[\mathsf{ans}' = \mathsf{lb} \mid \neg\mathsf{Bad}] = 1$.
Therefore, $\Pr[\mathsf{ans} = \mathsf{ans}' \mid \neg\mathsf{Bad}] = 1$ in this case.

ii. Else, $((e, a, m), \mathsf{lb}) \notin Q_{\mathsf{enc}}$ for any lb:
In this case, the analysis is identical to Item 1(a)ii. We have $\Pr[\mathsf{ans} = \mathsf{ans}' \mid \neg\mathsf{Hit} \wedge \neg\mathsf{Col}_k] = 1$.

(b) The kth query is a d-query of the form $(d_{\mathsf{V}}, w, \mathsf{lb})$:

i. If $((e, x), \mathsf{lb}) \notin Q$ for any x:
Conditioned on $\neg\mathsf{Bad}$ (Cases 7, 8, and 9: Labels in leaf nodes) and $\neg\mathsf{Hit}$ (Cases 1 and 2: Hit a valid input), lb is an invalid label. The real oracle will answer $\perp$, i.e., $\Pr[\mathsf{ans} = \perp \mid \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = 1$.
The simulator will answer $\perp$.
Therefore, $\Pr[\mathsf{ans} = \mathsf{ans}' \mid \neg\mathsf{Bad}] = 1$.

ii. If $((e, a, m), \mathsf{lb}) \in Q$ for some (a, m):
Since the queries in leaf nodes do not make queries, both real oracle and simulator will compute $\mathsf{V}(a, w)$, answer (a, m) if $1 \leftarrow \mathsf{V}(a, w)$ and $\perp$ otherwise.
Therefore, $\Pr[\mathsf{ans} = \mathsf{ans}'] = 1$ in this case.

When considering the condition that none of Bad, Hit, or Col_k occurs, the distributions of the kth answers from Sim_{k-1} and Sim_k are identical. It follows directly that the distributions of the remaining $\ell - k$ Q-A pairs in these two experiments are identical under the same condition. Therefore, we have $\Pr[\mathsf{Hyb}_{k-1} = 1 | \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}] = \Pr[\mathsf{Hyb}_k = 1 | \neg\mathsf{Col}_k \wedge \neg\mathsf{Bad} \wedge \neg\mathsf{Hit}]$.

5.8 On the Monolithic Separation

6 Updates and Errata

In this full version, the first two sections are essentially identical to the published proceeding version [HLL25]. However, we fixed some minor issues, listed below.

- In Section 2.2 of the proceeding version, we assumed that "$(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ are all deterministic" w.l.o.g., which was wrong and replaced by only assuming KGen is deterministic for simplicity. The assumption was only used in Section 2.2 to define the (deterministic) function $F(\mathbf{u})$, and we can accommodate randomized algorithms in such F; see Footnote 3.

Acknowledgment

We thank the anonymous reviewers of Crypto 2025 and TCC 2025 for their valuable feedback, including insightful directions, fixing bugs, comparison to related works, relating to the separation of IO, and showing that unbounded ciphertexts are necessary. W. L. thanks Mingyuan Wang for discussing the open problems on IPFE during the early stages of the project.

References

[ABDP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*, volume 9020 of *Lecture Notes in Computer Science*, pages 733–751. Springer, Berlin, Heidelberg, March / April 2015. 2, 5, 10, 26

[ABDP16] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Better security for functional encryption for inner product evaluations. Cryptology ePrint Archive, Report 2016/011, 2016. 10

[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, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 500–518. Springer, Berlin, Heidelberg, August 2013. 5

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 308–326. Springer, Berlin, Heidelberg, August 2015. 1

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 333–362. Springer, Berlin, Heidelberg, August 2016. 2, 10

[AS15] Gilad Asharov and Gil Segev. Limits on the power of indistinguishability obfuscation and functional encryption. In Venkatesan Guruswami, editor, *56th Annual Symposium on Foundations of Computer Science*, pages 191–209. IEEE Computer Society Press, October 2015. 1, 2, 4

[AV19] Prabhanjan Ananth and Vinod Vaikuntanathan. Optimal bounded-collusion secure functional encryption. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 174–198. Springer, Cham, December 2019. 1, 4

[BDOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, Berlin, Heidelberg, May 2004. 1

[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, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Berlin, Heidelberg, August 2001. 1

[BPR$^+$08] Dan Boneh, Periklis A. Papakonstantinou, Charles Rackoff, Yevgeniy Vahlis, and Brent Waters. On the impossibility of basing identity based encryption on trapdoor permutations. In *49th Annual Symposium on Foundations of Computer Science*, pages 283–292. IEEE Computer Society Press, October 2008. 2, 4, 34

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, Berlin, Heidelberg, March 2011. 1, 4

[BSW12] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: a new vision for public-key cryptography. *Commun. ACM*, 55(11):56–64, November 2012. 1

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th Annual Symposium on Foundations of Computer Science*, pages 171–190. IEEE Computer Society Press, October 2015. 1

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 535–554. Springer, Berlin, Heidelberg, February 2007. 1, 4

[CFM21] Geoffroy Couteau, Pooya Farshim, and Mohammad Mahmoody. Black-box uselessness: Composing separations in cryptography. In James R. Lee, editor, *ITCS 2021: 12th Innovations in Theoretical Computer Science Conference*, volume 185, pages 47:1–47:20. LIPIcs, January 2021. 23

[CKP15] Ran Canetti, Yael Tauman Kalai, and Omer Paneth. On obfuscation with random oracles. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 456–467. Springer, Berlin, Heidelberg, March 2015. 19

[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, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 733–764. Springer, Cham, December 2018. 5

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. 3

[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 Annual Symposium on Foundations of Computer Science*, pages 40–49. IEEE Computer Society Press, October 2013. 1, 4

[GGHZ16] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 480–511. Springer, Berlin, Heidelberg, January 2016. 1

[GKLM12] Vipul Goyal, Virendra Kumar, Satyanarayana V. Lokam, and Mohammad Mahmoody. On black-box reductions between predicate encryption schemes. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 440–457. Springer, Berlin, Heidelberg, March 2012. 2, 4, 10, 34

[GKM$^+$00] Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold, and Mahesh Viswanathan. The relationship between public key encryption and oblivious transfer. In *41st Annual Symposium on Foundations of Computer Science*, pages 325–335. IEEE Computer Society Press, November 2000. 2, 4

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 555–564. ACM Press, June 2013. 1, 2, 4

[GMM17] Sanjam Garg, Mohammad Mahmoody, and Ameer Mohammed. Lower bounds on obfuscation from all-or-nothing encryption primitives. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 661–695. Springer, Cham, August 2017. i, 1, 3, 4, 5, 6, 7, 13, 24, 25, 26, 27, 28, 30, 31, 35, 37, 38

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 194–213. Springer, Berlin, Heidelberg, February 2007. 4

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523. Springer, Berlin, Heidelberg, August 2015. 4

[HLL25] Jinye He, Shiyu Li, and Wei-Kai Lin. Lower bounds on inner-product functional encryption from all-or-nothing encryption primitives. In *TCC*, 2025. The publisher's version of this paper. 72

[HLOW24] Mohammad Hajiabadi, Roman Langrehr, Adam O'Neill, and Mingyuan Wang. On the black-box complexity of private-key inner-product functional encryption. In *TCC 2024: 22nd Theory of Cryptography Conference, Part III*, Lecture Notes in Computer Science, pages 318–343. Springer, Cham, November 2024. i, 2, 5, 6, 7, 8, 13, 23, 26, 32, 33, 34

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing*, pages 44–61. ACM Press, May 1989. 3, 4

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM Symposium on Theory of Computing*, pages 60–73. ACM Press, June 2021. 1

[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, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 670–699. Springer, Cham, May / June 2022. 1

[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: 11th International Conference on Security in Communication Networks*, volume 11035 of *Lecture Notes in Computer Science*, pages 544–562. Springer, Cham, September 2018. 2

[KY09] Jonathan Katz and Arkady Yerukhimovich. On black-box constructions of predicate encryption from trapdoor permutations. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 197–213. Springer, Berlin, Heidelberg, December 2009. 2, 4

[Mer78] Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, April 1978. 3

[MP12] Mohammad Mahmoody and Rafael Pass. The curious case of non-interactive commitments - on the power of black-box vs. non-black-box use of primitives. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 701–718. Springer, Berlin, Heidelberg, August 2012. 19

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. 1

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In Richard A. DeMillo, David P. Dobkin, Anita K. Jones, and Richard J. Lipton, editors, *Foundations of Secure Computation*, pages 165–179. Academic Press, 1978. 3

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20. Springer, Berlin, Heidelberg, February 2004. 3, 4, 24

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

[Wat14] Brent Waters. A punctured programming approach to adaptively secure functional encryption. Cryptology ePrint Archive, Report 2014/588, 2014. 1

[Wat15] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 678–697. Springer, Berlin, Heidelberg, August 2015. 1