Dynamic Decentralized Functional Encryptions from Pairings in the Standard Model

Duy Nguyen[0009-0002-7892-9146]

Telecom Paris, Institut Polytechnique de Paris, France
`dinh.nguyen@telecom-paris.fr`

Abstract. Dynamic Decentralized Functional Encryption (DDFE), introduced by Chotard *et al.* (CRYPTO'20), represents a robust generalization of (Multi-Client) Functional Encryption. It allows users to dynamically join and contribute private inputs to individually controlled joint functions without requiring a trusted authority.

Recently, Shi *et al.* (PKC'23) proposed the first Multi-Client Functional Encryption scheme for function-hiding inner products (FH-IP) without relying on random oracles. Unfortunately, their construction still requires a trusted key authority, leaving open the question of whether a full-fledged FH-IP-DDFE can exist in the standard model.

In this work, we answer this question affirmatively by introducing Updatable Pseudorandom Zero Sharing, a novel concept that provides both the critical functionality and security properties needed to construct secure DDFE schemes in the standard model.

Our second contribution is a novel proof strategy, which preserves adaptive security when transforming any functional encryption scheme for FH-IP into FH-IP-DDFE. Together, these two techniques enable a modular construction of FH-IP-DDFE that is secure against adaptive message and key queries in the standard model.

Additionally, our pseudorandom zero-sharing scheme is highly versatile, enabling the first DDFE for attribute-weighted sums in the standard model, complementing the recent ROM-based construction by Agrawal *et al.* (CRYPTO'23).

Keywords: dynamic decentralized, functional encryption, pairing, standard model

1 Introduction

1.1 Dynamic Decentralized Functional Encryption

Functional Encryption. Functional Encryption (FE) is a robust cryptographic paradigm introduced by Sahai and Waters [SW05, BSW11]. It addresses the all-or-nothing limitations of standard public-key cryptosystems by offering fine-grained control over access to encrypted data through functional decryption keys. More precisely, each ciphertext ct_x encrypts a specific value x, and each decryption key dk_f encapsulates a function f. When a receiver uses dk_f to decrypt ct_x, what it can learn is only the result $f(x)$, and nothing more about x.

Abdalla *et al.* [ABDP15] addressed the challenge of constructing FE based on standard assumptions for restricted, yet expressive, classes of computational functions. In their work, they presented FE constructions for inner products (IPFE), where each private input is represented as a vector $\boldsymbol{x}$, each function is represented by a vector $\boldsymbol{y}$, and decryption yields $\boldsymbol{x}^\top \cdot \boldsymbol{y}$.

Within the realm of practical FE based on standard assumptions for computation over encrypted data, starting from [ABDP15], significant improvements have been made in terms of security [BJK15, ALS16, ALMT20], functionality [BCFG17, AGW20, ACGU20], efficiency [CLT18, TT18, DP19, MKMS22] and support for multi-user scenarios [ACF+18, CDG+18, CDSG+20, ZLZ+24].

Functional Encryption for Multiple Users. The concepts of Multi-Input Functional Encryption (MIFE) and Multi-Client Functional Encryption (MCFE) were introduced in [GGG+14, GKL+13]. These concepts generalize FE to enable multiple clients to independently contribute their encrypted individual inputs to the computation of joint functions. Concretely, with the help of possibly a trusted authority, each slot owner receives a private encryption key sk_i and encrypts an input x_i under some label ℓ as $\mathsf{ct}_{x_i,\ell}$. By collecting ciphertexts of all slot owners $(\mathsf{ct}_{x_i,\ell})_i$ under a same label and a functional decryption key dk_f generated from the authority's master secret key msk, a receiver can obtain no more information than $f((x_i)_i)$.

In the multi-client setting, each slot owner is assumed to be an independent client: the security guarantees that only ciphertexts under a same label can be decrypted together, and the input privacy of honest clients holds even when a subset of clients is corrupted. In the multi-input setting, all labels are assumed to be the same, all slot owners are originally assumed to be the same user (yet each input can be sent independently instead of simultaneously as in single-input FE), and then no corruption is considered.

Therefore, any MCFE designed for a specific functionality automatically implies a MIFE for the same functionality by using a fixed label for all encryptions. Conversely, an MIFE designed for general functions directly implies an MCFE for general functions, as the label can be contained in every plaintext, and the function can verify that every slot uses the same label. However, when considering MIFE schemes for only restricted classes of functions in the literature, this equivalence does not hold.

Since their introduction, the research line on practical MIFE/MCFE constructions has witnessed a dynamic array of works seeking developments in the aspects of security [CDG+18,LT19,AGT22,SV23], functionality [CDSG+20, ACGU20, AGT21a, NPP22, ATY23], and in the relation with single-input FE [ABG19].

Dynamic Decentralized Functional Encryption. Standard MCFE models often encounter the key escrow problem, stemming from the reliance on a trusted authority. To address this challenge, Chotard *et al.* introduced the concept of decentralized MCFE (DMCFE) [CDG+18] and later Dynamic Decentralized Functional Encryption (DDFE) [CDSG+20]. In DMCFE, the need for a trusted authority is entirely removed, granting each client complete control over their encrypted data and the generation of functional decryption keys. DDFE is an extended notion of DMCFE by preserving all decentralized features while enabling the join of new clients at various stages during the lifetime of the system through a non-interactive setup.

The development of DMCFE/DDFE since the introduction of the first scheme for inner products [CDG+18] has seen significant progress on various aspects. These include a variety of functionalities, as seen in constructions for sums, inner products, and all-or-nothing message encapsulations in [CDSG+20] and attribute-weighted sums in [ATY23]. Efforts to enhance security have led to DMCFE constructions for function-hiding inner products [AGT21b], for lattice-based assumptions in the standard model [LT19], and for optimal security notions [NPP23b]. The relations between DMCFE and other FE notions have been established in these works [ABG19, ABKW19, AGT21b]. Practical features like verifiability [NPP23a] and robustness [LWG+23] have also been integrated, enhancing the applicability of DMCFE in various real-world scenarios.

Multi-Party Functional Encryption. In addition to the above notions of functional encryption, there are other concepts designed for various multi-user settings. These include those supporting distributed keys [MJ18,Cha07,LW11,BCFG17], as well as those supporting both distributed ciphertexts and keys [ACF+20]. All these notions fall under the umbrella of Multi-Party Functional Encryption [AGT21b]. However, in this paper, our focus will be exclusively on DDFE, which supports both distributed ciphertexts and keys and stands out as the most generalized notion within Multi-Party Functional Encryption.

Open Problems in DDFE. To the best of our knowledge, DDFE constructions based on standard assumptions remain limited to the classes of function-hiding inner products (FH-IP) and function-revealing attribute-weighted sums (AWS), which strictly capture the class of function-revealing inner products. These functionalities can be described more precisely as follows:

– In DDFE for FH-IP: each client, identified by a public key $\mathsf{pk} \in \mathcal{PK}$, uses its own corresponding secret key $\mathsf{sk}_{\mathsf{pk}}$ to encrypt its private input $\boldsymbol{x}_{\mathsf{pk}}$ under a public message label ℓ_M and for a public user list $\mathcal{U}_M$. Similarly, each client uses $\mathsf{sk}_{\mathsf{pk}}$ to generate a decryption key for its private vector $\boldsymbol{y}_{\mathsf{pk}}$ under a public key label ℓ_K and for a public user list $\mathcal{U}_K$. The labels ℓ_M and ℓ_K impose constraints on which messages and functions can be aggregated, respectively, while the lists $\mathcal{U}_M$ and $\mathcal{U}_K$ specifies the sets of parties whose ciphertexts and decryption keys can be combined, respectively. The FH-IP functionality verifies the conditions $[\mathcal{U}_M = \mathcal{U}_K]$, $[(\ell_M, \mathcal{U}_M)$ is consistent across all ciphertexts$]$ and $[(\ell_K, \mathcal{U}_K)$ is consistent across all decryption keys$]$ during decryption,. If these conditions are met, the FH-IP functionality outputs

$$\sum_{\mathsf{pk} \in \mathcal{U}_K} \boldsymbol{x}_{\mathsf{pk}}^\top \cdot \boldsymbol{y}_{\mathsf{pk}};$$

otherwise, it outputs nothing. The function-hiding security ensures that no additional information on individual $\boldsymbol{x}_{\mathsf{pk}}$ and $\boldsymbol{y}_{\mathsf{pk}}$ is revealed.

– In DDFE for AWS: each client pk chooses a user list $\mathcal{U}_M$, a label ℓ to encrypt its AWS inputs $\{\boldsymbol{z}_{\mathsf{pk},j}\}_{j\in[N_{\mathsf{pk}}]}$ which are private and $\{\boldsymbol{x}_{\mathsf{pk},j}\}_{j\in[N_{\mathsf{pk}}]}$ which are public. For key generation, each client pk chooses a set of users $\mathcal{U}_K$ and a list of arithmetic branching programs (ABPs) $\boldsymbol{f} = \{f_{\mathsf{pk}}\}_{\mathsf{pk}\in\mathcal{U}_K}$. The functionality verifies the conditions $[\mathcal{U}_M = \mathcal{U}_K]$, $[(\ell, \mathcal{U}_M)$ is consistent across all ciphertexts$]$ and $[\boldsymbol{f}$ is consistent across all decryption keys$]$. If these conditions are met, the AWS functionality outputs

$$\sum_{\mathsf{pk}\in\mathcal{U}_K}\sum_{j\in[N_{\mathsf{pk}}]} f_{\mathsf{pk}}(\boldsymbol{x}_{\mathsf{pk},j})^\top \cdot \boldsymbol{z}_{\mathsf{pk},j};$$

otherwise, it outputs nothing. The security ensures that no additional information on individual $\{\boldsymbol{z}_{\mathsf{pk},j}\}_{j\in[N_{\mathsf{pk}}]}$ is revealed.

It is important to highlight that all state-of-the-art DDFE constructions [AGT21b, ATY23] for the above functionalities are pairing-based and rely on random oracles for their security proofs.

On the other hand, in the development of (Decentralized) MCFE for (function-hiding) inner products, several works [LT19, ABG19, SV23] have focused on achieving standard-model security for their respective (Decentralized) MCFE schemes. Our work contributes to this ongoing research by addressing the first question:

Do there exist any DDFE constructions for function-hiding inner products and for attribute-weighted sums that are secure under standard assumptions in the standard model?

All existing FH-IPFE schemes rely on pairings, while lattice-based constructions have been proven impossible under common approaches [Üna20], therefore the second question we explore is:

Does the adaptive security of any pairing-based FE for function-hiding inner products imply the adaptive security[1] *of DDFE for the same functionality?*

This work provides affirmative answers to both questions, resulting in a modular construction of adaptively secure FH-IP-DDFE and a modular construction of selectively secure AWS-DDFE, both within the standard model.

1.2 Our Contributions

The contributions can be listed as follows:

– **Novel Pseudorandom Zero Sharing**: We introduce a new concept called *Updatable Pseudorandom Zero Sharing* (UZS). Compared to standard pseudorandom zero sharing, this notion simultaneously offers the following properties:
  1. it enables the local update from a zero share into a new one, and this updating algorithm can support the *bilinear update* property, which ensures the compatibility of the algorithm with operations allowed in pairing groups;
  2. its security additionally guarantees the pseudorandomness for the updated shares even if their corresponding non-updated shares are revealed; this security holds against an unbounded subset of corrupted parties;

  We demonstrate the feasibility of this notion by providing a construction in DDH groups, which is notably secure in the standard model.
– **Dynamic Decentralized Functional Encryptions Without RO**: Using UZS as a building block, we provide the first DDFE constructions for function-hiding inner products and attribute-weighted sums that are secure in the standard model. Comparative details with related works are provided in Figure 1.
– **Injection Lemma for Adaptive Security of FH-IP-DDFE**: As another independent contribution, in the security proof, we introduce an essential lemma that enables achieving adaptive security for FH-IP-DDFE from the adaptive security of *any* FH-IPFE scheme. The novel proof helps modularize the DDFE construction and establishes a previously unknown security relation between these two primitives.

[1] Here, adaptive security for DDFE refers to security under adaptively chosen messages and functions, similar to that of single-input FE.

Scheme	Function Class	Function Hiding	(Dynamic) Decentralized	Without RO	Adaptive key/message queries	Assumption + (any FE as Building Block)	Per-client CT size
[CDG+18]	IP	✗	✓	✗	✓	SXDH	$O_\lambda(d)$
[ABG19]	IP	✗	✓	✓	✓	IPFE	$O_\lambda(d \cdot n)$
[LT19]	IP	✗	✓	✓	✓	LWE	$O_\lambda(d)$
[CDSG+20]	IP	✗	✓	✗	✗	DDH + IPFE	$O_\lambda(d)$
[ABM+20]	IP	✗	✓	✗	✓	DCR	$O_\lambda(d)$
[AGT21b]	IP	✓	✓	✗	✗	SXDH +FH-IPFE	$O_\lambda(d)$
[SV23]	IP	✓	✗	✓	✓	DLin	$O_\lambda(d)$
Our FH-IP DDFE	IP	✓	✓	✓	✓	SXDH +FH-IPFE	$O_\lambda(d+n)$
[ATY23]	AWS	✗	✓	✗	✗	MDDH +AWSw/IP-FE	$O_\lambda(N(kn_0+n_1))$
Our AWS DDFE	AWS	✗	✓	✓	✗	SXDH + AWSw/IP-FE	$O_\lambda(N(kn_0+n_1+n))$

Fig. 1. Comparison with prior (Decentralized) MCFE schemes. The notation $O_\lambda(\cdot)$ indicates that terms related to the security parameter λ are hidden. We let d be an inner-product dimension, n be a number of clients whose ciphertexts and decryption keys can be combined, N be a $\mathsf{poly}(\lambda)$-unbounded number of AWS inputs, n_0 and n_1 be the input and output dimensions of ABPs, and k be the parameter of the MDDH assumption. For DDFE, all constructions are secure in static-corruption setting.

1.3 Technical Overview

Challenges in Removing Trusted Setup. In the (decentralized) multi-client setting, one of the fundamental security requirements is that only inputs encrypted under a same message label and only keys generated under a same key label can be decrypted together, otherwise the decryption should output nothing.

Thus, a common approach in constructing (decentralized) MCFE schemes, including those for (function-hiding) inner products and attribute-weighted sums, is to rely on a correlated one-time-pad technique: the one-time pads should be freshly generated by each pair of message and key labels, as well as by each client, to independently randomize the information corresponding to each client's private pair of input and key object.

Without obscuring the reliance on random oracles for its security, we can simplify the DDFE scheme for FH-IP in [AGT21b] as follows: leveraging a private use of function-hiding IPFE, each client $\mathsf{pk} \in \mathcal{U}$ encrypts a private $\boldsymbol{x}_{\mathsf{pk}}$ as

$$\mathsf{ct}_{\mathsf{pk}} = \mathsf{IPE.Enc}(\mathsf{sk}_{\mathsf{pk}}, [\boldsymbol{x}_{\mathsf{pk}}, \mathbf{0}, h_{\ell_M,\mathcal{U}}, 0]_1)$$

where $[h_{\ell_M,\mathcal{U}}]_1 = \mathcal{H}(\ell_M, \mathcal{U})$ and $\mathcal{H}$ is a hash function onto the group modeled as a random oracle. The decryption key for a private $\boldsymbol{y}_{\mathsf{pk}}$ is generated as

$$\mathsf{dk}_{\mathsf{pk}} = \mathsf{IPE.DKGen}(\mathsf{sk}_{\mathsf{pk}}, [\boldsymbol{y}_{\mathsf{pk}}, \mathbf{0}, z_{\mathsf{pk},\ell_K}, 0]_2)$$

where z_{pk,ℓ_K} is generated on label ℓ_K by a pseudorandom zero-sharing (PZS) scheme between parties in $\mathcal{U}$ so that $\sum_{\mathsf{pk}\in\mathcal{U}} z_{\mathsf{pk},\ell_K} = 0$. On one hand, the correctness holds as $\sum_{\mathsf{pk}\in\mathcal{U}} z_{\mathsf{pk},\ell_K} \cdot h_{\ell_M,\mathcal{U}} = 0$ in the sum of all FH-IPFE decryptions.

On the other hand, via zero slots, and by using hybrids relying on the function-hiding security of each client's FH-IPFE in the non-corrupted set $\mathcal{H}$ and each key label ℓ_K, one can move the term $[z_{\mathsf{pk},\ell_K}]_2$ from $\mathsf{dk}_{\mathsf{pk}}$ to the term $[z_{\mathsf{pk},\ell_K} \cdot h_{\ell_M,\mathcal{U}}]_1$ in $\mathsf{ct}_{\mathsf{pk}}$. At this step, one will have

$$[z_{\mathsf{pk},\ell_K} \cdot h_{\ell_M,\mathcal{U}}]_1 \overset{\mathsf{PZS},\mathsf{SXDH}}{\approx} [R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}}]_1$$

where $R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}}$ are random subjected to $\sum_{\mathsf{pk}\in\mathcal{H}} R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}} = -\sum_{\mathsf{pk}\in\mathcal{U}\setminus\mathcal{H}} z_{\mathsf{pk},\ell_K} \cdot h_{\ell_M,\mathcal{U}}$. Each $R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}}$ then serves as a correlated one-time pad that randomizes each term ${\boldsymbol{x}^b_{\mathsf{pk}}}^\top \cdot \boldsymbol{y}^b_{\mathsf{pk}}$ in the same way it randomizes ${\boldsymbol{x}^0_{\mathsf{pk}}}^\top \cdot \boldsymbol{y}^0_{\mathsf{pk}}$, where b is the challenge bit of the security game. It is important to note that for the SXDH assumption to hold, $[h_{\ell_M,\mathcal{U}}]_1$ must be a uniformly random group element to all clients, which requires the use of a random oracle.

A more recent construction in [SV23] addresses this problem in the multi-client setting, in which a trusted key authority is necessitated. Extending this construction to the decentralized setting is challenging, as the assumption of a trusted key authority is, in fact, stronger than the assumption of a random oracle.

It is also worth noticing that the generic transformation from single-input FE to decentralized MCFE for function-revealing inner products in [ABG19] cannot be extended to the function-hiding setting, as the transformation necessitates each client to know the entire joint inner-product function. Furthermore, the technique used to avoid the random oracle in LWE-based MCFE in [LT19] is inapplicable due to the impossibility of lattice-based FE for function-hiding inner products [Üna20].

Overall, as zero-sharing is treated as *black-box* in all existing FE constructions [ABG19, CDSG+20, AGT21b, SV23], updating shares into fresh ones may require a common reference string to simultaneously maintain both correlation and corruption-resistance. Therefore, to achieve security in the standard model, we provide a novel framework for zero-sharing construction.

Updatable Pseudorandom Zero-Sharing. In the construction from [KDK11], each zero share for a user i at a label ℓ is $\mathsf{share}_{i,\ell} = \sum_{j\in[n]\setminus i}(-1)^{i<j}\mathsf{PRF}_{k_{i,j}}(\ell)$. We parse each term $(-1)^{i<j}\mathsf{PRF}_{k_{i,j}}(\ell)$ that is generated with a pairwise PRF key $k_{i,j} = k_{j,i}$ as a pairwise seed. Then our approach can be described at a high level as follows: instead of randomizing each share using a random oracle, we randomize each of its pairwise seeds using another PRF, denoted PRF', which is computed using the same key of the corresponding pair.

More formally, we provide a definition and security models for the new notion Updatable Pseudorandom Zero-Sharing (UZS) in Section 3. This formalization will emphasize the properties needed in constructing DDFE schemes.

We provide a brief overview of our DDH-based construction in Figure 2, which eventually leads to the security in the standard model for pairing-based DDFE constructions. A natural extension to the MDDH assumption, which could be applied, for example in AWS-DDFE [ATY23] could be achieved using a similar construction strategy. When compared to the pseudorandom zero-sharing schemes in earlier works [KDK11, BIK+17, ABG19], it is worth noticing that zero shares in our scheme are group elements that sum up to the group identity, instead of scalars.

Algorithm	**Return**
$\mathsf{SetUp}(\lambda)$	a group $\mathbb{G}$ and pseudorandom functions $(\mathsf{PRF}, \mathsf{PRF}')$.
$\mathsf{KeyGen}()$	$\mathsf{sk}_i = (k_{i,j} = k_{j,i})_{j\in[n]\setminus[i]}$ for each party P_i.
$\mathsf{SeedGen}(\mathsf{sk}_i, \ell_s)$	$\mathsf{seed}_{i,\ell_s} = ([(-1)^{i<j}\mathsf{PRF}_{k_{i,j}}(\ell_s)])_{j\in[n]\setminus i}$.
$\mathsf{TokGen}(\mathsf{sk}_i, \ell_u)$	$\mathsf{token}_{i,\ell_u} = (\mathsf{PRF}'_{k_{i,j}}(\ell_u))_{j\in[n]\setminus i}$.
$\mathsf{SeedUpt}(\mathsf{seed}_{i,\ell_s}, \mathsf{token}_{i,\ell_u})$	$\mathsf{seed}_{i,\ell_s\|\|\ell_u} = ([(-1)^{i<j}\mathsf{PRF}_{k_{i,j}}(\ell_s)\cdot\mathsf{PRF}'_{k_{i,j}}(\ell_u)])_{j\in[n]\setminus i}$.
$\mathsf{ShareEval}(\mathsf{seed}_{i,\ell})$	$\mathsf{share}_{i,\ell} = \sum_{j\in[n]\setminus i}[c_{i,j,\ell}]$ where $\mathsf{seed}_{i,\ell} = ([c_{i,j,\ell}])_{j\in[n]\setminus i}$.

Fig. 2. Construction for UZS in DDH groups between n parties

We show how the scheme achieves correctness and security using matrix representation:

– The matrix $[\boldsymbol{A}_{\ell_s}]$ (see Figure 3) serves as a seeding matrix, with each i-th row representing seed_{i,ℓ_s}. This seeding matrix adopts an anti-symmetric form: $A_{\ell_s,(i,i)} = 0$ for all $i \in [n]$ and $A_{\ell_s,(i,j)} = -A_{\ell_s,(j,i)}$ for all $(i,j \in [n], i \neq j)$.
– The matrix $\boldsymbol{B}_{\ell_u}$ (see Figure 4) serves as an updating matrix, with each i-th row representing $\mathsf{token}_{i,\ell_u}$. This updating matrix adopts a symmetric form: $B_{\ell_u,(i,i)} = 0$ for all $i \in [n]$ and $B_{\ell_u,(i,j)} = B_{\ell_u,(j,i)}$ for all $(i,j \in [n], i \neq j)$.
– The matrix $[\boldsymbol{C}_{\ell_s\|\ell_u}]$ (see Figure 5) serves as an updated seeding matrix, with each i-th row representing $\mathsf{seed}_{i,\ell_s\|\ell_u}$. This updated matrix results from a Hadamard product between $[\boldsymbol{A}_{\ell_s}]$ and $\boldsymbol{B}_{\ell_u}$, which preserves the anti-symmetric form from $[\boldsymbol{A}_{\ell_s}]$.

$$[\boldsymbol{A}_{\ell_s}] = \left[\begin{array}{cccc} 0 & -\mathsf{PRF}_{k_{1,2}}(\ell_s) & \cdots & -\mathsf{PRF}_{k_{1,n}}(\ell_s) \\ \mathsf{PRF}_{k_{2,1}}(\ell_s) & 0 & \cdots & -\mathsf{PRF}_{k_{2,n}}(\ell_s) \\ \vdots & \vdots & \ddots & \vdots \\ \mathsf{PRF}_{k_{n,1}}(\ell_s) & \mathsf{PRF}_{k_{n,2}}(\ell_s) & \cdots & 0 \end{array}\right] \begin{array}{l} \to \mathsf{seed}_{1,\ell_s} = ([A_{\ell_s,(1,j)}])_{j\in[n]\setminus\{1\}} \\ \to \mathsf{seed}_{2,\ell_s} = ([A_{\ell_s,(2,j)}])_{j\in[n]\setminus\{2\}} \\ \\ \to \mathsf{seed}_{n,\ell_s} = ([A_{\ell_s,(n,j)}])_{j\in[n]\setminus\{n\}} \end{array}$$

Fig. 3. Seeding matrix $[\boldsymbol{A}_{\ell_s}] \in \mathbb{G}^{n\times n}$.

$$\boldsymbol{B}_{\ell_u} = \left(\begin{array}{cccc} 0 & \mathsf{PRF}'_{k_{1,2}}(\ell_u) & \cdots & \mathsf{PRF}'_{k_{1,n}}(\ell_u) \\ \mathsf{PRF}'_{k_{2,1}}(\ell_u) & 0 & \cdots & \mathsf{PRF}'_{k_{2,n}}(\ell_u) \\ \vdots & \vdots & \ddots & \vdots \\ \mathsf{PRF}'_{k_{n,1}}(\ell_u) & \mathsf{PRF}'_{k_{n,2}}(\ell_u) & \cdots & 0 \end{array}\right) \begin{array}{l} \to \mathsf{token}_{1,\ell_u} = (B_{\ell_u,(1,j)})_{j\in[n]} \\ \to \mathsf{token}_{2,\ell_u} = (B_{\ell_u,(2,j)})_{j\in[n]} \\ \\ \to \mathsf{token}_{n,\ell_u} = (B_{\ell_u,(n,j)})_{j\in[n]} \end{array}$$

Fig. 4. Updating matrix $\boldsymbol{B}_{\ell_s} \in \mathbb{Z}_p^{n\times n}$.

$$[\boldsymbol{C}_{\ell_s||\ell_u}] = [\boldsymbol{A}_{\ell_s} \odot \boldsymbol{B}_{\ell_u}] \overset{\mathsf{PRF},\mathsf{DDH}}{\approx}$$

$$\begin{array}{cccc} & & & \text{DDH holds for non-corrupted pairs } \{(i,j)\}_{i\neq 2, j\neq 2} \\ & & & \uparrow \end{array}$$

$$\left(\begin{array}{cccc} 0 & -[\mathsf{PRF}_{k_{1,2}}(\ell_s)\mathsf{PRF}'_{k_{1,2}}(\ell_u)] & \cdots & [-\mathsf{RF}_{(1,n)}(\ell_s||\ell_u)] \\ [\mathsf{PRF}_{k_{2,1}}(\ell_s)\mathsf{PRF}'_{k_{2,1}}(\ell_u)] & 0 & \cdots & -[\mathsf{PRF}_{k_{2,n}}(\ell_s)\mathsf{PRF}'_{k_{2,n}}(\ell_u)] \\ \vdots & \vdots & \ddots & \vdots \\ [\mathsf{RF}_{(1,n)}(\ell_s||\ell_u)] & [\mathsf{PRF}_{k_{n,2}}(\ell_s)\mathsf{PRF}'_{k_{n,2}}(\ell_u)] & \cdots & 0 \end{array}\right) \begin{array}{l} \\ \to \text{PRF keys } \{k_{2,i}\}_{i\in[n], i\neq 2} \text{ are corrupted} \\ \\ \\ \end{array}$$

$$\begin{array}{c} \downarrow \\ \text{PRF keys } \{k_{i,2}\}_{i\in[n], i\neq 2} \text{ are corrupted} \end{array}$$

Fig. 5. Indistinguishability for entries in the updated seeding matrix $[\boldsymbol{C}_{\ell_s||\ell_u}]$, where RF denotes a random function. For simple illustration, we assume only user P_2 is corrupted by the adversary.

In both cases, when $\ell = \ell_s$ or $\ell = \ell_s||\ell_u$, each $\mathsf{share}_{i,\ell}$ computes the sum of all entries on the i-th row of an *antisymmetric matrix*. Therefore, $\sum_{i\in[n]} \mathsf{share}_{i,\ell}$ computes the sum of all entries in the matrix, resulting in the group identity $[0]$. This implies the correctness of the scheme.

At a high level, the standard security of a pseudorandom zero-sharing scheme guarantees that when the adversary corrupts a subset of parties and computes their shares on its own, the shares of the remaining honest users are computationally indistinguishable from a correlated random distribution. However, this security is insufficient for the security proofs in DDFE schemes. Our scheme not only satisfies this standard security but also provides indistinguishability from a random distribution for the updated shares: even when the adversary has access to the seeding matrix $[\boldsymbol{A}_{\ell_s}]$ and thereby has access to all the shares of the honest users on ℓ_s, the indistinguishability still holds for the updated honest users' shares in $[\boldsymbol{C}_{\ell_s||\ell_u}]$.

The reason is that the product maintains the integrity of the honest entries from $[\boldsymbol{A}_{\ell_s}]$ and $\mathcal{B}_{\ell_u}$ to $[\boldsymbol{C}_{\ell_s||\ell_u}]$. We then have $[\mathsf{PRF}_{k_{i,j}}(\ell_s)\mathsf{PRF}'_{k_{i,j}}(\ell_u)] \overset{\mathsf{PRF},\mathsf{DDH}}{\approx} \mathsf{RF}_{(i,j)}(\ell_s||\ell_u)$ for each honest pair (i,j) (see Figure 5). This implies that the honest shares in $[\boldsymbol{C}_{\ell_s||\ell_u}]$ are independent from those in $[\boldsymbol{A}_{\ell_s}]$. Moreover, by using the Multi-DDH assumption (Definition 2), which tightly reduces to the DDH assumption using the random-self reducibility, the indistinguisability of the honest updated shares on $(\ell_s||\ell_u)$ holds for a polynomial number of labels ℓ_s given a label ℓ_u. Therefore, we obtain a security without RO that is applicable to DDFE schemes.

The cost of achieving DDFE schemes in the standard model will be an increase in ciphertext size per client by an additive factor of $O_\lambda(n)$. Each ciphertext now hides a row of the updating matrix, while each functional key hides a row of the seeding matrix. While this trade-off increases the ciphertext size, it does not compromise the completeness of our solution with respect to the open question on the feasibility of FH-IP-DDFE in the standard model, as posed in [SV23]. Moreover, it remains more efficient than the multiplicative overhead of $O_\lambda(n)$ in [ABG19] (see Figure 1). On the other hand, it is challenging to compress seeds and tokens, as the relation between updated shares does not hold anymore.

A concrete construction for the dynamic setting is provided in Section 3.3, which relies on a non-interactive key exchange protocol NIKE, a pseudorandom function PRF and the DDH assumption. Its security for a restricted setting of *one-time-update* and static corruption is provided in Theorem 1.

From UZS to DDFE Schemes Without RO. For each client $\mathsf{pk} \in \mathcal{U}$, the encryption of $\boldsymbol{x}_{\mathsf{pk}}$ under $(\mathcal{U}, \ell_M)$ and the generation of decryption key for $\boldsymbol{y}_{\mathsf{pk}}$ under $(\mathcal{U}, \ell_K)$ can be briefly described as follows:

$$\mathsf{ct}_{\mathsf{pk}} = \mathsf{IPE.Enc}_{(1^{\rho(\lambda,d)+|\mathcal{U}|},\mathsf{sk}_{\mathsf{pk}})}([\boldsymbol{x}_{\mathsf{pk}}, \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \boldsymbol{0}^{\rho(\lambda,d)}]_1);$$
$$\mathsf{dk}_{\mathsf{pk}} = \mathsf{IPE.DKGen}_{(1^{\rho(\lambda,d)+|\mathcal{U}|},\mathsf{sk}_{\mathsf{pk}})}([\boldsymbol{y}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, \boldsymbol{0}^{\rho(\lambda,d)}]_2)$$

Here, d is an inner-product dimension, $[\boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}]_1 = \mathsf{seed}^{\ell_K}_{\mathsf{pk},\mathcal{U}}$ and $\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}} = \mathsf{token}^{\ell_M}_{\mathsf{pk},\mathcal{U}}$ are generated using the UZS scheme. The IPE scheme is instantiated for inner products of dimension $(\rho(\lambda,d)+|\mathcal{U}|)$ where $\rho(\lambda,d)$ is polynomial.

The correctness of the scheme holds since

$$\begin{aligned}\mathsf{IPE.Dec}(\mathsf{ct}_{\mathsf{pk}}, \mathsf{dk}_{\mathsf{pk}}) &= [\boldsymbol{x}^\top_{\mathsf{pk}} \cdot \boldsymbol{y}_{\mathsf{pk}} + {\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}]_T \\ &= [\boldsymbol{x}^\top_{\mathsf{pk}} \cdot \boldsymbol{y}_{\mathsf{pk}}]_T + e([1]_1, \mathsf{share}^{\ell_K||\ell_M}_{\mathsf{pk},\mathcal{U}})\end{aligned}$$

by the bilinear-update property of the UZS scheme.

For the security proof, we use a sequence of hybrid games in Figure 9 and Figure 10, with the goal of gradually replacing $\boldsymbol{x}^b$ with $\boldsymbol{x}^0$ in every ciphertext under each message label ℓ_M. Eventually, we remove $\boldsymbol{y}^b$ and leave $\boldsymbol{y}^0$ in every honest functional key to fully eliminate the challenge bit b. The security of UZS applies through a formal reduction in $\mathbf{G}^\star_{\ell_M.3}$ (see Figure 10). It is important to emphasize that this strategy aims to achieve security for adaptive message and key queries, which differs from the strategies for selective ones used in previous works [AGT21b, SV23].

Using a similar approach with the UZS scheme, we also construct a selectively secure AWS-DDFE scheme in the standard model, as detailed in Section 5. However, since the technique for preserving adaptive security applies only to FH-IP-DDFE, we leave as an open question whether it is possible to achieve adaptively secure AWS-DDFE from any adaptively secure AWS/IP-FE scheme.

Adaptive Security Preserving. In [AGT21b, SV23], the constructions of FH-IP-MCFE that use FH-IPFE as a black-box achieve security for only selective message and key queries. For simplicity in illustrating the core ideas, we assume that under each label, there is at most one queried message and one queried key[2]. The primary obstacle to achieving adaptive security lies in a specific step within the hybrid games, where the challenger must encode both a zero share $R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}}$ and the product of the challenge message and key ${\boldsymbol{x}^b_{\mathsf{pk},\ell_M}}^\top \cdot \boldsymbol{y}^b_{\mathsf{pk},\ell_K}$ in the same slot. This forms a sum ${\boldsymbol{x}^b_{\mathsf{pk},\ell_M}}^\top \cdot \boldsymbol{y}^b_{\mathsf{pk},\ell_K} + R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}}$, which, due to the randomness of the zero share and the admissibility conditions, is indistinguishable from ${\boldsymbol{x}^0_{\mathsf{pk},\ell_M}}^\top \cdot \boldsymbol{y}^0_{\mathsf{pk},\ell_K} + R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}}$. From here, a sequence of symmetric games is applied to $\boldsymbol{x}^0_{\mathsf{pk},\ell_M}$ and $\boldsymbol{y}^0_{\mathsf{pk},\ell_K}$. Complexity leveraging is not possible in this case, as guessing key or message queries would result in exponential security loss, while the indistinguishability between the games before and after encoding that sum is guaranteed only by a computational assumption. To overcome this obstacle, the adaptively-secure construction in [SV23] is based on a specific FH-IPFE scheme to leverage the underlying computational assumption. In another related work [Tom19], Tomida presented an MIFE construction for FH-IP that preserves adaptive security when derived from any FH-IPFE, but it remains unclear whether there exists an extension from this MIFE construction to DDFE.

Our approach to address this issue is that for each argument on ℓ_M, we introduce intermediate one-time pads $\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}$ and $\boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}}$ to mask the message-containing slots in ciphertexts as $(\boldsymbol{x}^b_{\mathsf{pk},\ell_M} + \boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}})$. The resulting offset, $-(\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}} \cdot \boldsymbol{y}^b_{\mathsf{pk},\ell_K} + \boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}} \cdot \boldsymbol{y}^0_{\mathsf{pk},\ell_K})$, is encoded and randomized using the zero share $R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}}$ in the functional keys. We also introduce the Injection Lemma (see Lemma 2), which considers the randomness of both the one-time pads and the zero share, alongside the admissibility conditions, to enable a perfectly indistinguishable transition from $(\boldsymbol{x}^b_{\mathsf{pk},\ell_M} + \boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}})$ to $(\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \boldsymbol{x}^0_{\mathsf{pk},\ell_M} + \boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}})$. From this point, a sequence of symmetric games is applied to $\boldsymbol{x}^0_{\mathsf{pk},\ell_M}$ and $\boldsymbol{y}^0_{\mathsf{pk},\ell_K}$. By using these intermediate one-time pads, we overcome the challenge of having to encode $\boldsymbol{x}^b_{\mathsf{pk},\ell_M}$ and $\boldsymbol{y}^b_{\mathsf{pk},\ell_K}$ in the same slot, thereby obtaining adaptive security for FH-IP-DDFE from any adaptively secure FH-IPFE.

2 Preliminaries

We defer the definitions of arithmetic branching programs, pseudorandom functions, non-interactive key exchange, and all-or-nothing encapsulation in Appendix A.

2.1 Notations

Given any $n \in \mathbb{N}$, we denote by $[n]$ the set of integers $\{1, ..., n\}$. Given any set $\mathcal{A}$, $\mathcal{L}(\mathcal{A})$ will denote the set of finite lists of elements of $\mathcal{A}$, and $\mathcal{S}(\mathcal{A})$ will denote the set of finite subsets of $\mathcal{A}$. While both lists and sets are ordered by default, lists may contain repeated elements. We denote by $|\mathcal{A}|$ the cardinal of any finite set $\mathcal{A}$. For any vector $\boldsymbol{a} \in \mathbb{A}^n$, we denote by a_i the i-th component of $\boldsymbol{a}$. Similarly, for any matrix $\boldsymbol{A} \in \mathbb{A}^{m\times n}$, we denote by $A_{i,j}$ the component at the position (i,j) of $\boldsymbol{A}$.

We also use the following notation for condition-based function-selecting functions

$$[f_0/f_1]^{\mathsf{con}}(\mathsf{inp}) = \begin{cases} \mathsf{out} \leftarrow f_0(\mathsf{inp}) & \text{if } \mathsf{con} = 0; \\ \mathsf{out} \leftarrow f_1(\mathsf{inp}) & \text{if } \mathsf{con} = 1. \end{cases}$$

2.2 Prime Order Group.

Let GGen be a prime-order group generator, a probabilistic polynomial time (PPT) algorithm that on input the security parameter 1^λ returns a description $\mathcal{G} = (\mathbb{G}, p, P)$ of an additive cyclic group $\mathbb{G}$ of order p for a 2λ-bit prime p, whose generator is P. For $a \in \mathbb{Z}_p$, define $[a] = aP \in \mathbb{G}$ as the *implicit representation* of a in $\mathbb{G}$.

From a random element $[a] \in \mathbb{G}$, it is computationally hard to compute the value a (the discrete logarithm problem). Given $[a], [b] \in \mathbb{G}$ and a scalar $x \in \mathbb{Z}_p$, one can efficiently compute $[ax] \in \mathbb{G}$ and $[a + b] = [a] + [b] \in G$.

[2] A complete analysis considering multiple queries is provided in Section 4.

Definition 1 (Decisional Diffie-Hellman Assumption). *The Decisional Diffie-Hellman Assumption states that, for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\mathsf{Adv}_{\mathit{DDH}}(\mathcal{A}) := \left| \mathrm{P}\left[\mathcal{A}(\mathcal{G}, \mathcal{D}_b) = b \;\middle|\; \begin{array}{l} b \xleftarrow{\$} \{0,1\}, \mathcal{G} \xleftarrow{\$} \mathsf{GGen}(1^\lambda) \\ a, r, s \xleftarrow{\$} \mathbb{Z}_p, d_0 = ar, d_1 = s \\ \mathcal{D}_b = ([a], [r], [d_b]) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

Definition 2 (m-Multi DDH Assumption [CDG+18]). *For all $\lambda \in \mathbb{N}$ and for every PPT adversary $\mathcal{A}$ running within time t, then*

$$\mathsf{Adv}_{m\text{-}\mathit{DDH}}(\mathcal{A}, t) := \left| \mathrm{P}\left[\mathcal{A}(\mathcal{G}, \mathcal{D}_b) = b \;\middle|\; \begin{array}{l} b \xleftarrow{\$} \{0,1\}, \mathcal{G} \xleftarrow{\$} \mathsf{GGen}(1^\lambda) \\ X, Y_j, Z_j \xleftarrow{\$} \mathbb{G}\ \forall j \in [m] \\ \mathcal{D}_0 = (X, (Y_j, \mathit{CDH}(X, Y_j))_{j=1}^m) \\ \mathcal{D}_1 = (X, (Y_j, Z_j)_{j=1}^m) \end{array} \right] - \frac{1}{2} \right|$$

is bounded by $\mathsf{Adv}_{\mathit{DDH}}(\mathcal{A}, t + 4m \times t_{\mathbb{G}})$, where $t_{\mathbb{G}}$ is the time for an exponentiation in $\mathbb{G}$.

2.3 Pairing Group.

Let PGGen be a pairing group generator, a PPT algorithm that on input the security parameter 1^λ returns a description $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, P_1, P_2, e)$ of asymmetric pairing groups where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are additive cyclic groups of order p for a 2λ-bit prime p, P_1 and P_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable (non-degenerate) bilinear group elements. For $s \in \{1, 2, T\}$ and $a \in \mathbb{Z}_p$, we define $[a]_s = aP_s \in \mathbb{G}_s$ as the implicit representation of a in $\mathbb{G}_s$, and then for any $0 < B < \frac{p}{2}$, we define $\mathbb{G}_s^{[-B,B]} = \{[a] \in \mathbb{G}_s : a \in [-B, B]\}$. Given $[a]_1$, $[b]_2$, one can efficiently compute $[ab]_T$ using the pairing e.

Definition 3 (Symmetric eXternal Diffie-Hellman Assumption). *The Symmetric eXternal Diffie-Hellman (SXDH) Assumption states that, in a pairing group $\mathcal{PG} \xleftarrow{\$} \mathsf{PGen}(1^\lambda)$, the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

2.4 Dynamic Decentralized Functional Encryption

Definition 4 (Dynamic Decentralized Functional Encryption). *A dynamic decentralized functional encryption scheme over a set of public keys $\mathcal{PK}$ for functionality $F \colon \mathcal{L}(\mathcal{PK} \times \mathcal{K}) \times \mathcal{L}(\mathcal{PK} \times \mathcal{M}) \to \{0,1\}^*$ consists of five algorithms:*

- $\mathsf{SetUp}(1^\lambda)$*: On input a parameter 1^λ, it generates and outputs public parameters pp. Those parameters are implicit arguments to all the other algorithms.*
- $\mathsf{KeyGen}()$*: It generates and outputs a party's public key $\mathsf{pk} \in \mathcal{PK}$ and the corresponding secret key $\mathsf{sk}_{\mathsf{pk}}$.*
- $\mathsf{Enc}(\mathsf{sk}_{\mathsf{pk}}, m)$*: On input a party's secret key $\mathsf{sk}_{\mathsf{pk}}$, a value $m \in \mathcal{M}$ to encrypt, it outputs a ciphertext $\mathsf{ct}_{\mathsf{pk},m}$.*
- $\mathsf{DKGen}(\mathsf{sk}_{\mathsf{pk}}, k)$*: On input a party's secret key $\mathsf{sk}_{\mathsf{pk}}$, a key space object k, it outputs a functional decryption key $\mathsf{dk}_{\mathsf{pk},k}$.*
- $\mathsf{Dec}((\mathsf{dk}_{\mathsf{pk},k_{\mathsf{pk}}})_{\mathsf{pk} \in \mathcal{U}_K}, (\mathsf{ct}_{\mathsf{pk},m_{\mathsf{pk}}})_{\mathsf{pk} \in \mathcal{U}_M})$*: On input a list of functional decryption keys $(\mathsf{dk}_{\mathsf{pk},k_{\mathsf{pk}}})_{\mathsf{pk} \in \mathcal{U}_K}$, a finite list of ciphertexts $(\mathsf{ct}_{\mathsf{pk},m_{\mathsf{pk}}})_{\mathsf{pk} \in \mathcal{U}_\mathcal{M}}$, where $\mathcal{U}_\mathcal{M}, \mathcal{U}_K \in \mathcal{L}(\mathcal{PK})$ are the lists of senders and receivers respectively, it outputs a value $y \in \{0,1\}^*$.*

Correctness. *For all parameters $\lambda \in \mathbb{N}$, all polynomial size lists $\mathcal{U}_M, \mathcal{U}_K \in \mathcal{L}(\mathcal{PK})$, $(\mathsf{pk}, k_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_K} \in \mathcal{L}(\mathcal{PK} \times \mathcal{K})$ and $(\mathsf{pk}, m_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_M} \in \mathcal{L}(\mathcal{PK} \times \mathcal{M})$, it holds that*

$$\Pr\left[\mathsf{Dec}((\mathsf{dk}_{\mathsf{pk},k_{\mathsf{pk}}})_{\mathsf{pk} \in \mathcal{U}_K}, (\mathsf{ct}_{\mathsf{pk},m_{\mathsf{pk}}})_{\mathsf{pk} \in \mathcal{U}_M}) = F((\mathsf{pk}, k_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_K}, (\mathsf{pk}, m_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_M})\right] = 1,$$

where the probability is taken over all algorithms.

In this work, we assume that each user is identified by a public key pk, which it can generate on its own with the (unique) associated secret key, using KeyGen. Anyone can thus dynamically join the system, by publishing its public key.

Remark 1 (Empty lists). As in [CDSG+20], we denote by ϵ_K the empty list in $\mathcal{L}(\mathcal{PK} \times \mathcal{K})$, indicating that there is no key required. Furthermore, ϵ_M denotes the empty list in $\mathcal{L}(\mathcal{PK} \times \mathcal{M})$, indicating that there is no message required.

Definition 5 (Security for DDFE). *For* $\mathrm{xx} \in \{\text{sel-sta}, \text{sta}, \text{any}\}$, $\mathrm{yy} \in \{\text{sym}, \text{asym}\}$, *a* xx-yy-IND *security game of* DDFE *for every PPT adversary* $\mathcal{A}$ *is defined with access to the oracles* QNewHon, QEnc, QDKGen *and* QCor *described below:*

- *Initialize: the challenger runs the setup algorithm* $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$ *and chooses a random bit* $b \xleftarrow{\$} \{0,1\}$. *It provides* pp *to the adversary* $\mathcal{A}$*;*
- *Participation creation queries* $\mathsf{QNewHon}()$*: it generates* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) \leftarrow \mathsf{KeyGen}()$*, stores the association* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}})$ *and returns* pk *to the adversary;*
- *Challenge message queries* $\mathsf{QEnc}(\mathsf{pk}, m^0, m^1)$*: it outputs the ciphertext* $\mathsf{ct}_m \leftarrow (\mathsf{sk}, m^b)$ *where* sk *is associated with* pk*. If* pk *is not associated with any secret key, nothing is returned;*
- *Challenge key queries* $\mathsf{QDKGen}(\mathsf{pk}, k^0, k^1)$*: it outputs the decryption key* $\mathsf{dk}_k \leftarrow \mathsf{DKGen}(\mathsf{sk}, k^b)$ *where* sk *is associated with* pk*. If* pk *is not associated with any secret key, nothing is returned;*
- *Corruption queries* $\mathsf{QCor}(\mathsf{pk})$*: it outputs the secret key* sk *associated to* pk*. If* pk *is not associated with any secret key, nothing is returned;*
- *Finalize:* $\mathcal{A}$ *provides its guess* b' *on the bit* b*, and this procedure outputs the result* β *according to the admissibility condition given below.*

Let $\mathcal{PK}$ *be the set of parties on which* $\mathsf{QNewHon}()$ *is queried,* $\mathcal{C} \subset \mathcal{PK}$ *be the set of corrupted parties,* $\mathcal{H} = \mathcal{PK} \setminus \mathcal{C}$ *be the set of honest (non-corrupted) participants at the end of the game. Finalize outputs the bit* $\beta = (b' = b)$ *if the Condition* $(*)$ *is satisfied, otherwise Finalize outputs* $\beta \xleftarrow{\$} \{0,1\}$*. Condition* $(*)$ *holds if all the following conditions hold:*

- *there do not exist two lists of messages* $(\boldsymbol{m}^0 = (\mathsf{pk}, m^0_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_M}, \boldsymbol{m}^1 = (\mathsf{pk}, m^1_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_M})$*, including* (ϵ_M, ϵ_M)*, and two lists of keys* $(\boldsymbol{k}^0 = (\mathsf{pk}, k^0_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_K}, \boldsymbol{k}^1 = (\mathsf{pk}, k^1_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_K})$*, including* (ϵ_K, ϵ_K)*, such that*
  - $F(\boldsymbol{k}^0, \boldsymbol{m}^0) \neq F(\boldsymbol{k}^1, \boldsymbol{m}^1)$*;*
  - $\forall \mathsf{pk} \in \mathcal{U}_M$*,* $[\mathsf{QEnc}(\mathsf{pk}, m^0_{\mathsf{pk}}, m^1_{\mathsf{pk}})$ *was made and* $\mathsf{pk} \in \mathcal{H}]$ *or* $[m^0_{\mathsf{pk}} = m^1_{\mathsf{pk}} \in \mathcal{M}$ *and* $\mathsf{pk} \in \mathcal{C}]$*;*
  - $\forall \mathsf{pk} \in \mathcal{U}_K$*,* $[\mathsf{QDKGen}(\mathsf{pk}, k^0_{\mathsf{pk}}, k^1_{\mathsf{pk}})$ *was made and* $\mathsf{pk} \in \mathcal{H}]$ *or* $[k^0_{\mathsf{pk}} = k^1_{\mathsf{pk}} \in \mathcal{K}$ *and* $\mathsf{pk} \in \mathcal{C}]$*.*
- *when* $\mathrm{xx} = \text{sel-sta}$*: the adversary sends all its* $\mathsf{QNewHon}()$ *queries in one shot. After that it sends in one shot all* $\mathsf{QEnc}(\mathsf{pk}, m^0, m^1)$*,* $\mathsf{QDKGen}(\mathsf{pk}, k^0, k^1)$ *and* $\mathsf{QCor}(\mathsf{pk})$ *queries*[3]*;*
- *when* $\mathrm{xx} = \text{sta}$*: the adversary sends all its* $\mathsf{QNewHon}()$ *queries in one shot. After that it sends in one shot all* $\mathsf{QCor}(\mathsf{pk})$ *queries only;*
- *when* $\mathrm{yy} = \text{sym}$*: for* $\mathsf{pk} \in \mathcal{C}$*, the* $\mathsf{QDKGen}(\mathsf{pk}, k^0_{\mathsf{pk}}, k^1_{\mathsf{pk}})$ *and* $\mathsf{QEnc}(\mathsf{pk}, m^0_{\mathsf{pk}}, m^1_{\mathsf{pk}})$ *queries must satisfy* $k^0_{\mathsf{pk}} = k^1_{\mathsf{pk}}$ *and* $m^0_{\mathsf{pk}} = m^1_{\mathsf{pk}}$*, respectively.*[4]

We say DDFE *is* xx-yy-IND*-secure if given any parameter* $\lambda \in \mathbb{N}$*, for every PPT adversary* $\mathcal{A}$*, the following holds*

$$\mathsf{Adv}^{\text{xx-yy}}_{\mathsf{DDFE}}(\mathcal{A}) := \left| \Pr[\beta = 1] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

We also have sta-sym-IND*-security implies* sel-sta-sym-IND*-security.*

Definition 6 (Function-Hiding Inner-Product DDFE). *A function-hiding* IP-DDFE *scheme over a pairing group* $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p)$ *is defined for a dimension* $d \in \mathbb{N}$*, a message bound* $X < \frac{p}{2}$*, a function bound* $Y < \frac{p}{2}$*, and label spaces* $(\mathcal{L}_M, \mathcal{L}_K)$ *as follows:*

$$\mathcal{K} = \left(\mathbb{G}_2^{[-Y,Y]}\right)^d \times \mathcal{S}(\mathcal{PK}) \times \mathcal{L}_K;$$

$$\mathcal{M} = \left(\mathbb{G}_1^{[-X,X]}\right)^d \times \mathcal{S}(\mathcal{PK}) \times \mathcal{L}_M.$$

[3] This setting is equivalent to the selective setting defined in [CDSG+20].

[4] The symmetric setting is a natural restriction for FH-IP-DDFE, as knowing $\mathsf{sk}_{\mathsf{pk}}$ from corruption queries allows any adversary to generate itself valid ciphertexts (or decryption keys) to learn the challenge decryption keys (or ciphertexts) from inner-products.

Then, one has

$$F(\epsilon_K, (\mathsf{pk}, [\boldsymbol{x}_{\mathsf{pk}}]_1, \mathcal{U}_{\mathsf{pk}}, \ell_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}) = (\mathcal{U}_{\mathsf{pk}}, \ell_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}};$$
$$F((\mathsf{pk}, ([\boldsymbol{y}_{\mathsf{pk}}]_2, \mathcal{U}_{\mathsf{pk}}, \ell_{\mathsf{pk}}))_{\mathsf{pk}\in\mathcal{U}}, \epsilon_M) = (\mathcal{U}_{\mathsf{pk}}, \ell_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}$$

for any $\mathcal{U} \in \mathcal{L}(\mathcal{PK})$ *and*

$$F((\mathsf{pk}, k_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_K}, (\mathsf{pk}, m_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M}) = \begin{cases} \sum_{\mathsf{pk}\in\mathcal{U}_K} \boldsymbol{x}_{\mathsf{pk}}^\top \cdot \boldsymbol{y}_{\mathsf{pk}} & \text{if condition } (*) \\ \perp & \text{otherwise.} \end{cases}$$

FH-IP-DDFE *condition* $(*)$ *is:*

– $\mathcal{U}_K = \mathcal{U}_M = \mathcal{U}$;
– $\exists \ell_K \in \mathcal{L}_K$, $\forall \mathsf{pk} \in \mathcal{U}$, $k_{\mathsf{pk}} = ([\boldsymbol{y}_{\mathsf{pk}}]_2, \mathcal{U}, \ell_K)$;
– $\exists \ell_M \in \mathcal{L}_M$, $\forall \mathsf{pk} \in \mathcal{U}$, $m_{\mathsf{pk}} = ([\boldsymbol{x}_{\mathsf{pk}}]_1, \mathcal{U}, \ell_M)$.

Remark 2 (Single-Input Inner-Product). We denote by IPE a single-input FE for function-hiding inner products, that was defined as in Definition 6 for the case $|\mathcal{PK}| = 1$.

Definition 7 (Attribute-Weighted-Sum DDFE). *A function-revealing* AWS-DDFE *scheme over a pairing group* $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p)$ *is defined for the class of arithmetic branching programs* $\mathcal{F}^{\mathsf{ABP}}_{n_0,n_1}$ *and a label space* $\mathcal{L}_M$ *as follows:*

$$\mathcal{K} = \left\{ \boldsymbol{f} := (\mathsf{pk}, f_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_K} \text{ where } f_{\mathsf{pk}} \in \mathcal{F}^{\mathsf{ABP}}_{n_0,n_1} \text{ and } \mathcal{U}_K \in \mathcal{S}(\mathcal{PK}) \right\}$$
$$\mathcal{M} = \bigcup_{i\in\mathbb{N}} (\mathbb{Z}_p^{n_0} \times \mathbb{Z}_p^{n_1})_i \times \mathcal{S}(\mathcal{PK}) \times \mathcal{L}_M.$$

Then, one has

$$F(\epsilon_K, (\mathsf{pk}, (\boldsymbol{x}_{\mathsf{pk},j}, \boldsymbol{z}_{\mathsf{pk},j})_{j\in[N_{\mathsf{pk}}]}, \mathcal{U}_{\mathsf{pk}}, \ell_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}) = ((\boldsymbol{x}_{\mathsf{pk},j})_{j\in[N_{\mathsf{pk}}]}, \mathcal{U}_{\mathsf{pk}}, \ell_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}};$$
$$F((\mathsf{pk}, (\boldsymbol{f}_{\mathsf{pk}}, \mathcal{U}_{\mathsf{pk}}))_{\mathsf{pk}\in\mathcal{U}}, \epsilon_M) = (\boldsymbol{f}_{\mathsf{pk}}, \mathcal{U}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}$$

for any $\mathcal{U} \in \mathcal{L}(\mathcal{PK})$ *and*

$$F((\mathsf{pk}, k_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_K}, (\mathsf{pk}, m_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M}) = \begin{cases} [\sum_{\mathsf{pk}\in\mathcal{U}_K} \sum_{j\in[N_{\mathsf{pk}}]} f_{\mathsf{pk}}(\boldsymbol{x}_{\mathsf{pk},j})^\top \boldsymbol{z}_{\mathsf{pk},j}]_T & \text{if } (*) \\ \perp & \text{otherwise.} \end{cases}$$

AWS-DDFE *condition* $(*)$ *is:*

– $\mathcal{U}_K = \mathcal{U}_M = \mathcal{U}$;
– $\exists \boldsymbol{f} \in (\mathsf{pk}, \mathcal{F}^{\mathsf{ABP}}_{n_0,n_1})_{\mathsf{pk}\in\mathcal{U}}$, $\forall \mathsf{pk}' \in \mathcal{U}$, $k_{\mathsf{pk}'} = (\boldsymbol{f}, \mathcal{U})$;
– $\exists \ell_M \in \mathcal{L}_M$, $\forall \mathsf{pk} \in \mathcal{U}$, $m_{\mathsf{pk}} = ((\boldsymbol{x}_{\mathsf{pk},j}, \boldsymbol{z}_{\mathsf{pk},j})_{j\in[N_{\mathsf{pk}}]}, \mathcal{U}, \ell_M)$.

Definition 8 (Single-Input FE for AWSw/IP). *Consider the case* $|\mathcal{PK}| = 1$ *in Definition 4 for the single-input setting, an* FE *for attribute-weighted sums with function-hiding inner product scheme over* $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p)$ *is defined for the class of arithmetic branching programs* $\mathcal{F}^{\mathsf{ABP}}_{n_0,n_1}$ *as follows:*

$$\mathcal{K} = \mathcal{F}^{\mathsf{ABP}}_{n_0,n_1} \times \mathbb{G}_2^d$$
$$\mathcal{M} = \bigcup_{i\in\mathbb{N}} (\mathbb{Z}_p^{n_0} \times \mathbb{Z}_p^{n_1})_i \times \mathbb{G}_1^d.$$

Then, one has

$$F(\epsilon_K, ((\boldsymbol{x}_j, \boldsymbol{z}_j)_{j\in[N]}, [\boldsymbol{s}]_1)) = ((\boldsymbol{x}_j)_{j\in[N]});$$
$$F((f, [\boldsymbol{t}]_2), \epsilon_M) = f;$$

and

$$F((f, [\boldsymbol{t}]_2), ((\boldsymbol{x}_j, \boldsymbol{z}_j)_{j\in[N]}, [\boldsymbol{s}]_1)) = [\sum_{j\in[N]} f(\boldsymbol{x}_j)^\top \boldsymbol{z}_j + \boldsymbol{s}^\top \boldsymbol{t}]_T.$$

We note that a single-input FE construction for AWSw/IP in the standard model is provided in [ATY23].

3 Updatable Pseudorandom Zero Sharing

In this section, we provide a definition and a security model for Updatable Pseudorandom Zero-Sharing, which serves as the key building block for later pairing-based DDFE constructions. A scheme in DDH groups that supports the bilinear update property will also be provided.

3.1 Definition

Definition 9 (Updatable Pseudorandom Zero Sharing). *Given a set of users $\mathcal{PK}$, a seeding label space $\mathcal{L}_S$ and an updating label space $\mathcal{L}_U$, an updatable pseudorandom zero sharing scheme* UZS *over a group* $(\mathbb{A}, +)$ *consists of six algorithms:*

- $\mathsf{SetUp}(\lambda)$*: On input a security parameter* λ*, it outputs public parameters* pp*. Those parameters are implicit arguments to all the other algorithms.*
- $\mathsf{KeyGen}()$*: It outputs a party's public key* $\mathsf{pk} \in \mathcal{PK}$ *and the corresponding secret key* $\mathsf{sk}_{\mathsf{pk}}$*.*
- $\mathsf{SeedGen}(\mathsf{sk}_{\mathsf{pk}}, \mathcal{U}, \ell_s)$*: On input a secret key* $\mathsf{sk}_{\mathsf{pk}}$*, a user set* $\mathcal{U} \in \mathcal{S}(\mathcal{PK})$*, and a seeding label* $\ell_s \in \mathcal{L}_S$*, it outputs a seed* $\mathsf{seed}^{\ell_s}_{\mathsf{pk},\mathcal{U}}$ *if* $\mathsf{pk} \in \mathcal{U}$*. Otherwise, it outputs* $\perp$*.*
- $\mathsf{TokGen}(\mathsf{sk}_{\mathsf{pk}}, \mathcal{U}, \ell_u)$*: On input a secret key* $\mathsf{sk}_{\mathsf{pk}}$*, a user set* $\mathcal{U} \in \mathcal{S}(\mathcal{PK})$*, and an updating label* $\ell_u \in \mathcal{L}_U$*, it outputs a token* $\mathsf{token}^{\ell_u}_{\mathsf{pk},\mathcal{U}}$ *if* $\mathsf{pk} \in \mathcal{U}$*. Otherwise, it outputs* $\perp$*.*
- $\mathsf{SeedUpt}(\mathsf{seed}^{\ell_s}_{\mathsf{pk},\mathcal{U}}, \mathsf{token}^{\ell_u}_{\mathsf{pk},\mathcal{U}})$*: On input a seed* $\mathsf{seed}^{\ell_s}_{\mathsf{pk},\mathcal{U}}$ *and a token* $\mathsf{token}^{\ell_u}_{\mathsf{pk},\mathcal{U}}$*, it outputs a seed* $\mathsf{seed}^{\ell_s||\ell_u}_{\mathsf{pk},\mathcal{U}}$*.*
- $\mathsf{ShareEval}(\mathsf{seed}^{\ell}_{\mathsf{pk},\mathcal{U}})$*: On input a seed* $\mathsf{seed}^{\ell}_{\mathsf{pk},\mathcal{U}}$ *for* $\ell \in \mathcal{L}_S \cup (\mathcal{L}_S \times \mathcal{L}_U)$*, it outputs a share* $\mathsf{share}^{\ell}_{\mathsf{pk},\mathcal{U}}$*.*

Correctness. *For any security parameter* $\lambda \in \mathbb{N}$*, any* $\ell_s \in \mathcal{L}_S$*, any* $\ell_u \in \mathcal{L}_U$*, any* $\mathcal{U} \in \mathcal{S}(\mathcal{PK})$*, then it holds that*

$$\Pr\left[\sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{share}^{\ell_s}_{\mathsf{pk},\mathcal{U}} = \sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{share}^{\ell_s||\ell_u}_{\mathsf{pk},\mathcal{U}} = 0_{\mathbb{A}}\right] = 1$$

where the probability is taken over all algorithms.

Definition 10 (Correlated Pseudorandomness for UZS**).** *For* $\mathrm{xx} \in \{\mathrm{otu}, \mathrm{any}\}$, $\mathrm{yy} \in \{\mathrm{sta}, \mathrm{adt}\}$, *a* xx-yy-IND *security game of* UZS *for every PPT adversary* $\mathcal{A}$ *is defined with access to the oracles* $\mathsf{QNewHon}, \mathsf{QCor}, \mathsf{QTokGen}, \mathsf{QSeedGen}$, *and* QShare *described below:*

- *Initialize: the challenger runs the setup algorithm* $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$ *and chooses a random bit* $b \overset{\$}{\leftarrow} \{0,1\}$*. It initializes the sets* $\mathcal{H}$ *and* $\mathcal{C}$ *of honest participants and corrupted participants respectively to* $\emptyset$*, and provides* pp *to the adversary* $\mathcal{A}$*;*
- *Participant creation queries* $\mathsf{QNewHon}()$*: it generates* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) \leftarrow \mathsf{KeyGen}()$ *to simulate a new participant, stores the association* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}})$ *in the set* $\mathcal{H}$*, and returns* pk *to the adversary;*
- *Corruption queries* $\mathsf{QCor}(\mathsf{pk})$*: it moves the association* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}})$ *from* $\mathcal{H}$ *to* $\mathcal{C}$ *and returns the secret key* sk*. If* pk *is not associated with any secret key in* $\mathcal{H}$*, then nothing is returned;*
- *Seed generation queries* $\mathsf{QSeedGen}(\mathsf{pk}, \mathcal{U}, \ell_s)$*: if one of conditions* $[\mathsf{pk} \notin \mathcal{U}]$ *or* $[\ell_s \notin \mathcal{L}_S]$ *or* $[\mathsf{pk}$ *is not associated with any secret key in either* $\mathcal{H}$ *or* $\mathcal{C}]$*, then nothing is returned. Otherwise it returns* $\mathsf{seed}^{\ell_s}_{\mathsf{pk},\mathcal{U}} \leftarrow \mathsf{SeedGen}(\mathsf{sk}, \mathcal{U}, \ell_s)$ *to the adversary.*
- *Token generation queries* $\mathsf{QTokGen}(\mathsf{pk}, \mathcal{U}, \ell_u)$*: if one of conditions* $[\mathsf{pk} \notin \mathcal{U}]$ *or* $[\ell_u \notin \mathcal{L}_U]$ *or* $[\mathsf{pk}$ *is not associated with any secret key in either* $\mathcal{H}$ *or* $\mathcal{C}]$*, nothing is returned. Otherwise it returns* $\mathsf{token}^{\ell_u}_{\mathsf{pk},\mathcal{U}} \leftarrow \mathsf{TokGen}(\mathsf{sk}, \mathcal{U}, \ell_u)$ *to the adversary.*
- *Challenge share queries* $\mathsf{QShare}(\mathcal{U}, \ell)$*: if* $[\ell \notin \mathcal{L}_S \cup (\mathcal{L}_S \times \mathcal{L}_U)]$ *or* $[|\mathcal{H} \cap \mathcal{U}| = 0]$*, nothing is returned. We define the following share distributions:*
  - *A pseudorandom share generation algorithm* $\mathsf{PShareGen}(\mathcal{S}, \mathcal{U}, \ell)$*: it outputs* $\perp$ *if* $\mathcal{S} \not\subset \mathcal{U}$*, otherwise the challenger generates* $\mathsf{seed}^{\ell}_{\mathsf{pk},\mathcal{U}}$ *as follows:*
    - *if* $\ell = \ell_s \in \mathcal{L}_S$*:* $\mathsf{seed}^{\ell_s}_{\mathsf{pk},\mathcal{U}} = \mathsf{QSeedGen}(\mathsf{pk}, \mathcal{U}, \ell_s)$*;*
    - *if* $\ell = \ell_s||\ell_u \in \mathcal{L}_S \times \mathcal{L}_U$*:* $\mathsf{seed}^{\ell}_{\mathsf{pk},\mathcal{U}} = \mathsf{SeedUpt}(\mathsf{seed}^{\ell_s}_{\mathsf{pk},\mathcal{U}}, \mathsf{token}^{\ell_u}_{\mathsf{pk},\mathcal{U}})$

      $$\text{where} \begin{cases} \mathsf{seed}^{\ell_s}_{\mathsf{pk},\mathcal{U}} = \mathsf{QSeedGen}(\mathsf{pk}, \mathcal{U}, \ell_s) \\ \mathsf{token}^{\ell_u}_{\mathsf{pk},\mathcal{U}} = \mathsf{QTokGen}(\mathsf{pk}, \mathcal{U}, \ell_u) \end{cases}$$

    and outputs $\mathsf{share}^{\ell}_{\mathsf{pk},\mathcal{U}} \leftarrow \mathsf{ShareEval}(\mathsf{seed}^{\ell}_{\mathsf{pk},\mathcal{U}})$ *for all* $\mathsf{pk} \in \mathcal{S}$*.*

- *A uniformly correlated random distribution for any* $(\mathcal{S}, \mathcal{U}, \ell)$ *where* $\mathcal{S} \subset \mathcal{U}$

$$\mathcal{R}^{\ell}_{\mathcal{S},\mathcal{U}} := \left\{ (s_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{S}} \xleftarrow{\$} \mathbb{A}^{|\mathcal{S}|} \mid \sum_{\mathsf{pk} \in \mathcal{S}} s_{\mathsf{pk}} = - \sum_{\mathsf{pk} \in \mathcal{U} \setminus \mathcal{S}} t_{\mathsf{pk}} \right\}$$

where $(t_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U} \setminus \mathcal{S}} = \mathsf{PShareGen}(\mathcal{U} \setminus \mathcal{S}, \mathcal{U}, \ell)$.

Then,

- *if* $b = 0$*, the challenger returns* $(\mathsf{share}^{\ell}_{\mathsf{pk},\mathcal{U}})_{\mathsf{pk} \in \mathcal{H} \cap \mathcal{U}} \leftarrow \mathsf{PShareGen}(\mathcal{H} \cap \mathcal{U}, \mathcal{U}, \ell)$ *to the adversary;*
- *if* $b = 1$*, the challenger returns* $(\mathsf{share}^{\ell}_{\mathsf{pk},\mathcal{U}})_{\mathsf{pk} \in \mathcal{H} \cap \mathcal{U}} \leftarrow \mathcal{R}^{\ell}_{\mathcal{H} \cap \mathcal{U}, \mathcal{U}}$ *to the adversary;*

– *Finalize:* $\mathcal{A}$ *provides its guess* b' *on the bit* b*, and this procedure outputs the result* β *of the game, according to the admissibility condition below which aims at preventing trivial wins.*

Let $\mathcal{H}$ *and* $\mathcal{C}$ *be the sets of honest users and corrupted users at the end of the game respectively. Finalize outputs the bit* $\beta = (b' = b)$ *if the Condition* $(*)$ *is satisfied, otherwise Finalize outputs* $\beta \xleftarrow{\$} \{0, 1\}$*. Condition* $(*)$ *holds if all the following conditions hold:*

– *there are no* $\mathsf{QCor}(\mathsf{pk})$ *queries after a* $\mathsf{QShare}(\mathcal{U}, \ell)$ *query was made;*
– *there does not exist* $\ell_s \in \mathcal{L}_S$ *such that a* $\mathsf{QShare}(\mathcal{U}, \ell_s)$ *query and a* $\mathsf{QSeedGen}(\mathsf{pk}, \ell_s)$ *query are sent;*
– *there does not exist* $(\ell_s, \ell_u) \in \mathcal{L}_S \times \mathcal{L}_U$ *such that a* $\mathsf{QShare}(\mathcal{U}, \ell_s || \ell_u)$ *query and a* $\mathsf{QTokGen}(\mathsf{pk}, \ell_u)$ *query are sent;*
– *when* $\mathrm{xx} = \mathrm{otu}$*: for any* $\mathcal{U}$*, the queries of the form* $\mathsf{QShare}(\mathcal{U}, \ell_s || \cdot)$ *can be sent for only one* $\ell_u \in \mathcal{L}_U$*.*
– *when* $\mathrm{yy} = \mathrm{sta}$*: the adversary sends all its* $\mathsf{QNewHon}()$ *queries in one shot. After that it sends in one shot all* $\mathsf{QCor}(\mathsf{pk})$ *queries.*

We say UZS *is* xx-yy-IND*-secure if given any parameter* $\lambda \in \mathbb{N}$*, for every PPT adversary* $\mathcal{A}$*, the following holds*

$$\mathsf{Adv}^{\text{xx-yy}}_{\mathsf{UZS}}(\mathcal{A}) = \left| \Pr[\beta = 1] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

The security model described above captures the security of a standard pseudorandom zero-sharing scheme: the adversary has full access to the oracles and can win the game by only distinguishing the pseudorandom non-updated shares from the correlated random ones. Additionally, Condition $(*)$ does not prevent the adversary from querying the seeds that result in the updated challenge shares. This intuitively means that even when the adversary has access to the seeds of honest users, the resulting updated shares remain indistinguishable from a correlated random distribution.

On the other hand, extending a standard pseudorandom zero-sharing scheme over $\mathcal{L} = \mathcal{L}_S \times \mathcal{L}_U$ to an updatable one can be achieved by using its share generation algorithm to create new shares on concatenated labels of the form $(\ell_s || \ell_u)$. However, this approach may require operations that are more complex than those allowed in pairing groups. Therefore, we specify a more pairing-friendly property for the updating algorithm of an UZS scheme.

Definition 11 (Bilinear Update). *An updatable pseudorandom zero sharing scheme* UZS *of security parameter* λ *is said to satisfy the bilinear update property if each seed is of the form* $[\boldsymbol{a}] \in \mathbb{A}^{\rho(\lambda)}$*, each token is of the form* $\boldsymbol{b} \in \mathbb{Z}^{\rho(\lambda)}$ *where* ρ *is a* λ*-dependent parameter, and for any* $\mathsf{pk} \in \mathcal{PK}$*, any* $\ell_s \in \mathcal{L}_S$*, and any* $\ell_u \in \mathcal{L}_U$*, it holds that*

$$[\boldsymbol{a}^{\ell_s}_{\mathsf{pk},\mathcal{U}}] \odot_\lambda \boldsymbol{b}^{\ell_u}_{\mathsf{pk},\mathcal{U}} = \mathsf{SeedUpt}([\boldsymbol{a}^{\ell_s}_{\mathsf{pk},\mathcal{U}}], \boldsymbol{b}^{\ell_u}_{\mathsf{pk},\mathcal{U}}),$$

where $\odot_\rho : \mathbb{A}^\rho \times \mathbb{Z}^\rho \to \mathbb{A}^\rho$ *is an entry-wise bilinear map.*

3.2 Construction in DDH Groups

Let $\mathsf{PRF} : \{0,1\}^* \to \mathbb{Z}_p$ be a pseudorandom function, NIKE be a non-interactive key exchange protocol, $\mathcal{L}_S$ be a seeding label space, and $\mathcal{L}_U$ be an updating label space. A construction of UZS is described in Figure 6. We emphasize that zero shares in the scheme are group elements that sum up to the group identity.

Construction:

- $\mathsf{SetUp}(1^\lambda)$: It generates $\mathcal{G} \leftarrow \mathsf{GGen}(1^\lambda)$ and $\mathsf{NIKE.pp} \leftarrow \mathsf{NIKE.SetUp}(1^\lambda)$ and returns
$$\mathsf{pp} = (\mathcal{G}, \mathsf{NIKE.pp}, \mathsf{PRF}, \mathcal{L}_S, \mathcal{L}_U).$$
The parameters pp are implicit to other algorithms.
- $\mathsf{KeyGen}()$: Each user samples
  - NIKE keys $(\mathsf{pk}, \mathsf{NIKE.sk}_{\mathsf{pk}}) \leftarrow \mathsf{NIKE.KeyGen}()$;
  - a shared key $k_{\mathsf{pk},\mathsf{pk}'} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{pk}', \mathsf{NIKE.sk}_{\mathsf{pk}})$ for each published $\mathsf{pk}' \in \mathcal{PK} \setminus \{\mathsf{pk}\}$.

  It returns
$$(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) = \left(\mathsf{pk}, (\mathsf{NIKE.sk}_{\mathsf{pk}}, (k_{\mathsf{pk},\mathsf{pk}'})_{\mathsf{pk}' \in \mathcal{PK}\setminus\mathsf{pk}})\right).$$
- $\mathsf{SeedGen}(\mathsf{sk}_{\mathsf{pk}}, (\mathcal{U}, \ell_s))$: It computes
$$\boldsymbol{a}^{\ell_s}_{\mathsf{pk},\mathcal{U}} = \left((-1)^{\mathsf{pk}<\mathsf{pk}'}\mathsf{PRF}_{k_{\mathsf{pk},\mathsf{pk}'}}(\text{"}s\text{"}||\mathcal{U}||\ell_s)\right)_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}},$$
and returns $\mathsf{seed}^{\ell_s}_{\mathsf{pk},\mathcal{U}} = [\boldsymbol{a}^{\ell_s}_{\mathsf{pk},\mathcal{U}}]$. If $\mathsf{pk} \notin \mathcal{U}$, it returns $\perp$.
- $\mathsf{TokGen}(\mathsf{sk}_{\mathsf{pk}}, (\mathcal{U}, \ell_u))$: It computes
$$\boldsymbol{b}^{\ell_u}_{\mathsf{pk},\mathcal{U}} = \left(\mathsf{PRF}_{k_{\mathsf{pk},\mathsf{pk}'}}(\text{"}u\text{"}||\mathcal{U}||\ell_u)\right)_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}},$$
and returns $\mathsf{token}^{\ell_u}_{\mathsf{pk},\mathcal{U}} = \boldsymbol{b}^{\ell_u}_{\mathsf{pk},\mathcal{U}}$. If $\mathsf{pk} \notin \mathcal{U}$, it returns $\perp$.
- $\mathsf{SeedUpt}(\mathsf{seed}^{\ell_s}_{\mathsf{pk},\mathcal{U}}, \mathsf{token}^{\ell_u}_{\mathsf{pk},\mathcal{U}})$: It parses
  - $\mathsf{seed}^{\ell_s}_{\mathsf{pk},\mathcal{U}} = [\boldsymbol{a}^{\ell_s}_{\mathsf{pk},\mathcal{U}}]$,
  - $\mathsf{token}^{\ell_u}_{\mathsf{pk},\mathcal{U}} = \boldsymbol{b}^{\ell_u}_{\mathsf{pk},\mathcal{U}}$,
  - $\rho = |\mathcal{U}| - 1$;

  and returns
$$\mathsf{seed}^{\ell_s||\ell_u}_{\mathsf{pk},\mathcal{U}} = [\boldsymbol{a}^{\ell_s}_{\mathsf{pk},\mathcal{U}}] \odot_\rho \boldsymbol{b}^{\ell_u}_{\mathsf{pk},\mathcal{U}}.$$
- $\mathsf{ShareEval}(\mathsf{seed}^{\ell}_{\mathsf{pk},\mathcal{U}})$: It parses $\rho = |\mathcal{U}| - 1$, $\mathsf{seed}^{\ell}_{\mathsf{pk},\mathcal{U}} = [\boldsymbol{a}^{\ell}_{\mathsf{pk},\mathcal{U}}] \in \mathbb{G}^\rho$, and returns
$$\mathsf{share}^{\ell}_{\mathsf{pk},\mathcal{U}} = \sum_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}} [a^{\ell}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}].$$

Fig. 6. Updatable Pseudorandom Zero-Sharing in DDH groups.

Correctness. Given $\lambda \in \mathbb{N}$, $\mathsf{pp} \leftarrow \mathsf{SetUp}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) \leftarrow \mathsf{KeyGen}()$ $\forall \mathsf{pk} \in \mathcal{PK}$, $\mathcal{U} \in \mathcal{S}(\mathcal{PK})$, any labels $\ell_s \in \mathcal{L}_S$ and $\ell_u \in \mathcal{L}_U$, from the above scheme, one has

$$\begin{aligned}
\sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{share}^{\ell_s}_{\mathsf{pk},\mathcal{U}} &= \sum_{\mathsf{pk}\in\mathcal{U}} \sum_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}} [a^{\ell_s}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}] \\
&= \sum_{\mathsf{pk}\in\mathcal{U}} \sum_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}} [(-1)^{\mathsf{pk}<\mathsf{pk}'} c_{\mathsf{pk},\mathsf{pk}'}] \\
&= \sum_{\substack{(\mathsf{pk},\mathsf{pk}')\in\mathcal{U}^2 \\ \mathsf{pk}\neq\mathsf{pk}'}} [(-1)^{\mathsf{pk}<\mathsf{pk}'} c_{\mathsf{pk},\mathsf{pk}'} + (-1)^{\mathsf{pk}'<\mathsf{pk}} c_{\mathsf{pk}',\mathsf{pk}}] \\
&= \sum_{\substack{(\mathsf{pk},\mathsf{pk}')\in\mathcal{U}^2 \\ \mathsf{pk}\neq\mathsf{pk}'}} [0] = [0].
\end{aligned}$$

where $c_{\mathsf{pk},\mathsf{pk}'} := \mathsf{PRF}_{k_{\mathsf{pk},\mathsf{pk}'}}(\text{"}s\text{"}||\mathcal{U}||\ell_s)$ and $c_{\mathsf{pk}',\mathsf{pk}} = c_{\mathsf{pk},\mathsf{pk}'}$. Similarly, one has

$$\begin{aligned}
\sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{share}^{\ell_s||\ell_u}_{\mathsf{pk},\mathcal{U}} &= \sum_{\mathsf{pk}\in\mathcal{U}} \sum_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}} [a^{\ell_s}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'} \cdot b^{\ell_u}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}] \\
&= \sum_{\mathsf{pk}\in\mathcal{U}} \sum_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}} [(-1)^{\mathsf{pk}<\mathsf{pk}'} c'_{\mathsf{pk},\mathsf{pk}'}] \\
&= \sum_{\substack{(\mathsf{pk},\mathsf{pk}')\in\mathcal{U}^2 \\ \mathsf{pk}\neq\mathsf{pk}'}} [0] = [0].
\end{aligned}$$

where $c'_{\mathsf{pk},\mathsf{pk}'} := \mathsf{PRF}_{k_{\mathsf{pk},\mathsf{pk}'}}(\text{"}s\text{"}||\mathcal{U}||\ell_s) \cdot \mathsf{PRF}_{k_{\mathsf{pk},\mathsf{pk}'}}(\text{"}u\text{"}||\mathcal{U}||\ell_u)$ and $c'_{\mathsf{pk}',\mathsf{pk}} = c'_{\mathsf{pk},\mathsf{pk}'}$. □

Remark 3 (Bilinear Update). The UZS scheme in the above construction satisfies the bilinear update property in Definition 11.

3.3 Security Analysis

Theorem 1 (Indistinguishability for UZS). *If* NIKE *is a* IND*-secure non-interactive key exchange protocol, and the DDH assumption holds in* $\mathbb{G}$*, then the* UZS *scheme constructed in Section 3.2 is* otu-sta-IND *secure (as in Definition 10) in the standard model.*

Proof. In the static corruption game, we can fix $\mathcal{PK}$ to be the set of parties generated by $\mathsf{QNewHon}()$, $\mathcal{C}$ to be the set of corrupted parties and $\mathcal{H} = \mathcal{PK} \setminus \mathcal{C}$ to be the set of honest parties. Let q_{QNewHon} and q_{QShare} be the number of QNewHon and QShare queries respectively. We consider two cases separately: $|\mathcal{H}| < 2$ and $|\mathcal{H}| \geq 2$.

The case of $|\mathcal{H}| < 2$. There is no information about b in this case: the output of $\mathsf{QShare}(\mathcal{U}, \ell)$ does not return $\perp$ only if $|\mathcal{H} \cap \mathcal{U}| = 1$, then the distribution $\mathcal{R}^{\ell}_{\mathcal{S},\mathcal{U}}$ contains a single value to assign to the honest share. This value is equal to $\mathsf{PShareGen}(\mathcal{H} \cap \mathcal{U}, \mathcal{U}, \ell)$.

The case of $|\mathcal{H}| \geq 2$. We proceed via a hybrid argument by using the games described in Figure 12. In this argument, the game $\mathbf{G}_0$ corresponds to the otu-sta-IND security game as defined in Definition 10, and the game $\mathbf{G}_3$ corresponds to the case where the adversary's advantage is 0 since there is no challenge bit b. Given $\lambda \in \mathbb{N}$, we denote by Adv_i the advantage of a PPT adversary $\mathcal{A}$ running in time t in each game $\mathbf{G}_i$, and $\mathsf{Adv}_{\mathrm{xx}}$ be the best advantage of any PPT adversary running in time t against the primitive xx that is setup with λ.

Game $\mathbf{G}_1$: The change is that for each $(\mathsf{pk}, \mathsf{pk}') \in \mathcal{H}^2$, the challenger uses uniformly random shared keys $k_{\mathsf{pk},\mathsf{pk}'}$ in answering to $\mathsf{QSeedGen}(\mathsf{pk}^\star, \mathcal{U}, \ell_u)$, $\mathsf{QTokGen}(\mathsf{pk}^\star, \mathcal{U}, \ell_s)$ queries for $\mathsf{pk}^\star \in \{\mathsf{pk}, \mathsf{pk}'\}$ (then in $\mathsf{QShare}(\mathcal{U}, \ell)$ queries). The indistinguishability is implied by the security of the non-interactive key exchange protocol, given in Lemma 5.

Game $\mathbf{G}_2$: The change is that for each $(\mathsf{pk}, \mathsf{pk}') \in \mathcal{H}^2$, the challenger uses a random function $\mathsf{RF}_{\mathsf{pk},\mathsf{pk}'} = \mathsf{RF}_{\mathsf{pk}',\mathsf{pk}}$ instead of $\mathsf{PRF}_{k_{\mathsf{pk},\mathsf{pk}'}}$ in generating answers to the queries $\mathsf{QSeedGen}(\mathsf{pk}^\star, \mathcal{U}, \ell_u)$ and $\mathsf{QTokGen}(\mathsf{pk}^\star, \mathcal{U}, \ell_s)$ for $\mathsf{pk}^\star \in \{\mathsf{pk}, \mathsf{pk}'\}$, (then in $\mathsf{QShare}(\mathcal{U}, \ell)$ queries). The indistinguishability is implied by the security of the pseudorandom functions, given in Lemma 6.

Game $\mathbf{G}_3$: The change is that for each $\mathsf{QShare}(\mathcal{U}, \ell)$ query, the challenger answers independently from b by sampling $(\mathsf{share}_{\mathsf{pk},\ell})_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} \overset{\$}{\leftarrow} \mathcal{R}^{\ell}_{\mathcal{H}\cap\mathcal{U},\mathcal{U}}$, where the distribution $\mathcal{R}^{\ell}_{\mathcal{H}\cap\mathcal{U},\mathcal{U}}$ is defined in the QShare oracle in Definition 10. The indistinguishability is implied by the DDH assumption, given in Lemma 1.

From the transitions above, one completes the theorem. □

We show the main proof technique that lies in the following lemma.

Lemma 1 **(UZS: Transition from $\mathbf{G}_2$ to $\mathbf{G}_3$).** *For any PPT adversary $\mathcal{A}$, the advantage in distinguishing two games is*

$$|\mathsf{Adv}_2 - \mathsf{Adv}_3| \leq \frac{1}{2} q_{\mathsf{QNewHon}}(q_{\mathsf{QNewHon}} - 1) q_{\mathsf{QShare}} \cdot \mathsf{Adv}_{\mathsf{DDH}}(t + 4q_{\mathsf{QShare}} \times t_{\mathbb{G}}).$$

Proof. For every $\mathcal{U}$ queried in the form of $\mathsf{QShare}(\mathcal{U}, \cdot)$ such that $|\mathcal{H} \cap \mathcal{U}| \geq 2$, we use multiple hybrid games that go all over the pairs in $\mathcal{P}_{\mathcal{U}} = \{(\mathsf{pk}, \mathsf{pk}') := (\mathsf{pk}', \mathsf{pk}) \in (\mathcal{H}\cap\mathcal{U})^2, \mathsf{pk} \neq \mathsf{pk}'\}$. We assume that all pairs in $\mathcal{P}_{\mathcal{U}}$ are bijectively mapped by κ to $[q_{\mathcal{U}}]$ for some integer $q_{\mathcal{U}} \geq 1$. For $i \in [q_{\mathcal{U}}]$, we define the following sequence of games as follows:

Game $\mathbf{G}_{2.\mathcal{U}.i}$: In this game, for any $\mathsf{QShare}(\mathcal{U}, \ell_s || \ell_u^\star)$ query[5], when $b = 0$, for all $\mathsf{pk} \in \mathcal{H} \cap \mathcal{U}$, the challenger computes

$$\begin{aligned}
[\boldsymbol{a}^{\ell_s}_{\mathsf{pk},\mathcal{U}}] &= \mathsf{QSeedGen}(\mathsf{pk}, \mathcal{U}, \ell_s),\\
\boldsymbol{b}^{\ell_u^\star}_{\mathsf{pk},\mathcal{U}} &= \mathsf{QTokGen}(\mathsf{pk}, \mathcal{U}, \ell_u^\star),\\
\mathsf{seed}^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U}} &= [\boldsymbol{a}^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U}}]
\end{aligned}$$

where

$$\begin{cases}
a^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'} \overset{\$}{\leftarrow} \mathbb{Z}_p & \text{if } (\mathsf{pk},\mathsf{pk}') \in \mathcal{P}_{\mathcal{U}} \text{ and } \kappa(\mathsf{pk},\mathsf{pk}') < i\\
\boxed{a^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'} \overset{\$}{\leftarrow} \mathbb{Z}_p} & \text{if } (\mathsf{pk},\mathsf{pk}') \in \mathcal{P}_{\mathcal{U}} \text{ and } \kappa(\mathsf{pk},\mathsf{pk}') = i\\
a^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'} = a^{\ell_s}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'} \cdot b^{\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'} & \text{if } (\mathsf{pk},\mathsf{pk}') \in \mathcal{P}_{\mathcal{U}} \text{ and } \kappa(\mathsf{pk},\mathsf{pk}') > i\\
a^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'} = a^{\ell_s}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'} \cdot b^{\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'} & \text{if } (\mathsf{pk},\mathsf{pk}') \notin \mathcal{P}_{\mathcal{U}}
\end{cases}$$

and outputs $\mathsf{share}^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U}} = \mathsf{ShareEval}(\mathsf{seed}^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U}})$. The change between $\mathbf{G}_{2.i-1}$ and $\mathbf{G}_{2.i}$ is in the $\mathsf{PShareGen}$ algorithm (defined by the QShare oracle in Definition 10) and is highlighted in gray.

For every $\mathcal{U}$, we assume that $\mathbf{G}_{2.\mathcal{U}.0}$ is the game where $\mathsf{QShare}(\mathcal{U}, \cdot)$ is the same as in $\mathbf{G}_2$. For each transition from $\mathbf{G}_{2.\mathcal{U}.i-1}$ to $\mathbf{G}_{2.\mathcal{U}.i}$ for $i \in [q_{\mathcal{U}}]$, we build an adversary $\mathcal{B}$ against the multi-DDH assumption (Definition 2), which can be described as follows:

- To answer $\mathsf{QSeedGen}(\mathsf{pk}, \mathcal{U}, \ell_s)$ and $\mathsf{QSeedGen}(\mathsf{pk}', \mathcal{U}, \ell_s)$ queries, the adversary $\mathcal{B}$ implicitly uses $[c^{\ell_s}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}] := Y_{\ell_s} \overset{\$}{\leftarrow} \mathbb{G}$ from the multi-DDH instance when $\kappa(\mathsf{pk},\mathsf{pk}') = i$ to compute $[a^{\ell_s}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}] = [(-1)^{\mathsf{pk}<\mathsf{pk}'} c^{\ell_s}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}]$ and $[a^{\ell_s}_{\mathsf{pk}',\mathcal{U},\mathsf{pk}}] = [(-1)^{\mathsf{pk}'<\mathsf{pk}} c^{\ell_s}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}]$ respectively.
- To answer $\mathsf{QShare}(\mathcal{U}, \ell_s||\ell_u^\star)$ queries, if $b = 0$, $\mathcal{B}$ implicitly uses $[c^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}] := Z_{\ell_s}$ from the multi-DDH instance when $\kappa(\mathsf{pk},\mathsf{pk}') = i$ to compute $[a^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}] = [(-1)^{\mathsf{pk}<\mathsf{pk}'} c^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}]$ and $[a^{\ell_s||\ell_u^\star}_{\mathsf{pk}',\mathcal{U},\mathsf{pk}}] = [(-1)^{\mathsf{pk}'<\mathsf{pk}} c^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}]$ respectively.
- $\mathcal{B}$ outputs $\mathcal{A}$'s guess for the challenge bit $\mathsf{DDH}^{\mathsf{multi}}.b$.

The adversary $\mathcal{B}$ has a complete multi-DDH challenge $\mathcal{D} = (X, (Y_{\ell_s})_{\ell_s}, (Z_{\ell_s})_{\ell_s})$ where X can be implicitly considered as $[b^{\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}] \overset{\$}{\leftarrow} \mathbb{G}$.

- When $\mathsf{DDH}^{\mathsf{multi}}.b = 0$, one has $[c^{\ell_s||\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}] = Z_{\ell_s} = \mathsf{CDH}(X, Y_{\ell_s}) = [c^{\ell_s}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}] \cdot b^{\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}$, which corresponds to $\mathbf{G}_{2.\mathcal{U}.i-1}$.

[5] In the one-time-update setting, for each $\mathcal{U}$, ℓ_s can change while $\ell_u^\star$ is fixed.

– When $\mathsf{DDH}^{\mathsf{multi}}.b = 1$, one has $[c^{\ell_s\|\ell_u^\star}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}] = Z_{\ell_s} \xleftarrow{\$} \mathbb{G}$, which corresponds to $\mathbf{G}_{2.\mathcal{U}.i}$.

Therefore, the computational gap between each $\mathbf{G}_{2.\mathcal{U}.i-1}$ and $\mathbf{G}_{2.\mathcal{U}.i}$ happens only when $b = 0$ and is then bounded by $\frac{1}{2} \cdot \mathsf{Adv}_{\mathsf{DDH}}(\lambda, t + 4q_{\mathsf{QShare}} \cdot t_{\mathbb{G}})$.

The last step is to show that for the last $\mathcal{U}^\star$ queried to $\mathsf{QShare}(\cdot,\cdot)$, one has $\mathbf{G}_{2.\mathcal{U}^\star.q_{\mathcal{U}^\star}} = \mathbf{G}_3$. It suffices to describe the case $b = 0$. We note that $\mathsf{QShare}(\mathcal{U},\ell) = (\mathsf{share}^{\ell}_{\mathsf{pk},\mathcal{U}})_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$ where $\mathsf{share}^{\ell}_{\mathsf{pk},\mathcal{U}} = \sum_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}}[a^{\ell}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}]$. By all transitions until $\mathbf{G}_{2.\mathcal{U}^\star.q_{\mathcal{U}^\star}}$, we have $a^{\ell}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'} \xleftarrow{\$} \mathbb{Z}_p$ for any pair of honest users $(\mathsf{pk},\mathsf{pk}')$, any set $\mathcal{U}$, any label ℓ. As $a^{\ell}_{\mathsf{pk}',\mathcal{U},\mathsf{pk}} = -a^{\ell}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}$, then

$$\left(\mathsf{share}^{0,\ell}_{\mathsf{pk},\mathcal{H}\cap\mathcal{U}} := \sum_{\mathsf{pk}'\in\mathcal{H}\cap\mathcal{U}\setminus\{\mathsf{pk}\}} a^{\ell}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'}\right)_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$$

are uniformly random shares of zero among users in $\mathcal{H}\cap\mathcal{U}$. Therefore, one has

$$\left(\mathsf{share}^{\ell}_{\mathsf{pk},\mathcal{U}} := \sum_{\mathsf{pk}'\in\mathcal{C}\cap\mathcal{U}} a^{\ell}_{\mathsf{pk},\mathcal{U},\mathsf{pk}'} + \mathsf{share}^{0,\ell}_{\mathsf{pk},\mathcal{H}\cap\mathcal{U}}\right)_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$$

are uniformly random shares of -$\sum_{\mathsf{pk}\in\mathcal{C}\cap\mathcal{U}} \mathsf{share}^{\ell}_{\mathsf{pk},\mathcal{U}}$, which is identical to the distribution $\mathcal{R}^{\ell}_{\mathcal{H}\cap\mathcal{U}}$. Since the number of sets $\mathcal{U}$ queried to $\mathsf{QShare}(\cdot,\ell)$ for $\ell \in \mathcal{L}_S \times \mathcal{L}_U$ is bounded by q_{QShare}, and each $q_{\mathcal{U}}$ is bounded by $\binom{q_{\mathsf{QNewHon}}-q_{\mathsf{QCor}}}{2}$, one obtains

$$|\mathsf{Adv}_2 - \mathsf{Adv}_3| \le \frac{1}{2}\binom{q_{\mathsf{QNewHon}} - q_{\mathsf{QCor}}}{2} q_{\mathsf{QShare}} \cdot \mathsf{Adv}_{\mathsf{DDH}}(\lambda, t + 4q_{\mathsf{QShare}} \times t_{\mathbb{G}}).$$

□

4 Function-Hiding Inner-Product DDFE

In this section, we construct a DDFE scheme for function-hiding inner products. As an additional independent contribution, we present a lemma that supports transforming an IPE scheme into a DDFE scheme in a black-box manner while preserving the adaptive security of the original scheme. By simultaneously applying the UZS scheme and this lemma to construction, we achieve an FH-IP-DDFE scheme that is adaptively secure in the standard model.

4.1 Lemma for Adaptive Security

We first state a remark from the admissibility for FH-IP-DDFE, which will be extremely useful for proofs of lemmas and theorems in our DDFE construction.

Remark 4 (Invariance in FH-IP-DDFE). From DDFE security model (Definition 5) and FH-IP functionality (Definition 6), given any admissible adversary against FH-IP-DDFE and any challenge bit b, Condition $(*)$ implies that among encryption queries under $(\ell_M,\mathcal{U})$ and decryption-key queries under $(\ell_K,\mathcal{U})$, the equalities

$$\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}\setminus\mathsf{pk}^\star} {\boldsymbol{x}^{b}_{\mathsf{pk}}}^{\top} \cdot \boldsymbol{y}^{b}_{\mathsf{pk}} + {\boldsymbol{x}^{b,\tau_M}_{\mathsf{pk}^\star}}^{\top} \cdot \boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}^\star} = \sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}\setminus\mathsf{pk}^\star} {\boldsymbol{x}^{0}_{\mathsf{pk}}}^{\top} \cdot \boldsymbol{y}^{0}_{\mathsf{pk}} + {\boldsymbol{x}^{0,\tau_M}_{\mathsf{pk}^\star}}^{\top} \cdot \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}^\star}$$

holds for each $\mathsf{pk}^\star \in \mathcal{H}\cap\mathcal{U}$ and each pair of index (τ_M,τ_K) that numerates repetitions of encryption queries under $(\mathsf{pk}^\star,\ell_M,\mathcal{U})$ and repetitions of decryption-key queries under $(\mathsf{pk}^\star,\ell_K,\mathcal{U})$ respectively. Then one has the value

$$\Delta^{b}_{\mathsf{pk}^\star} := {\boldsymbol{x}^{0,1}_{\mathsf{pk}^\star}}^{\top} \cdot \boldsymbol{y}^{0,1}_{\mathsf{pk}^\star} - {\boldsymbol{x}^{b,1}_{\mathsf{pk}^\star}}^{\top} \cdot \boldsymbol{y}^{b,1}_{\mathsf{pk}^\star} = {\boldsymbol{x}^{0,\tau_M}_{\mathsf{pk}^\star}}^{\top} \cdot \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}^\star} - {\boldsymbol{x}^{b,\tau_M}_{\mathsf{pk}^\star}}^{\top} \cdot \boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}^\star}$$

be an invariance across (τ_M,τ_K). Furthermore, each variance is a share of zero, i.e. $\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} \Delta^{b}_{\mathsf{pk}} = 0$.

The supporting lemma for the adaptive security of DDFE is shown below. At a high level, this lemma provides an information-theoretically secure masking technique for messages in DDFE.

Lemma 2 ($\mathsf{FH\text{-}IP\text{-}DDFE}$: **Injection Lemma**)**.** *Considering the security game of DDFE (in Definition 5) for function-hiding inner products, given any challenge bit b, any user set $\mathcal{U}$, any message tuple $\boldsymbol{q}_{\ell_M} = (\boldsymbol{x}^0_{\mathsf{pk}}, \boldsymbol{x}^1_{\mathsf{pk}}, \mathcal{U}, \ell_M)_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$ and any $\mathcal{Q}_K$-set of key tuples $\{\boldsymbol{q}_{\ell_K} = (\boldsymbol{y}^{0,\tau_K}_{\mathsf{pk},\ell_K}, \boldsymbol{y}^{1,\tau_K}_{\mathsf{pk},\ell_K}, \mathcal{U}, \ell_K)_{\tau_K\in[\mathsf{rep}_{\ell_K}],\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}\}_{\ell_K\in\mathcal{Q}_K}$ that come from admissible queries with respect to Condition $(*)$, then the outputs of experiments $\mathsf{Exp}^{\alpha}_{\mathrm{INJ}}(b, \mathcal{U}, \boldsymbol{q}_{\ell_M}, \{\boldsymbol{q}_{\ell_K}\}_{\ell_K\in\mathcal{Q}_K})$ for $\alpha \in \{0,1\}$ defined in Figure 7 are perfectly indistinguishable.*

$\underline{\mathsf{Exp}^{\alpha}_{\mathrm{INJ}}(b, \mathcal{U}, \boldsymbol{q}_{\ell_M}, \{\boldsymbol{q}_{\ell_K}\}_{\ell_K\in\mathcal{Q}_K})\text{:}}$

1. It generates: $\forall \mathsf{pk} \in \mathcal{H}\cap\mathcal{U}$,
   - $\boldsymbol{u}_{\mathsf{pk}}, \boldsymbol{v}_{\mathsf{pk}} \xleftarrow{\$} \mathbb{Z}_p^d$;
   - $R^{\ell_K}_{\mathsf{pk}} \xleftarrow{\$} \mathbb{Z}_p$ subjected to $\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} R^{\ell_K}_{\mathsf{pk}} = 0$ for each $\ell_K \in \mathcal{Q}_K$.
2. It computes: $\forall \ell_K \in \mathcal{Q}_K$,
   - $z^{\tau_K}_{\mathsf{pk},\ell_K} = (\boldsymbol{u}_{\mathsf{pk}}^\top \cdot \boldsymbol{y}^{b,\tau_K}_{\mathsf{pk},\ell_K} + \boldsymbol{v}_{\mathsf{pk}}^\top \cdot \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk},\ell_K})$ for each $\tau_K \in [\mathsf{rep}_{\ell_K}]$.
3. Output:
   - if $\alpha = 0$, it returns
   $$(\boldsymbol{u}_{\mathsf{pk}}, \boldsymbol{v}_{\mathsf{pk}}, \{(R^{\ell_K}_{\mathsf{pk}} + z^{\tau_K}_{\mathsf{pk},\ell_K})_{\tau_K\in[\mathsf{rep}_{\ell_K}]}\}_{\ell_K\in\mathcal{Q}_K})_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}};$$
   - if $\alpha = 1$, it returns
   $$(-\boldsymbol{x}^b_{\mathsf{pk}} + \boldsymbol{u}_{\mathsf{pk}}, \boldsymbol{x}^0_{\mathsf{pk}} + \boldsymbol{v}_{\mathsf{pk}}, \{(R^{\ell_K}_{\mathsf{pk}} + z^{\tau_K}_{\mathsf{pk},\ell_K})_{\tau_K\in[\mathsf{rep}_{\ell_K}]}\}_{\ell_K\in\mathcal{Q}_K})_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}.$$

Fig. 7. Experiment description in Injection Lemma. The injected elements are highlighted in blue.

Proof. From Remark 4, given each pk and ℓ_K, one has an invariance $\Delta^b_{\mathsf{pk},\ell_K}$ that is equal to $({\boldsymbol{x}^0}^\top_{\mathsf{pk}} \cdot \boldsymbol{y}^{\tau_K,0}_{\mathsf{pk},\ell_K} - {\boldsymbol{x}^b}^\top_{\mathsf{pk}} \cdot \boldsymbol{y}^{\tau_K,b}_{\mathsf{pk},\ell_K})$ for every $\tau_K \in [\mathsf{rep}_{\ell_K}]$. In the case $\alpha = 0$, new variables are set as follows

$$\boldsymbol{u}'_{\mathsf{pk}} := \boldsymbol{x}^b_{\mathsf{pk}} + \boldsymbol{u}_{\mathsf{pk}}; \qquad \boldsymbol{v}'_{\mathsf{pk}} := -\boldsymbol{x}^0_{\mathsf{pk}} + \boldsymbol{v}_{\mathsf{pk}}; \qquad R'^{\ell_K}_{\mathsf{pk}} := R^{\ell_K}_{\mathsf{pk}} + \Delta^b_{\mathsf{pk},\ell_K};$$
$$\begin{aligned} z'^{\tau_K}_{\mathsf{pk},\ell_K} &:= \boldsymbol{u}'^\top_{\mathsf{pk}} \cdot \boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}} + \boldsymbol{v}'^\top_{\mathsf{pk}} \cdot \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}} \\ &= z^{\tau_K}_{\mathsf{pk},\ell_K} - ({\boldsymbol{x}^0}^\top_{\mathsf{pk}} \cdot \boldsymbol{y}^{\tau_K,0}_{\mathsf{pk},\ell_K} - {\boldsymbol{x}^b}^\top_{\mathsf{pk}} \cdot \boldsymbol{y}^{\tau_K,b}_{\mathsf{pk},\ell_K}) \\ &= z^{\tau_K}_{\mathsf{pk},\ell_K} - \Delta^b_{\mathsf{pk},\ell_K}. \end{aligned}$$

Then the output of the experiment when $\alpha = 0$ can be rewritten as

$$(-\boldsymbol{x}^b_{\mathsf{pk}} + \boldsymbol{u}'_{\mathsf{pk}}, \boldsymbol{x}^0_{\mathsf{pk}} + \boldsymbol{v}'_{\mathsf{pk}}, \{(R'^{\ell_K}_{\mathsf{pk}} + z'^{\tau_K}_{\mathsf{pk},\ell_K})_{\tau_K\in[\mathsf{rep}_{\ell_K}]}\}_{\ell_K\in\mathcal{Q}_K})_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$$

Since $(\Delta^b_{\mathsf{pk},\ell_K})_{\mathsf{pk}}$ are zero shares, then $(R'^{\ell_K}_{\mathsf{pk}})_{\mathsf{pk}}$ are also uniformly-random zero shares. The lemma is then completed as the rewritten output is of the same distribution as the output in the experiment $\alpha = 1$. □

4.2 FH-IP-DDFE Construction

For every client, let d be an inner-product dimension, let X and Y be a message bound and a function bound of size $\mathsf{poly}(\lambda)$ respectively, let $\mathcal{L}_M$ and $\mathcal{L}_K$ be a message-label space and key-label space respectively. The scheme is described in Figure 8 with the following primitives:

- $\mathsf{IPE} = (\mathsf{iSetup}, \mathsf{iKeyGen}, \mathsf{iEnc}, \mathsf{iDKGen}, \mathsf{iDec})$ be a single-input function-hiding IPFE;
- $\mathsf{UZS} = (\mathsf{uSetUp}, \mathsf{uKeyGen}, \mathsf{uSeedGen}, \mathsf{uTokGen}, \mathsf{uSeedUpt}, \mathsf{uShareEval})$ be a bilinear-updatable pseudorandom zero-sharing scheme over $\mathbb{G}_2$ for a seeding label space $\mathcal{L}_K$ and an updating label space $\mathcal{L}_M$.
- $\mathsf{AoNE} = (\mathsf{aSetup}, \mathsf{aKeyGen}, \mathsf{aEnc}, \mathsf{aDKGen}, \mathsf{aDec})$ be an all-or-nothing encapsulation scheme.

Remark 5 (Size of Ciphertext/Decryption Key). In the above FH-IP-DDFE construction, if one uses the AoNE scheme that is constructed from a rate-1 identity-based encryption and employed in the hybrid-encryption mode as in [CDSG+20], then the size of each DDFE ciphertext/decryption key will be $O_\lambda(d + |\mathcal{U}|)$.

Construction:

- $\mathsf{SetUp}(1^\lambda)$: It generates $\mathcal{PG} \leftarrow \mathsf{PGGen}(1^\lambda)$ and sets up parameters of the underlying schemes: $\mathsf{ipp} \leftarrow \mathsf{iSetup}(1^\lambda)$; $\mathsf{upp} \leftarrow \mathsf{uSetUp}(1^\lambda)$; $\mathsf{app} \leftarrow \mathsf{aSetup}(1^\lambda)$. It returns
$$\mathsf{pp} = (\mathcal{PG}, \mathsf{ipp}, \mathsf{upp}, \mathsf{app}).$$
The parameters pp are implicit to other algorithms.
- $\mathsf{KeyGen}()$: Each client samples
$$k_{\mathsf{pk}} \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}; \quad (\mathsf{upk}, \mathsf{usk}_{\mathsf{pk}}) \leftarrow \mathsf{uKeyGen}(); \quad (\mathsf{apk}, \mathsf{ask}_{\mathsf{pk}}) \leftarrow \mathsf{aKeyGen}().$$
It returns $\mathsf{pk} = (\mathsf{upk}, \mathsf{apk})$ and $\mathsf{sk}_{\mathsf{pk}} = (k_{\mathsf{pk}}, \mathsf{usk}_{\mathsf{pk}}, \mathsf{ask}_{\mathsf{pk}})$.
- $\mathsf{Enc}(\mathsf{sk}_{\mathsf{pk}}, m)$: It parses $m = (\boldsymbol{m}, \mathcal{U}_M, \ell_M)$ and computes
  1. a UZS token: $\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}_M} \leftarrow \mathsf{uTokGen}(\mathsf{usk}_{\mathsf{pk}}, (\mathcal{U}_M, \ell_M))$;
  2. a random coin for IPE key generation: $\mathsf{coin}_{\mathsf{pk}} \leftarrow \mathsf{PRF}_{k_{\mathsf{pk}}}(\mathcal{U}_M)$;
  3. a $6d + |\mathcal{U}_M|$-length IPE secret key: $\mathsf{isk}_{\mathsf{pk}} = \mathsf{iKeyGen}(1^{6d+|\mathcal{U}_M|}; \mathsf{coin}_{\mathsf{pk}})$;
  4. an IPE encryption: $\boldsymbol{x} = (\boldsymbol{m}, \mathbf{0}^d)$
  $$\mathsf{ict}_{\mathsf{pk}} \leftarrow \mathsf{iEnc}(\mathsf{isk}_{\mathsf{pk}}, [\boldsymbol{x}, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}_M}, 0]_1);$$
  5. an AoNE layer on $\mathsf{ict}_{\mathsf{pk}}$: $\mathsf{act}_{\mathsf{pk}} \leftarrow \mathsf{aEnc}(\mathsf{ask}_{\mathsf{pk}}, (\mathsf{ict}_{\mathsf{pk}}, \mathcal{U}_M, \ell_M \| "ct"))$.

  It returns the ciphertext
$$\mathsf{ct}_{\mathsf{pk}} = (\mathsf{act}_{\mathsf{pk}}, \mathcal{U}_M, \ell_M).$$
- $\mathsf{DKGen}(\mathsf{sk}_{\mathsf{pk}}, k)$: It parses $k = (\boldsymbol{k}, \mathcal{U}_K, \ell_K)$ and computes
  1. a UZS seed: $[\boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}_K}]_2 \leftarrow \mathsf{uTokGen}(\mathsf{usk}_{\mathsf{pk}}, \mathcal{U}_K, \ell_K)$;
  2. a random coin for IPE key generation: $\mathsf{coin}_{\mathsf{pk}} \leftarrow \mathsf{PRF}_{k_{\mathsf{pk}}}(\mathcal{U}_K)$;
  3. a $6d + |\mathcal{U}_K|$-length IPE secret key: $\mathsf{isk}_{\mathsf{pk}} = \mathsf{iKeyGen}(1^{6d+|\mathcal{U}_K|}; \mathsf{coin}_{\mathsf{pk}})$;
  4. an IPE decryption key: $\boldsymbol{y} = (\boldsymbol{k}, \mathbf{0}^d)$
  $$\mathsf{idk}_{\mathsf{pk}} \leftarrow \mathsf{iDKGen}(\mathsf{isk}_{\mathsf{pk}}, [\boldsymbol{y}, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}_K}, 0]_2);$$
  5. an AoNE layer on $\mathsf{idk}_{\mathsf{pk}}$: $\mathsf{act}_{\mathsf{pk}} \leftarrow \mathsf{aEnc}(\mathsf{ask}_{\mathsf{pk}}, (\mathsf{idk}_{\mathsf{pk}}, \mathcal{U}_K, \ell_K \| "dk"))$.

  It returns the decryption key
$$\mathsf{dk}_{\mathsf{pk}} = (\mathsf{act}_{\mathsf{pk}}, \mathcal{U}_K, \ell_K).$$
- $\mathsf{Dec}\left((\mathsf{dk}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_K}, (\mathsf{ct}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M}, (\mathcal{U}_M, \ell_M), (\mathcal{U}_K, \ell_K)\right)$: If $\mathcal{U}_M = \mathcal{U}_K = \mathcal{U}$ is not true, it returns $\perp$. Otherwise,
  1. it parses $\mathsf{dk}_{\mathsf{pk}} = (\mathsf{act}_{\mathsf{pk}}, \mathcal{U}, \ell_K)$ and recovers the IPE decryption keys
  $$(\mathsf{idk}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}} = \mathsf{aDec}((\mathsf{act}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}, \mathcal{U}, \ell_K \| "dk");$$
  2. it parses $\mathsf{ct}_{\mathsf{pk}} = (\mathsf{act}'_{\mathsf{pk}}, \mathcal{U}, \ell_M)$ and recovers the IPE ciphertexts
  $$(\mathsf{ict}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}} = \mathsf{aDec}((\mathsf{act}'_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}, \mathcal{U}, \ell_M \| "ct");$$
  3. it computes $[\alpha]_T = \sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{iDec}(\mathsf{ict}_{\mathsf{pk}}, \mathsf{idk}_{\mathsf{pk}})$ and returns α.

Fig. 8. DDFE for Function-Hiding Inner Products

Correctness. By the correctness of the AoNE scheme, one can always recover IPE ciphertexts $(\mathsf{ict}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}$ and decryption keys $(\mathsf{idk}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}$. The correctness is then implied by the correctness of the IPE scheme and the UZS scheme:

$$\begin{aligned}
&\sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{iDec}(\mathsf{ict}_{\mathsf{pk}}, \mathsf{idk}_{\mathsf{pk}})\\
&= \sum_{\mathsf{pk}\in\mathcal{U}} [\boldsymbol{x}_{\mathsf{pk}}^{\top} \cdot \boldsymbol{y}_{\mathsf{pk}} + {\boldsymbol{b}_{\mathsf{pk},\mathcal{U}}^{\ell_M}}^{\top} \cdot \boldsymbol{a}_{\mathsf{pk},\mathcal{U}}^{\ell_K}]_T\\
&=[\sum_{\mathsf{pk}\in\mathcal{U}} \boldsymbol{x}_{\mathsf{pk}}^{\top} \cdot \boldsymbol{y}_{\mathsf{pk}}]_T + e\left([1]_1, \sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{uShareEval}(\mathsf{uSeedUpt}(\mathsf{seed}_{\mathsf{pk},\mathcal{U}}^{\ell_K}, \mathsf{token}_{\mathsf{pk},\mathcal{U}}^{\ell_M}))\right)\\
&=[\sum_{\mathsf{pk}\in\mathcal{U}} \boldsymbol{x}_{\mathsf{pk}}^{\top} \cdot \boldsymbol{y}_{\mathsf{pk}}]_T + e\left([1]_1, \sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{share}_{\mathsf{pk},\mathcal{U}}^{\ell_K||\ell_M}\right)\\
&=[\sum_{\mathsf{pk}\in\mathcal{U}} \boldsymbol{x}_{\mathsf{pk}}^{\top} \cdot \boldsymbol{y}_{\mathsf{pk}}]_T + e\left([1]_1, [0]_2\right) = [\sum_{\mathsf{pk}\in\mathcal{U}} \boldsymbol{x}_{\mathsf{pk}}^{\top} \cdot \boldsymbol{y}_{\mathsf{pk}}]_T = [\sum_{\mathsf{pk}\in\mathcal{U}} \boldsymbol{m}_{\mathsf{pk}}^{\top} \cdot \boldsymbol{k}_{\mathsf{pk}}]_T.
\end{aligned}$$

As the inner product $\sum_{\mathsf{pk}\in\mathcal{U}} \boldsymbol{m}_{\mathsf{pk}}^{\top} \cdot \boldsymbol{k}_{\mathsf{pk}}$ is of size $\mathsf{poly}(\lambda)$, it can always be recovered. □

4.3 Security Analysis

Theorem 2 (Indistinguishability for FH-IP-DDFE). *If IPE is a single-input* sym-IND-*secure FE for function-hiding inner products, AoNE is a* sym-IND-*secure all-or-nothing encapsulation, and UZS is an* otu-sta-IND-*secure updatable pseudorandom zero sharing, then the* FH-IP-DDFE *scheme constructed in Figure 8 is* sta-sym-IND *secure (as in Definition 5) in the standard model.*

Proof. In the static-corruption game, we can fix $\mathcal{PK}$ to be the set of parties generated by QNewHon() queries, $\mathcal{C}$ to be the set of corrupted parties in $\mathcal{PK}$ and $\mathcal{H} = \mathcal{PK}\backslash\mathcal{C}$ to be the set of honest parties. Let q_{xx} be the number of xx-oracle queries where xx $\in \{\mathsf{QNewHon}, \mathsf{QEnc}, \mathsf{QDKGen}, \mathsf{QCor}\}$. Given $\lambda \in \mathbb{N}$, we denote by $\mathsf{Adv}_{\mathbf{G}_i}$ the advantage of an PPT adversary $\mathcal{A}$ in each game $\mathbf{G}_i$, and Adv_{xx} be the best advantage of any PPT adversary against the primitive xx that is setup with λ.

The DDFE scheme can be parsed as a scheme for messages and keys in the forms $\boldsymbol{x}^b = (\boldsymbol{m}^b, \mathbf{0}^d)$ and $\boldsymbol{y}^b = (\boldsymbol{k}^b, \mathbf{0}^d)$ respectively. Let $\mathcal{Q}_M$ and $\mathcal{Q}_K$[6] be the set of adaptive encryption queries and decryption key queries sent by $\mathcal{A}$ respectively by the end of the game. When $\mathsf{pk} \in \mathcal{H}$, a decryption key query $(\mathsf{pk}, \boldsymbol{y}^0, \boldsymbol{y}^1, \mathcal{U}_K, \ell_K)$[7] $\in \mathcal{Q}_K$ is said to be incomplete if there exists $\mathsf{pk}' \in \mathcal{H} \cap \mathcal{U}_K$ such that there is no decryption key query $(\mathsf{pk}', \boldsymbol{y}'^0, \boldsymbol{y}'^1, \mathcal{U}_K, \ell_K) \in \mathcal{Q}_K$. Similarly, when $\mathsf{pk} \in \mathcal{H}$, an encryption query $(\mathsf{pk}, \boldsymbol{x}^0, \boldsymbol{x}^1, \mathcal{U}_M, \ell_M) \in \mathcal{Q}_M$ is said to be incomplete if there exists $\mathsf{pk}' \in \mathcal{H} \cap \mathcal{U}_M$ such that there is no encryption query $(\mathsf{pk}', \boldsymbol{x}'^0, \boldsymbol{x}'^1, \mathcal{U}_M, \ell_M) \in \mathcal{Q}_M$.

We proceed via a global hybrid argument: we describe the changes in the IND game by using the games $\mathbf{G}_0$, $\mathbf{G}_1$, and $\mathbf{G}_3$ (see Figure 13). Notably, the game $\mathbf{G}_0$ corresponds to sta-sym-IND security game as defined in Definition 5, and the game $\mathbf{G}_3$ corresponds to the case where adversary's advantage is 0 since there is no challenge bit b. We first prove the indistinguishability under the assumption that all encryption queries and decryption-key queries are complete.

The case of complete queries. The transition between $\mathbf{G}_1$ and $\mathbf{G}_3$ requires intermediate games $\mathbf{G}_{1.1}$, $\{\mathbf{G}_{1.1.\ell_M}\}_{\ell_M\in\mathcal{Q}_M}$, and $\mathbf{G}_2$ (see Figure 9). The transition between $\{\mathbf{G}_{1.1.\ell_M}\}_{\ell_M\in\mathcal{Q}_M}$ requires intermediate $(\mathbf{G}^{\star}_{\ell_M.i})_{i\in[6]}$ (see Figure 10) for each ℓ_M.

The security notion of UZS applies to the transition from $\mathbf{G}^{*}_{\ell_M.2}$ to $\mathbf{G}^{*}_{\ell_M.3}$ as in the lemma below.

Lemma 3 (FH-IP-DDFE: **Transition from** $\mathbf{G}^{*}_{\ell_M.2}$ **to** $\mathbf{G}^{*}_{\ell_M.3}$**).** *For any PPT adversary $\mathcal{A}$, the advantage in distinguishing two games is*

$$\left|\mathsf{Adv}_{\mathbf{G}^{*}_{\ell_M.2}} - \mathsf{Adv}_{\mathbf{G}^{*}_{\ell_M.3}}\right| \leq \mathsf{Adv}_{\mathsf{UZS}}^{\text{otu-sta}}.$$

[6] $\mathcal{Q}_M$ and $\mathcal{Q}_K$ contain elements of the form $(\mathsf{pk}, \cdot, \cdot, \mathcal{U}, \ell)$. For a string xx $\in \{0,1\}^*$, we denote by xx $\in \mathcal{Q}_M$ or xx $\in \mathcal{Q}_K$ if there exists a query containing xx.

[7] For ease of exposition, we omit subscript/superscript indexes in contexts where these indexes are explicitly mentioned in queries.

Game	iEnc	iKeyGen	Assumption
$\mathbf{G}_1$	$(\boldsymbol{x}^{b,\tau_M}_{\mathsf{pk}}, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, 0)$	$(\boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}}, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, 0)$	
$\mathbf{G}_{1.1}$	$(\boldsymbol{x}^{b,\tau_M}_{\mathsf{pk}}, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, 0)$	$(\boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \mathbf{0}^{2d}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, 0)$	IND of IPE
$\mathbf{G}_{1.1.\ell_M}$ $\ell_M \in \mathcal{Q}_M$	$\ell'_M < \ell_M$: $(\mathbf{0}^{2d}, \boldsymbol{x}^{0,\tau_M}_{\mathsf{pk}}, \mathbf{0}^{2d}, \boldsymbol{b}^{\ell'_M}_{\mathsf{pk},\mathcal{U}}, 0)$ $\ell'_M \geq \ell_M$: $(\boldsymbol{x}^{b,\tau_M}_{\mathsf{pk}}, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{b}^{\ell'_M}_{\mathsf{pk},\mathcal{U}}, 0)$	same as in $\mathbf{G}_{2.1}$	explained in Figure 10
$\mathbf{G}_2$	$(\mathbf{0}^{2d}, \boldsymbol{x}^{0,\tau_M}_{\mathsf{pk}}, \mathbf{0}^{2d}, \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, 0)$	$(\mathbf{0}^{2d}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \mathbf{0}^{2d}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, 0)$	IND of IPE
$\mathbf{G}_3$	$(\boldsymbol{x}^{0,\tau_M}_{\mathsf{pk}}, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, 0)$	$(\boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, 0)$	IND of IPE

Fig. 9. Sequence of hybrids for transition from $\mathbf{G}_1$ to $\mathbf{G}_3$ in the security proof of FH-IP-DDFE (see Theorem 2), under the assumption that all queries are *complete*. All changes are made within iEnc and iDKGen algorithms for the replies of encryption and decryption key generation oracles respectively, and higlighted in blue.

Proof. We build an adversary $\mathcal{B}$ against the otu-sta-IND security of UZS from an adversary $\mathcal{A}$ that distinguishes between two games in the transition. To simulate a FH-IP-DDFE challenger, $\mathcal{B}$ uses the UZS oracles to handle all UZS related operations in DDFE.

- For each encryption query on $(\mathsf{pk}, \boldsymbol{x}^0_{\mathsf{pk}}, \boldsymbol{x}^1_{\mathsf{pk}}, \mathcal{U}, \ell'_M)$,
  - If $\ell'_M \neq \ell_M$: $\mathcal{B}$ obtains $\boldsymbol{b}^{\ell'_M}_{\mathsf{pk},\mathcal{U}} \leftarrow \mathsf{QTokGen}(\mathsf{pk}, \mathcal{U}, \ell'_M)$ to complete the message to IPE encryption.
  - If $\ell'_M = \ell_M$: it does not have to obtain $\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}$ as the message to IPE encryption is $(\boldsymbol{x}^b_{\mathsf{pk}} + \boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \mathbf{0}^{|\mathcal{U}|-1}, 1)$ in this case.
- For each decryption-key query on $(\mathsf{pk}, \boldsymbol{y}^0_{\mathsf{pk}}, \boldsymbol{y}^1_{\mathsf{pk}}, \mathcal{U}, \ell_K)$,
  - $\mathcal{B}$ obtains $[\boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}]_2 \leftarrow \mathsf{QSeedGen}(\mathsf{pk}, \mathcal{U}, \ell_K)$ and $([R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}}]_2)_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} \leftarrow \mathsf{QShare}(\mathcal{U}, \ell_K \| \ell_M)$;
  - $\mathcal{B}$ completes the IPE key in $\mathbb{G}_2$ as

$$[\boldsymbol{y}^b_{\mathsf{pk}}, \boldsymbol{y}^0_{\mathsf{pk}}, \boldsymbol{y}^0_{\mathsf{pk}}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}} + z^{\tau_K}_{\mathsf{pk},\ell_K}]_2.$$

The admissibility condition $(*)$ of UZS holds since

- all the corruption queries in FH-IP-DDFE are sent in one shot;
- $\mathsf{QShare}(\mathcal{U}, \ell_K \| \ell_M)$ queries are made for the same ℓ_M on every $\mathcal{U}$ and there are no $\mathsf{QTokGen}(\mathsf{pk}, \mathcal{U}, \ell_M)$ queries required.

When UZS.$b = 0$, one has $[R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}}]_2 = [{\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}]_2$ which corresponds to $\mathbf{G}^*_{\ell_M.2}$; and when UZS.$b = 1$, one has $([R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}}]_2)_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$ are random shares of $-\sum_{\mathsf{pk}\in\mathcal{C}\cap\mathcal{U}}[{\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}]_2$, which corresponds to $\mathbf{G}^*_{\ell_M.3}$. Therefore, one has

$$\left|\mathsf{Adv}_{\mathbf{G}^*_{\ell_M.2}} - \mathsf{Adv}_{\mathbf{G}^*_{\ell_M.3}}\right| \leq \mathsf{Adv}^{\text{otu-sta}}_{\mathsf{UZS}}.$$

□

The Injection Lemma applies to the transition from $\mathbf{G}^*_{\ell_M.4}$ to $\mathbf{G}^*_{\ell_M.5}$ as in the lemma below.

Lemma 4 (FH-IP-DDFE: **Transition from $\mathbf{G}^*_{\ell_M.4}$ to $\mathbf{G}^*_{\ell_M.5}$**). *The two games $\mathbf{G}^*_{\ell_M.4}$ and $\mathbf{G}^*_{\ell_M.5}$ are identical.*

Proof. In these games, we parse the randomness $R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}} = R^{0,\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}} + R^{1,\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}}$ where $(R^{1,\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}})_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$ are random shares of $-\sum_{\mathsf{pk}\in\mathcal{C}\cap\mathcal{U}} {\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}$ and $(R^{0,\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}})_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$ are random shares of 0.

Given each $\mathcal{U}$, we set $\boldsymbol{q}_{\ell_M} = (\boldsymbol{x}^{0,1}_{\mathsf{pk}}, \boldsymbol{x}^{1,1}_{\mathsf{pk}}, \mathcal{U}, \ell_M)_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$ and for each $\ell_K \in \mathcal{Q}_K$, we set $\boldsymbol{q}_{\ell_K} = (\boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{1,\tau_K}_{\mathsf{pk}}, \mathcal{U}, \ell_K)_{\tau_K\in[\mathsf{rep}_{\ell_K}],\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$. Then the difference between two games are the outputs of $\{\mathsf{Exp}^{\alpha}_{\mathrm{INJ}}(b, \mathcal{U}, \boldsymbol{q}_{\ell_M}, \{\boldsymbol{q}_{\ell_K}\}_{\ell_K\in\mathcal{Q}_K})\}_{\mathcal{U}\in\mathcal{Q}_M}$, where $\alpha = 0$ corresponds to $\mathbf{G}^*_{\ell_M.4}$ and $\alpha = 1$ corresponds to

Game	Adjustment	Assumption
$\mathbf{G}^\star_{\ell_M} := \mathbf{G}_{1.1.\ell_M}$	as in Figure 9	
$\mathbf{G}^\star_{\ell_M.1}$	iEnc: on label ℓ_M, $(\boldsymbol{x}^{b,\tau_M}_{\mathsf{pk}} + \boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \mathbf{0}^{2d}, \boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, 1)$ where $\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}} \xleftarrow{\$} \mathbb{Z}_p^d$ iKeyGen: on all labels ℓ_K, $(\boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, -\boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, 0)$	IND of IPE
$\mathbf{G}^\star_{\ell_M.2}$	iEnc: on label ℓ_M, $(\boldsymbol{x}^{b,\tau_M}_{\mathsf{pk}} + \boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \mathbf{0}^{\vert U\vert-1}, 1)$ where $\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}} \xleftarrow{\$} \mathbb{Z}_p^d$ iKeyGen: on all labels ℓ_K, $(\boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, {\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}} + z^{\tau_K}_{\mathsf{pk},\ell_K})$ where $z^{\tau_K}_{\mathsf{pk},\ell_K} = -{\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}}$	IND of IPE
$\mathbf{G}^\star_{\ell_M.3}$	iEnc: on label ℓ_M, $(\boldsymbol{x}^{b,\tau_M}_{\mathsf{pk}} + \boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \mathbf{0}^{\vert U\vert-1}, 1)$ where $\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}} \xleftarrow{\$} \mathbb{Z}_p^d$ iKeyGen: on all labels ℓ_K, $(\boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}} + z^{\tau_K}_{\mathsf{pk},\ell_K})$ where $(R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}})_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$ are random subjected to $\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}} = -\sum_{\mathsf{pk}\in\mathcal{C}\cap\mathcal{U}} {\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}$	IND of UZS
$\mathbf{G}^\star_{\ell_M.4}$	iEnc: on label ℓ_M, $(\boldsymbol{x}^{b,\tau_M}_{\mathsf{pk}} + \boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \mathbf{0}^{2d}, \boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \mathbf{0}^{\vert U\vert-1}, 1)$ where $\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}} \xleftarrow{\$} \mathbb{Z}_p^d$ iKeyGen: on all labels ℓ_K, $(\boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}} + z^{\tau_K}_{\mathsf{pk},\ell_K})$ where $z^{\tau_K}_{\mathsf{pk},\ell_K} = -({\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}} + {\boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}})$	IND of IPE
$\mathbf{G}^\star_{\ell_M.5}$	iEnc: on label ℓ_M, $(\boldsymbol{x}^{b,\tau_M}_{\mathsf{pk}} - \boldsymbol{x}^{b,1}_{\mathsf{pk}} + \boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \mathbf{0}^{2d}, \boldsymbol{x}^{0,1}_{\mathsf{pk}} + \boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \mathbf{0}^{\vert U\vert-1}, 1)$ where $\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}} \xleftarrow{\$} \mathbb{Z}_p^d$ iKeyGen: $(\boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}} + z^{\tau_K}_{\mathsf{pk},\ell_K})$	Statistical by Injection Lemma
$\mathbf{G}^\star_{\ell_M.6}$	iEnc: on label ℓ_M, $(\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \mathbf{0}^{2d}, \boldsymbol{x}^{0,\tau_M}_{\mathsf{pk}} + \boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \mathbf{0}^{\vert U\vert-1}, 1)$ where $\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}} \xleftarrow{\$} \mathbb{Z}_p^d$ iKeyGen: $(\boldsymbol{y}^{b,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \boldsymbol{y}^{0,\tau_K}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_K}_{\mathsf{pk},\mathcal{U}}, R^{\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}} + z^{\tau_K}_{\mathsf{pk},\ell_K})$	IND of IPE
$\mathbf{G}^\star_{\ell_M+1} := \mathbf{G}_{1.1.\ell_M+1}$	as in Figure 9	Symmetric transitions as from $\mathbf{G}^\star_{\ell_M}$ to $\mathbf{G}^\star_{\ell_M.4}$

Fig. 10. Sequence of hybrids for each transition from $\mathbf{G}_{1.1.\ell_M}$ to $\mathbf{G}_{1.1.\ell_M+1}$ in Figure 9. We denote by $\mathcal{H}$ (and $\mathcal{C}$) the set of honest (and corrupted) users and $(\ell_M + 1)$ the subsequent message label of ℓ_M in the ordered set $\mathcal{Q}_M$ of encryption queries.

$\mathbf{G}^*_{\ell_M.5}$. By Lemma 2, we have $\mathsf{Exp}^0_{\mathrm{INJ}}(b, \mathcal{U}, \boldsymbol{q}_{\ell_M}, \{\boldsymbol{q}_{\ell_K}\}_{\ell_K \in \mathcal{Q}_K}) = \mathsf{Exp}^1_{\mathrm{INJ}}(b, \mathcal{U}, \boldsymbol{q}_{\ell_M}, \{\boldsymbol{q}_{\ell_K}\}_{\ell_K \in \mathcal{Q}_K})$ with respect to experiment randomness $(\boldsymbol{u}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, \boldsymbol{v}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, R^{0,\ell_M,\ell_K}_{\mathsf{pk},\mathcal{U}})$ that are generated independently for each $\mathcal{U}$. Therefore, the lemma is complete. ☐

Lemmas for other transitions and the indistinguishability in the case of incomplete queries are provided in Appendix B.2. By completing the proof in both cases, we complete the theorem. ☐

5 Attribute-Weighted-Sum DDFE

In this section, we construct a DDFE scheme for attribute-weighted sums that is sel-sta-sym-IND secure in the standard model.

5.1 Construction

Let $\mathcal{L}_M$ and $\mathcal{L}_K$ be a message-label space and a key-label space respectively. The AWS-DDFE scheme is described in Figure 11 with these primitives: $\mathsf{AWIPE} = (\mathsf{aiSetup}, \mathsf{aiKeyGen}, \mathsf{aiEnc}, \mathsf{aiDKGen}, \mathsf{aiDec})$ be a FE for attribute-weighted sums with function-hiding inner products; UZS be a bilinear-updatable pseudorandom zero-sharing scheme over $\mathbb{G}_2$ for a seeding-label space $\mathcal{L}_K$ and an updating-label space $\mathcal{L}_M$; AoNE be an all-or-nothing encapsulation scheme.

We defer the correctness and the efficiency analysis to Section C.

5.2 Security Analysis

Theorem 3 (Indistinguishability for AWS-DDFE**).** *If* AWIPE *is a single-input* sel-sym-IND-*secure FE for attribute-weighted sums with function-hiding inner products,* AoNE *is a* sel-sym-IND-*secure all-or-nothing encapsulation, and* UZS *is an* otu-sta-IND-*secure updatable pseudorandom zero sharing, then the* AWS-DDFE *scheme constructed in Figure 11 is* sel-sta-sym-IND *secure (as in Definition 5) in the standard model.*

We defer the proof of the above theorem to Section C.

Acknowledgements. We would like to thank Ky Nguyen, Duong Hieu Phan and David Pointcheval for fruitful discussions that motivated this work.

References

ABDP15. M. Abdalla, F. Bourse, A. De Caro, and D. Pointcheval. Simple functional encryption schemes for inner products. In *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*, *Lecture Notes in Computer Science* 9020, pages 733–751, Gaithersburg, MD, USA, March 30 – April 1, 2015. Springer, Heidelberg, Germany.

ABG19. M. Abdalla, F. Benhamouda, and R. Gay. From single-input to multi-client inner-product functional encryption. In *Advances in Cryptology – ASIACRYPT 2019, Part III*, *Lecture Notes in Computer Science* 11923, pages 552–582, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany.

ABKW19. M. Abdalla, F. Benhamouda, M. Kohlweiss, and H. Waldner. Decentralizing inner-product functional encryption. In *PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part II*, *Lecture Notes in Computer Science* 11443, pages 128–157, Beijing, China, April 14–17, 2019. Springer, Heidelberg, Germany.

ABM$^+$20. M. Abdalla, F. Bourse, H. Marival, D. Pointcheval, A. Soleimanian, and H. Waldner. Multi-client inner-product functional encryption in the random-oracle model. In *SCN 20: 12th International Conference on Security in Communication Networks*, *Lecture Notes in Computer Science* 12238, pages 525–545, Amalfi, Italy, September 14–16, 2020. Springer, Heidelberg, Germany.

ACF$^+$18. M. Abdalla, D. Catalano, D. Fiore, R. Gay, and B. Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In *Advances in Cryptology – CRYPTO 2018, Part I*, *Lecture Notes in Computer Science* 10991, pages 597–627, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

Construction:

- $\mathsf{SetUp}(1^\lambda)$: It generates $\mathcal{PG} \leftarrow \mathsf{PGGen}(1^\lambda)$ and sets up parameters of the underlying schemes: $\mathsf{aipp} \leftarrow \mathsf{aiSetup}(1^\lambda); \mathsf{upp} \leftarrow \mathsf{uSetUp}(1^\lambda); \mathsf{app} \leftarrow \mathsf{aSetup}(1^\lambda)$. It returns
$$\mathsf{pp} = (\mathcal{PG}, \mathsf{aipp}, \mathsf{upp}, \mathsf{app}).$$
The parameters pp are implicit to other algorithms.
- $\mathsf{KeyGen}()$: Each client samples
$$k_{\mathsf{pk}} \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}; \quad (\mathsf{upk}, \mathsf{usk}_{\mathsf{pk}}) \leftarrow \mathsf{uKeyGen}(); \quad (\mathsf{apk}, \mathsf{ask}_{\mathsf{pk}}) \leftarrow \mathsf{aKeyGen}().$$
It returns $\mathsf{pk} = (\mathsf{upk}, \mathsf{apk})$ and $\mathsf{sk}_{\mathsf{pk}} = (k_{\mathsf{pk}}, \mathsf{usk}_{\mathsf{pk}}, \mathsf{ask}_{\mathsf{pk}})$.
- $\mathsf{Enc}(\mathsf{sk}_{\mathsf{pk}}, m)$: It parses $m = ((\boldsymbol{x}_j, \boldsymbol{z}_j)_{j\in[N]}, \mathcal{U}_M, \ell_M)$[a] and computes
  1. a UZS token: $\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}_M} \leftarrow \mathsf{uTokGen}(\mathsf{usk}_{\mathsf{pk}}, \mathcal{U}_M, \ell_M)$;
  2. a random coin for AWIPE key generation: $\mathsf{coin}_{\mathsf{pk}} \leftarrow \mathsf{PRF}_{k_{\mathsf{pk}}}(\mathcal{U}_M)$;
  3. a AWIPE secret key: $\mathsf{aisk}_{\mathsf{pk}} = \mathsf{aiKeyGen}(1^{|\mathcal{U}_M|}_{\mathsf{ip}}; \mathsf{coin}_{\mathsf{pk}})$;
  4. an AWIPE encryption:
$$\mathsf{aict}_{\mathsf{pk}} \leftarrow \mathsf{aiEnc}(\mathsf{aisk}_{\mathsf{pk}}, (\boldsymbol{x}_j, \boldsymbol{z}_j)_{j\in[N]}, [\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}_M}, 0]_1);$$
  5. an AoNE layer on $\mathsf{ict}_{\mathsf{pk}}$: $\mathsf{act}_{\mathsf{pk}} \leftarrow \mathsf{aEnc}(\mathsf{ask}_{\mathsf{pk}}, (\mathsf{aict}_{\mathsf{pk}}, \mathcal{U}_M, \ell_M \| "ct"))$.

  It returns the ciphertext
$$\mathsf{ct}_{\mathsf{pk}} = (\mathsf{act}_{\mathsf{pk}}, \mathcal{U}_M, \ell_M).$$
- $\mathsf{DKGen}(\mathsf{sk}_{\mathsf{pk}}, k)$: It parses $k = (\boldsymbol{f} := (f_{\mathsf{pk}}, \mathsf{pk})_{\mathsf{pk}\in\mathcal{U}_K}, \mathcal{U}_K)$ and computes
  1. a UZS seed: $[\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}_K}]_2 \leftarrow \mathsf{uSeedGen}(\mathsf{usk}_{\mathsf{pk}}, \mathcal{U}_K, \ell_{\boldsymbol{f}})$;[b]
  2. a random coin for AWIPE key generation: $\mathsf{coin}_{\mathsf{pk}} \leftarrow \mathsf{PRF}_{k_{\mathsf{pk}}}(\mathcal{U}_K)$;
  3. a AWIPE secret key: $\mathsf{aisk}_{\mathsf{pk}} = \mathsf{aiKeyGen}(1^{|\mathcal{U}_K|}_{\mathsf{ip}}; \mathsf{coin}_{\mathsf{pk}})$;
  4. an AWIPE decryption key:
$$\mathsf{aidk}_{\mathsf{pk}} \leftarrow \mathsf{aiDKGen}(\mathsf{aisk}_{\mathsf{pk}}, f_{\mathsf{pk}}, [\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}_K}, 0]_2);$$
  5. an AoNE layer on $\mathsf{aidk}_{\mathsf{pk}}$: $\mathsf{act}_{\mathsf{pk}} \leftarrow \mathsf{aEnc}(\mathsf{ask}_{\mathsf{pk}}, (\mathsf{aidk}_{\mathsf{pk}}, \mathcal{U}_K, \ell_{\boldsymbol{f}} \| "dk"))$.

  It returns the decryption key
$$\mathsf{dk}_{\mathsf{pk}} = (\mathsf{act}_{\mathsf{pk}}, \mathcal{U}_K, \ell_{\boldsymbol{f}}).$$
- $\mathsf{Dec}\left((\mathsf{dk}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_K}, (\mathsf{ct}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M}, (\mathcal{U}_M, \ell_M), (\mathcal{U}_K, \ell_{\boldsymbol{f}})\right)$: If $\mathcal{U}_M = \mathcal{U}_K = \mathcal{U}$ is not true, it returns $\perp$. Otherwise,
  1. it parses $\mathsf{dk}_{\mathsf{pk}} = (\mathsf{act}_{\mathsf{pk}}, \mathcal{U}, \ell_{\boldsymbol{f}})$ and recovers the AWIPE decryption keys
$$(\mathsf{aidk}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}} = \mathsf{aDec}((\mathsf{act}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}, \mathcal{U}, \ell_{\boldsymbol{f}} \| "dk");$$
  2. it parses $\mathsf{ct}_{\mathsf{pk}} = (\mathsf{act}'_{\mathsf{pk}}, \mathcal{U}, \ell_M)$ and recovers the AWIPE ciphertexts
$$(\mathsf{aict}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}} = \mathsf{aDec}((\mathsf{act}'_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}, \mathcal{U}, \ell_M \| "ct");$$

  It returns $[\alpha]_T = \sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{aiDec}(\mathsf{aict}_{\mathsf{pk}}, \mathsf{aidk}_{\mathsf{pk}})$.

[a] Each client can choose an arbitrary number N of AWS inputs.
[b] $\ell_{\boldsymbol{f}} \in \mathcal{L}_K$ contains a description of $\boldsymbol{f}$.

Fig. 11. DDFE for Attribute-Weighted Sums

ACF$^+$20. S. Agrawal, M. Clear, O. Frieder, S. Garg, A. O'Neill, and J. Thaler. Ad hoc multi-input functional encryption. In *ITCS 2020: 11th Innovations in Theoretical Computer Science Conference*, pages 40:1–40:41, Seattle, WA, USA, January 12–14, 2020. LIPIcs.

ACGU20. M. Abdalla, D. Catalano, R. Gay, and B. Ursu. Inner-product functional encryption with fine-grained access control. In *Advances in Cryptology – ASIACRYPT 2020, Part III*, *Lecture Notes in Computer Science* 12493, pages 467–497, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

AGT21a. S. Agrawal, R. Goyal, and J. Tomida. Multi-input quadratic functional encryption from pairings. In *Advances in Cryptology – CRYPTO 2021, Part IV*, *Lecture Notes in Computer Science* 12828, pages 208–238, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

AGT21b. S. Agrawal, R. Goyal, and J. Tomida. Multi-party functional encryption. In *TCC 2021: 19th Theory of Cryptography Conference, Part II*, *Lecture Notes in Computer Science* 13043, pages 224–255, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany.

AGT22. S. Agrawal, R. Goyal, and J. Tomida. Multi-input quadratic functional encryption: Stronger security, broader functionality. In *TCC 2022: 20th Theory of Cryptography Conference, Part I*, *Lecture Notes in Computer Science* 13747, pages 711–740, Chicago, IL, USA, November 7–10, 2022. Springer, Heidelberg, Germany.

AGW20. M. Abdalla, J. Gong, and H. Wee. Functional encryption for attribute-weighted sums from k-Lin. In *Advances in Cryptology – CRYPTO 2020, Part I*, *Lecture Notes in Computer Science* 12170, pages 685–716, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

ALMT20. S. Agrawal, B. Libert, M. Maitra, and R. Titiu. Adaptive simulation security for inner product functional encryption. In *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part I*, *Lecture Notes in Computer Science* 12110, pages 34–64, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany.

ALS16. S. Agrawal, B. Libert, and D. Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In *Advances in Cryptology – CRYPTO 2016, Part III*, *Lecture Notes in Computer Science* 9816, pages 333–362, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

ATY23. S. Agrawal, J. Tomida, and A. Yadav. Attribute-based multi-input FE (and more) for attribute-weighted sums. In *Advances in Cryptology – CRYPTO 2023, Part IV*, *Lecture Notes in Computer Science* 14084, pages 464–497, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Heidelberg, Germany.

BCFG17. C. E. Z. Baltico, D. Catalano, D. Fiore, and R. Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In *Advances in Cryptology – CRYPTO 2017, Part I*, *Lecture Notes in Computer Science* 10401, pages 67–98, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

BIK$^+$17. K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1175–1191, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

BJK15. A. Bishop, A. Jain, and L. Kowalczyk. Function-hiding inner product encryption. In *Advances in Cryptology – ASIACRYPT 2015, Part I*, *Lecture Notes in Computer Science* 9452, pages 470–491, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

BSW11. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC 2011: 8th Theory of Cryptography Conference*, *Lecture Notes in Computer Science* 6597, pages 253–273, Providence, RI, USA, March 28–30, 2011. Springer, Heidelberg, Germany.

CDG$^+$18. J. Chotard, E. Dufour Sans, R. Gay, D. H. Phan, and D. Pointcheval. Decentralized multi-client functional encryption for inner product. In *Advances in Cryptology – ASIACRYPT 2018, Part II*, *Lecture Notes in Computer Science* 11273, pages 703–732, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

CDSG$^+$20. J. Chotard, E. Dufour-Sans, R. Gay, D. H. Phan, and D. Pointcheval. Dynamic decentralized functional encryption. In *Advances in Cryptology – CRYPTO 2020, Part I*, *Lecture Notes in Computer Science* 12170, pages 747–775, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

Cha07. M. Chase. Multi-authority attribute based encryption. In *TCC 2007: 4th Theory of Cryptography Conference*, *Lecture Notes in Computer Science* 4392, pages 515–534, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

CLT18. G. Castagnos, F. Laguillaumie, and I. Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In *Advances in Cryptology – ASIACRYPT 2018, Part II*, *Lecture Notes in Computer Science* 11273, pages 733–764, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

DP19. E. Dufour Sans and D. Pointcheval. Unbounded inner-product functional encryption with succinct keys. In *ACNS 19: 17th International Conference on Applied Cryptography and Network Security*, *Lecture Notes in Computer Science* 11464, pages 426–441, Bogota, Colombia, June 5–7, 2019. Springer, Heidelberg, Germany.

GGG$^+$14. S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou. Multi-input functional encryption. In *Advances in Cryptology – EUROCRYPT 2014*, *Lecture Notes in Computer Science* 8441, pages 578–602, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

GKL$^+$13. S. D. Gordon, J. Katz, F.-H. Liu, E. Shi, and H.-S. Zhou. Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/774, 2013. `https://eprint.iacr.org/2013/774`.

IW14. Y. Ishai and H. Wee. Partial garbling schemes and their applications. In *ICALP 2014: 41st International Colloquium on Automata, Languages and Programming, Part I*, *Lecture Notes in Computer Science* 8572, pages 650–662, Copenhagen, Denmark, July 8–11, 2014. Springer, Heidelberg, Germany.

KDK11. K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *PETS 2011: 11th International Symposium on Privacy Enhancing Technologies*, *Lecture Notes in Computer Science* 6794, pages 175–191, Waterloo, ON, Canada, July 27–29, 2011. Springer, Heidelberg, Germany.

LT19. B. Libert and R. Titiu. Multi-client functional encryption for linear functions in the standard model from LWE. In *Advances in Cryptology – ASIACRYPT 2019, Part III*, *Lecture Notes in Computer Science* 11923, pages 520–551, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany.

LW11. A. B. Lewko and B. Waters. Decentralizing attribute-based encryption. In *Advances in Cryptology – EUROCRYPT 2011*, *Lecture Notes in Computer Science* 6632, pages 568–588, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

LWG$^+$23. Y. Li, J. Wei, F. Guo, W. Susilo, and X. Chen. Robust decentralized multi-client functional encryption: Motivation, definition, and inner-product constructions. In *Advances in Cryptology – ASIACRYPT 2023, Part V*, *Lecture Notes in Computer Science* 14442, pages 134–165, Guangzhou, China, December 4–8, 2023. Springer, Heidelberg, Germany.

MJ18. Y. Michalevsky and M. Joye. Decentralized policy-hiding ABE with receiver privacy. In *ESORICS 2018: 23rd European Symposium on Research in Computer Security, Part II*, *Lecture Notes in Computer Science* 11099, pages 548–567, Barcelona, Spain, September 3–7, 2018. Springer, Heidelberg, Germany.

MKMS22. J. M. B. Mera, A. Karmakar, T. Marc, and A. Soleimanian. Efficient lattice-based inner-product functional encryption. In *PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part II*, *Lecture Notes in Computer Science* 13178, pages 163–193, Virtual Event, March 8–11, 2022. Springer, Heidelberg, Germany.

NPP22. K. Nguyen, D. H. Phan, and D. Pointcheval. Multi-client functional encryption with fine-grained access control. In *Advances in Cryptology – ASIACRYPT 2022, Part I*, *Lecture Notes in Computer Science* 13791, pages 95–125, Taipei, Taiwan, December 5–9, 2022. Springer, Heidelberg, Germany.

NPP23a. D. D. Nguyen, D. H. Phan, and D. Pointcheval. Verifiable decentralized multi-client functional encryption for inner product. In *Advances in Cryptology – ASIACRYPT 2023, Part V*, *Lecture Notes in Computer Science* 14442, pages 33–65, Guangzhou, China, December 4–8, 2023. Springer, Heidelberg, Germany.

NPP23b. K. Nguyen, D. H. Phan, and D. Pointcheval. Optimal security notion for decentralized multi-client functional encryption. In *ACNS 23: 21st International Conference on Applied Cryptography and Network Security, Part II*, *Lecture Notes in Computer Science* 13906, pages 336–365, Kyoto, Japan, June 19–22, 2023. Springer, Heidelberg, Germany.

SV23. E. Shi and N. Vanjani. Multi-client inner product encryption: Function-hiding instantiations without random oracles. In *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*, *Lecture Notes in Computer Science* 13940, pages 622–651, Atlanta, GA, USA, May 7–10, 2023. Springer, Heidelberg, Germany.

SW05. A. Sahai and B. R. Waters. Fuzzy identity-based encryption. In *Advances in Cryptology – EUROCRYPT 2005*, *Lecture Notes in Computer Science* 3494, pages 457–473, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

Tom19. J. Tomida. Tightly secure inner product functional encryption: Multi-input and function-hiding constructions. In *Advances in Cryptology – ASIACRYPT 2019, Part III*, *Lecture Notes in Computer Science* 11923, pages 459–488, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany.

TT18. J. Tomida and K. Takashima. Unbounded inner product functional encryption from bilinear maps. In *Advances in Cryptology – ASIACRYPT 2018, Part II*, *Lecture Notes in Computer Science* 11273, pages 609–639, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

Üna20. A. Ünal. Impossibility results for lattice-based functional encryption schemes. In *Advances in Cryptology – EUROCRYPT 2020, Part I*, *Lecture Notes in Computer Science* 12105, pages 169–199, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

ZLZ+24. Z. Zhu, J. Li, K. Zhang, J. Gong, and H. Qian. Registered functional encryptions from pairings. Cryptology ePrint Archive, Paper 2024/327, 2024. `https://eprint.iacr.org/2024/327`.

A More Definitions

A.1 Arithmetic Branching Programs

Definition 12 (Arithmetic Branching Programs (ABPs) [IW14, AGW20, ATY23]). *An arithmetic branching program $f : \mathbb{Z}_p^{n_0} \to \mathbb{Z}_p$ is defined by a prime p, a directed acyclic graph (V, E), two special vertices $v_0, v_1 \in V$, and a labeling function $\sigma : E \to \mathcal{F}^{\mathsf{Affine}}$, where $\mathcal{F}^{\mathsf{Affine}}$ consists of all affine functions $g : \mathbb{Z}_p^{n_0} \to \mathbb{Z}_p$. The size of f is the number of vertices $|V|$. Given an input $\boldsymbol{x} \in \mathbb{Z}_p^{n_0}$ to the ABP, we can assign a $\mathbb{Z}_p$ element to edge $e \in E$ by $\sigma(e)(\boldsymbol{x})$. Let P be the set of all paths from v_0 to v_1. Each element in P can be represented by a subset of E. The output of the ABP on input $\boldsymbol{x}$ is defined as $\sum_{E' \in P} \prod_{e \in E'} \sigma(e)(\boldsymbol{x})$. We can extend the definition of ABPs for functions $f : \mathbb{Z}_p^{n_0} \to \mathbb{Z}_p^{n_1}$ by evaluating each output in a coordinate-wise manner and denote such a function class by $\mathcal{F}^{\mathsf{ABP}}_{n_0,n_1}$.*

There exists a linear-time algorithm that converts any boolean formula, boolean branching program or arithmetic formula to an arithmetic branching program with a constant blow-up in the representation size, so ABPs can be considered as a stronger computational model than the others.

A.2 DDFE for All-or-Nothing Encapsulation

Definition 13 (All-or-Nothing Encapsulation [CDSG+20]). AoNE *is defined on messages of length L and label space $\mathcal{L}$ as follows:*

$$\mathcal{K} = \emptyset \qquad\qquad \mathcal{M} = \{0,1\}^L \times \mathcal{S}(\mathcal{PK}) \times \mathcal{L}$$

Then, $F(\epsilon_K, (\mathsf{pk}, (x_{\mathsf{pk}}, \mathcal{U}_{\mathsf{pk}}, \ell_{\mathsf{pk}}))_{\mathsf{pk}\in\mathcal{U}}) = (\mathcal{U}_{\mathsf{pk}}, \ell_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}$ for any $\mathcal{U} \in \mathcal{L}(\mathcal{PK})$ and

$$F(\epsilon_K, (\mathsf{pk}, m_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M}) = \begin{cases} (\mathsf{pk}, x_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M} & \text{if condition } (*) \\ \bot & \text{otherwise.} \end{cases}$$

and AoNE *condition $(*)$ is: $\exists \ell \in \mathcal{L}, \forall \mathsf{pk} \in \mathcal{U}_M, m_{\mathsf{pk}} = (x_{\mathsf{pk}}, \mathcal{U}_M, \ell)$.*

We note that a concrete AoNE construction that is secure in the standard model is provided in [CDSG+20].

A.3 Pseudorandom Functions (PRF)

Definition 14 (PRF). *A* PRF *from input space $\mathcal{X}$ to output space $\mathcal{Y}$ is secure if for any security parameter $\lambda \in \mathbb{N}$, and for every PPT adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\mathsf{Adv}_{\mathsf{PRF},\mathcal{A}}(\lambda) := \left| \mathrm{P}\left[\mathcal{A}^{\mathcal{O}^b_{\mathsf{PRF}}(\cdot)}(1^\lambda) = b \;\middle|\; \begin{array}{l} K \xleftarrow{\$} \{0,1\}^\lambda, b \xleftarrow{\$} \{0,1\} \\ \forall \ell \in \mathcal{X}: \\ \quad \mathcal{O}^0_{\mathsf{PRF}}(\ell) := \mathsf{PRF}_K(\ell) \\ \quad \mathcal{O}^1_{\mathsf{PRF}}(\ell) := \mathsf{RF}(\ell) \end{array} \right] - \frac{1}{2} \right| \le \mathsf{negl}(\lambda)$$

where $\mathcal{O}^b_{\mathsf{PRF}}(\cdot)$ is an oracle depending on the challenge bit b and RF *is a random function computed on the fly.*

A.4 Non-Interactive Key Exchange (NIKE)

Definition 15 (Non-Interactive Key Exchange). *A* NIKE *scheme consists of three PPT algorithms:*

- $\mathsf{SetUp}(\lambda)$: *On input a security parameter λ, it outputs public parameters* pp. *Those parameters are implicit arguments to all the other algorithms;*
- $\mathsf{KeyGen}()$*: It generates and outputs a party's public key* $\mathsf{pk} \in \mathcal{PK}$ *and the corresponding secret key* $\mathsf{sk}_{\mathsf{pk}}$*;*
- $\mathsf{SharedKey}(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}'})$*: On input a public key and a secret key corresponding to a different public key, it deterministically outputs a shared key K.*

Correctness. For all security parameters $\lambda \in \mathbb{N}$, it holds that

$$\Pr[\mathsf{SharedKey}(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}'}) = \mathsf{SharedKey}(\mathsf{pk}', \mathsf{sk}_{\mathsf{pk}})] = 1,$$

where the probability is taken over $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$*,* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) \leftarrow \mathsf{KeyGen}()$*,* $(\mathsf{pk}', \mathsf{sk}'_{\mathsf{pk}}) \leftarrow \mathsf{KeyGen}()$*.*

Definition 16 (Security for NIKE). *No adversary A should be able to win the following security game against a challenger $\mathcal{C}$, with unlimited and adaptive access to the oracles* QNewHon*,* QRev*,* QTest*, and* QCor *described below:*

- *Initialize: the challenger runs the setup algorithm* $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$ *and chooses a random bit* $b \xleftarrow{\$} \{0,1\}$*. It initializes the set $\mathcal{H}$ of honest participants to $\emptyset$. It provides* pp *to the adversary $\mathcal{A}$;*
- *Participant creation queries* QNewHon()*: it generates* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) \leftarrow \mathsf{KeyGen}()$*, stores the association* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}})$ *in the set $\mathcal{H}$ of honest keys, and returns* pk *to the adversary;*
- *Reveal queries* $\mathsf{QRev}(\mathsf{pk}, \mathsf{pk}')$ *: it requires at least one of* pk *and* pk' *be in $\mathcal{H}$. Assume* $\mathsf{pk} \in \mathcal{H}$*, then it returns* $\mathsf{SharedKey}(\mathsf{pk}', \mathsf{sk}_{\mathsf{pk}})$*;*
- *Test queries* $\mathsf{QTest}(\mathsf{pk}, \mathsf{pk}')$*: it requires that both* pk *and* pk' *were generated via* QNewHon*.*
  - *if $b = 0$, it returns* $\mathsf{SharedKey}(\mathsf{pk}', \mathsf{sk}_{\mathsf{pk}})$*;*
  - *if $b = 1$, it returns a (uniformly) random value and stores the value so it can consistently answer further queries to* $\mathsf{QTest}(\mathsf{pk}, \mathsf{pk}')$ *or* $\mathsf{QTest}(\mathsf{pk}', \mathsf{pk})$*.*
- *Corruption queries* $\mathsf{QCor}(\mathsf{pk})$*: it recovers the secret key* sk *associated to* pk *from $\mathcal{H}$ and returns it, then removes* $(\mathsf{pk}, \mathsf{sk})$ *from $\mathcal{H}$. If* pk *is not associated with any secret key (i.e. it is not in $\mathcal{H}$), then nothing is returned;*
- *Finalize: $\mathcal{A}$ provides its guess b' on the bit b, and this procedure outputs the result β according to the analysis given below.*

Finalize outputs the bit $\beta = (b' = b)$ unless a QCor *query was made for any public key which was involved in a query to* QTest*, or a* QRev *query was made for a pair of public keys which was also involved in a* QTest *query, in which case a random bit β is returned. We say* NIKE *is secure if given any parameter $\lambda \in \mathbb{N}$, for every PPT adversary $\mathcal{A}$, the following holds:*

$$\mathsf{Adv}_{\mathsf{NIKE}}(\mathcal{A}) = |\Pr[\beta = 1] - 1/2| \leq \mathsf{negl}(\lambda).$$

B Deferred Proofs

B.1 Updatable Zero Sharing

Lemma 5 (UZS: Transition from $\mathbf{G}_0$ to $\mathbf{G}_1$). *For any PPT adversary $\mathcal{A}$, the advantage in distinguishing two games is*

$$|\mathsf{Adv}_0 - \mathsf{Adv}_1| \leq \mathsf{Adv}_{\mathsf{NIKE}}.$$

Proof. We build an adversary $\mathcal{B}$ against the IND security of NIKE from an adversary $\mathcal{A}$ that distinguishes between $\mathbf{G}_0$ and $\mathbf{G}_1$. To simulate a UZS challenger, $\mathcal{B}$ uses the NIKE oracles as follows:

- For every QNewHon() query, $\mathcal{B}$ returns pk from the NIKE.QNewHon() query to $\mathcal{A}$.
- For every QCor(pk) query, $\mathcal{B}$ obtains $\mathsf{NIKE.sk}_{\mathsf{pk}}$ from the NIKE.QCor(pk) query, and computes $k_{\mathsf{pk},\mathsf{pk}'} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{pk}', \mathsf{NIKE.sk}_{\mathsf{pk}})$ for all $\mathsf{pk}' \in \mathcal{PK}$ to complete the reply to $\mathcal{A}$.
- Instead of generating by itself the keys $(k_{\mathsf{pk},\mathsf{pk}'})_{(\mathsf{pk},\mathsf{pk}')\in\mathcal{H}^2, \mathsf{pk}\neq\mathsf{pk}'}$, the adversary $\mathcal{B}$ uses the challenge shared keys $k_{\mathsf{pk},\mathsf{pk}'}$ from the $\mathsf{NIKE.QTest}(\mathsf{pk}, \mathsf{pk}')$ queries.
- $\mathcal{B}$ outputs $\mathcal{A}$'s guess for the challenge bit $\mathsf{NIKE}.b$.

The admissibility condition $(*)$ of NIKE holds since the set of corrupted users in UZS is sent in one shot. Therefore, when $\mathsf{NIKE}.b = 0$, $\mathcal{B}$ is playing $\mathbf{G}_0$; when $\mathsf{NIKE}.b = 1$, $\mathcal{B}$ is playing $\mathbf{G}_1$. □

Lemma 6 (UZS: Transition from $\mathbf{G}_1$ to $\mathbf{G}_2$). *For any PPT adversary $\mathcal{A}$, the advantage in distinguishing two games is*

$$|\mathsf{Adv}_1 - \mathsf{Adv}_2| \leq \binom{q_{\mathsf{QNewHon}} - q_{\mathsf{QCor}}}{2} \cdot \mathsf{Adv}_{\mathsf{PRF}}.$$

Proof. We proceed by using multiple hybrid games over each pair of different honest parties $(\mathsf{pk}, \mathsf{pk}') \in \mathcal{H}^2$. For each transition, we build an adversary $\mathcal{B}$ against the IND security of PRF from an adversary $\mathcal{A}$ that distinguishes two games in the transition. To simulate a UZS challenger, $\mathcal{B}$ uses the PRF oracle, instead of generating by itself the PRF key $k_{\mathsf{pk},\mathsf{pk}'}$ to handle all $\mathsf{PRF}_{k_{\mathsf{pk},\mathsf{pk}'}}$ related operations, and finally outputs $\mathcal{A}$'s guess for the challenge bit $\mathsf{PRF}.b$. Since the number of honest pairs is bounded by $\binom{q_{\mathsf{QNewHon}} - q_{\mathsf{QCor}}}{2}$, one completes the proof. □

```
$\mathbf{G}_0$, $\boxed{\mathbf{G}_1,\ \mathbf{G}_2,\ \boxed{\mathbf{G}_3}}$:
$\mathsf{pp} \leftarrow \mathsf{SetUp}(1^\lambda)$;    $b \xleftarrow{\$} \{0,1\}$
$(\mathcal{PK}, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$;    $(\mathsf{pk})_{\mathsf{pk}\in\mathcal{PK}} \leftarrow \mathsf{QNewHon}([\mathcal{PK}])$
$(\mathcal{C}, \mathsf{st}_2) \leftarrow \mathcal{A}((\mathsf{pk})_{\mathsf{pk}\in\mathcal{PK}}, \mathsf{st}_1)$;    $(\mathsf{sk}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{C}} \leftarrow \mathsf{QCor}([\mathcal{C}])$
$b' \leftarrow \mathcal{A}^{\mathsf{QSeedGen}(\cdot,\cdot,\cdot),\mathsf{QTokGen}(\cdot,\cdot,\cdot),\mathsf{QShare}(\cdot,\cdot)}((\mathsf{sk}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{C}}, \mathsf{st}_2)$
Output: $b'$ if Condition $(*)$ is satisfied, or $b' \xleftarrow{\$} \{0,1\}$ otherwise.

$\mathsf{QNewHon}()$:
$(\mathsf{pk}, \mathsf{NIKE.sk}_{\mathsf{pk}}) \leftarrow \mathsf{NIKE.KeyGen}()$
$\forall \mathsf{pk}' \in \mathcal{PK} : k_{\mathsf{pk},\mathsf{pk}'} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{pk}', \mathsf{sk}_{\mathsf{pk}})$
$\boxed{\text{For } (\mathsf{pk}_1, \mathsf{pk}_2) \in \mathcal{H}^2 \text{ and } \mathsf{pk}_1 \neq \mathsf{pk}_2 : k_{\mathsf{pk}_1,\mathsf{pk}_2} \xleftarrow{\$} \mathbb{Z}_p}$
Store $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) = (\mathsf{pk}, \mathsf{NIKE.sk}_{\mathsf{pk}}, (k_{\mathsf{pk},\mathsf{pk}'})_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}})$
Return $\mathsf{pk}$

$\mathsf{QCor}(\mathsf{pk})$:
If $\mathsf{pk} \notin \mathcal{PK}$, return $\perp$.
Return $\mathsf{sk}_{\mathsf{pk}} = (\mathsf{NIKE.sk}_{\mathsf{pk}}, (k_{\mathsf{pk},\mathsf{pk}'})_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}})$.

$\mathsf{QSeedGen}(\mathsf{pk}, \mathcal{U}, \ell_s)$:
$\boldsymbol{a}^{\ell_s}_{\mathsf{pk},\mathcal{U}} = \left((-1)^{\mathsf{pk}<\mathsf{pk}'}\mathsf{PRF}_{k_{\mathsf{pk},\mathsf{pk}'}}(\text{"}s\text{"}||\mathcal{U}||\ell_s)\right)_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}}$
$\boxed{\boldsymbol{a}^{\ell_s}_{\mathsf{pk},\mathcal{U}} = \left((-1)^{\mathsf{pk}<\mathsf{pk}'}[\mathsf{PRF}_{k_{\mathsf{pk},\mathsf{pk}'}}/\mathsf{RF}_{\mathsf{pk},\mathsf{pk}'}]^{(\mathsf{pk},\mathsf{pk}')\in\mathcal{H}^2}(\text{"}s\text{"}||\mathcal{U}||\ell_s)\right)_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}}}$
Return $\mathsf{seed}^{\ell_s}_{\mathsf{pk},\mathcal{U}} = [\boldsymbol{a}^{\ell_s}_{\mathsf{pk},\mathcal{U}}]$.

$\mathsf{QTokGen}(\mathsf{pk}, \mathcal{U}, \ell_u)$:
$\boldsymbol{b}^{\ell_u}_{\mathsf{pk},\mathcal{U}} = \left(\mathsf{PRF}_{k_{\mathsf{pk},\mathsf{pk}'}}(\text{"}u\text{"}||\mathcal{U}||\ell_u)\right)_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}}$
$\boxed{\boldsymbol{b}^{\ell_u}_{\mathsf{pk},\mathcal{U}} = \left([\mathsf{PRF}_{k_{\mathsf{pk},\mathsf{pk}'}}/\mathsf{RF}_{\mathsf{pk},\mathsf{pk}'}]^{(\mathsf{pk},\mathsf{pk}')\in\mathcal{H}^2}(\text{"}u\text{"}||\mathcal{U}||\ell_u)\right)_{\mathsf{pk}'\in\mathcal{U}\setminus\{\mathsf{pk}\}}}$
Return $\mathsf{token}^{\ell_u}_{\mathsf{pk},\mathcal{U}} = \boldsymbol{b}^{\ell_u}_{\mathsf{pk},\mathcal{U}}$.

$\mathsf{QShare}(\mathcal{U}, \ell)$:
Return $(\mathsf{share}^{\ell}_{\mathsf{pk},\mathcal{U}})_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} \leftarrow [\mathsf{PShareGen}(\mathcal{H}\cap\mathcal{U}, \mathcal{U}, \ell)/\mathcal{R}^{\ell}_{\mathcal{H}\cap\mathcal{U},\mathcal{U}}]^b$
$\boxed{(\mathsf{share}^{\ell}_{\mathsf{pk},\mathcal{U}})_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} \leftarrow \mathcal{R}^{\ell}_{\mathcal{H}\cap\mathcal{U},\mathcal{U}}}$
```

Fig. 12. Games for the otu-sta-IND proof of UZS in Theorem 1

B.2 Function-Hiding Inner-Product DDFE

```
$\mathbf{G}_0$, $\boxed{\mathbf{G}_1, \boxed{\mathbf{G}_3}}$:
$\mathsf{pp} \leftarrow \mathsf{SetUp}(1^\lambda)$;    $b \overset{\$}{\leftarrow} \{0,1\}$
$(\mathcal{PK}, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$;    $(\mathsf{pk})_{\mathsf{pk} \in \mathcal{PK}} \leftarrow \mathsf{QNewHon}([\mathcal{PK}])$
$(\mathcal{C}, \mathsf{st}_2) \leftarrow \mathcal{A}((\mathsf{pk})_{\mathsf{pk} \in \mathcal{PK}}, \mathsf{st}_1)$;    $(\mathsf{sk}_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{C}} \leftarrow \mathsf{QCor}([\mathcal{C}])$
$b' \leftarrow \mathcal{A}^{\mathsf{QEnc}(\cdot,\cdot,\cdot,\cdot),\mathsf{QDKGen}(\cdot,\cdot,\cdot,\cdot)}((\mathsf{sk}_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{C}}, \mathsf{st}_2)$
Output $b'$ if Condition $(*)$ is satisfied, or $b' \leftarrow \{0,1\}$ otherwise.

$\mathsf{QNewHon}()$:
$k_{\mathsf{pk}} \leftarrow \mathcal{K}_{\mathsf{PRF}}$;    $(\mathsf{upk}, \mathsf{usk}_{\mathsf{pk}}) \leftarrow \mathsf{uKeyGen}()$;    $(\mathsf{apk}, \mathsf{ask}_{\mathsf{pk}}) \leftarrow \mathsf{aKeyGen}()$
$\mathsf{pk} = (\mathsf{upk}, \mathsf{apk})$;    $\mathsf{sk}_{\mathsf{pk}} = (k_{\mathsf{pk}}, \mathsf{usk}_{\mathsf{pk}}, \mathsf{ask}_{\mathsf{pk}})$
Store $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}})$ and return $\mathsf{pk}$.

$\mathsf{QDKGen}(\mathsf{pk}, \boldsymbol{y}^0 = (\boldsymbol{k}^0, \mathbf{0}^d), \boldsymbol{y}^1 = (\boldsymbol{k}^1, \mathbf{0}^d), (\mathcal{U}, \ell))$:
$[\boldsymbol{a}]_2 \leftarrow \mathsf{uSeedGen}(\mathsf{usk}_{\mathsf{pk}}, (\mathcal{U}, \ell))$
$\mathsf{coin} \leftarrow \mathsf{PRF}_{k_{\mathsf{pk}}}(\mathcal{U})$    $\boxed{\mathsf{coin} \leftarrow \mathsf{RF}_{\mathsf{pk}}(\mathcal{U})\ \forall \mathsf{pk} \in \mathcal{H}}$
$\mathsf{isk} \leftarrow \mathsf{iKeyGen}(1^{6d+|\mathcal{U}|}; \mathsf{coin})$
$[\boldsymbol{g}]_2 = [(\boldsymbol{y}^b, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{a}, 0)]_2$    $\boxed{[\boldsymbol{g}]_2 = [(\boldsymbol{y}^0, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{a}, 0)]_2}$
$\mathsf{idk} \leftarrow \mathsf{iDKGen}(\mathsf{isk}, [\boldsymbol{g}]_2)$
$\mathsf{act} \leftarrow \mathsf{aEnc}(\mathsf{ask}_{\mathsf{pk}}, (\mathsf{idk}, \mathcal{U}, \ell \| \text{"}dk\text{"}))$
Return $(\mathsf{act}, \mathcal{U}, \ell)$.

$\mathsf{QEnc}(\mathsf{pk}, \boldsymbol{x}^0 = (\boldsymbol{m}^0, \mathbf{0}^d), \boldsymbol{x}^1 = (\boldsymbol{m}^1, \mathbf{0}^d), (\mathcal{U}, \ell))$:
$\boldsymbol{b} \leftarrow \mathsf{uTokGen}(\mathsf{usk}_{\mathsf{pk}}, (\mathcal{U}, \ell))$
$\mathsf{coin} \leftarrow \mathsf{PRF}_{k_{\mathsf{pk}}}(\mathcal{U})$    $\boxed{\mathsf{coin} \leftarrow \mathsf{RF}_{\mathsf{pk}}(\mathcal{U})\ \forall \mathsf{pk} \in \mathcal{H}}$
$\mathsf{isk} \leftarrow \mathsf{iKeyGen}(1^{6d+|\mathcal{U}|}; \mathsf{coin})$
$[\boldsymbol{h}]_1 = [(\boldsymbol{x}^b, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{b}, 0)]_1$    $\boxed{[\boldsymbol{h}]_1 = [(\boldsymbol{x}^0, \mathbf{0}^{2d}, \mathbf{0}^{2d}, \boldsymbol{b}, 0)]_1}$
$\mathsf{ict} \leftarrow \mathsf{iEnc}(\mathsf{isk}, [\boldsymbol{h}]_1)$
$\mathsf{act} \leftarrow \mathsf{aEnc}(\mathsf{ask}_{\mathsf{pk}}, (\mathsf{ict}_{\mathsf{pk}}, \mathcal{U}, \ell \| \text{"}ct\text{"}))$
Return $(\mathsf{act}, \mathcal{U}, \ell)$.

$\mathsf{QCor}(\mathsf{pk})$:
If $\mathsf{pk} \notin \mathcal{PK}$, return $\perp$.
Return $\mathsf{sk}_{\mathsf{pk}} = (k_{\mathsf{pk}}, \mathsf{usk}_{\mathsf{pk}}, \mathsf{ask}_{\mathsf{pk}})$.
```

Fig. 13. Games for the sta-sym-IND proof of FH-IP-DDFE in Theorem 2

The case of complete queries. The deferred lemmas for transitions are provided below.

Lemma 7 (FH-IP-DDFE: **Transition from $\mathbf{G}_0$ to $\mathbf{G}_1$).** *For any PPT adversary $\mathcal{A}$, the advantage in distinguishing two games is*

$$|\mathsf{Adv}_{\mathbf{G}_0} - \mathsf{Adv}_{\mathbf{G}_1}| \leq (q_{\mathsf{QNewHon}} - q_{\mathsf{QCor}}) \cdot \mathsf{Adv}_{\mathsf{PRF}}.$$

Proof. We proceed by using multiple hybrid games for each $\mathsf{pk} \in \mathcal{H}$. For each transition, we build an adversary $\mathcal{B}$ against the IND security of PRF from an adversary $\mathcal{A}$ that distinguishes two games in the transition. To simulate a FH-IP-DDFE challenger, $\mathcal{B}$ uses the PRF oracle, instead of generating by itself the PRF key k_{pk} to handle all $\mathsf{PRF}_{k_{\mathsf{pk}}}$ related operations, and finally outputs $\mathcal{A}$'s guess for the challenge bit $\mathsf{PRF}.b$. Since the number of honest parties is bounded by $(q_{\mathsf{QNewHon}} - q_{\mathsf{QCor}})$, one completes the proof. ◻

Lemma 8 (FH-IP-DDFE: **Transitions to $\mathbf{G}_{1.1}$, $\mathbf{G}^\star_{\ell_M.1}$, $\mathbf{G}^\star_{\ell_M.2}$, $\mathbf{G}^\star_{\ell_M.4}$, $\mathbf{G}^\star_{\ell_M.6}$, $\mathbf{G}_2$, $\mathbf{G}_3$ from their previous games).** *For any PPT adversary $\mathcal{A}$, the advantage in distinguishing between any two games is upper-bounded by*

$$(q_{\mathsf{QEnc}} + q_{\mathsf{QDKGen}}) \cdot \mathsf{Adv}^{\mathsf{sym}}_{\mathsf{IPE}}.$$

Proof. We proceed by using multiple hybrid games for each pair of honest $(\mathsf{pk}, \mathcal{U}) \in \mathcal{Q}_M \cup \mathcal{Q}_K$. We build an adversary $\mathcal{B}$ against the (adaptive) sym-IND security of IPE from an adversary $\mathcal{A}$ that distinguishes between two games in the considered transition. To simulate a $\mathsf{FH\text{-}IP\text{-}DDFE}$ challenger, $\mathcal{B}$ uses the IPE oracles to handle all IPE related operations for the reply of each $(\mathsf{pk}, \mathcal{U})$-involved query from $\mathcal{A}$.

– For each complete $\mathsf{QDKGen}(\mathsf{pk}, \boldsymbol{y}^0, \boldsymbol{y}^1, \mathcal{U}, \ell_K)$ query, let $\boldsymbol{g}^0_{\mathsf{pk},\mathcal{U}}$ and $\boldsymbol{g}^1_{\mathsf{pk},\mathcal{U}}$ be the IPE keys that $\mathcal{B}$ needs to prepare in the previous game and the subsequent game respectively. Then $\mathcal{B}$ sends $(\boldsymbol{g}^0_{\mathsf{pk},\mathcal{U}}, \boldsymbol{g}^1_{\mathsf{pk},\mathcal{U}})$ to IPE decryption key generation oracle. It uses the returned decryption key $\mathsf{idk}_{\mathsf{pk}}$ to complete the reply to $\mathcal{A}$.
– For each complete $\mathsf{QEnc}(\mathsf{pk}, \boldsymbol{x}^0, \boldsymbol{x}^1, \mathcal{U}, \ell_M)$ query, let $\boldsymbol{h}^0_{\mathsf{pk},\mathcal{U}}$ and $\boldsymbol{h}^1_{\mathsf{pk},\mathcal{U}}$ be the IPE messages that $\mathcal{B}$ needs to prepare in the previous game and the subsequent game respectively. Then $\mathcal{B}$ sends $(\boldsymbol{h}^0_{\mathsf{pk},\mathcal{U}}, \boldsymbol{h}^1_{\mathsf{pk},\mathcal{U}})$ to IPE encryption oracle. It uses the returned ciphertext $\mathsf{ict}_{\mathsf{pk}}$ to complete the reply to $\mathcal{A}$.
– $\mathcal{B}$ outputs $\mathcal{A}$'s guess for the challenge bit $\mathsf{IPE}.b$.

The admissibility condition $(*)$ of IPE in each transition holds since one always has ${\boldsymbol{g}^0_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{h}^0_{\mathsf{pk},\mathcal{U}} = {\boldsymbol{g}^1_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{h}^1_{\mathsf{pk},\mathcal{U}}$. Proceeding for each $(\mathsf{pk}, \mathcal{U})$ in queries where $\mathsf{pk} \in \mathcal{H}$, one completes the lemma. $\square$

The case of incomplete queries. We first remark that as all transitions in the case of complete queries do not need to exploit the structure of merged messages and keys, namely $\boldsymbol{x} = (\boldsymbol{m}, \mathbf{0}^d)$ and $\boldsymbol{y} = (\boldsymbol{k}, \mathbf{0}^d)$, then the indistinguishability of complete queries DDFE also holds for arbitrary $2d$-length message $\boldsymbol{x}$ and key $\boldsymbol{y}$. We use this remark for the following hybrid argument.

Game $\mathbf{G}^{\mathsf{in}}_1$: This game corresponds to $\mathbf{G}_1$, where for each $\mathsf{pk} \in \mathcal{H} \cap \mathcal{U}$, the challenger encrypts $\boldsymbol{x} = (\boldsymbol{m}^b, \mathbf{0}^d)$ to answer any query on $(\mathsf{pk}, \boldsymbol{m}^0, \boldsymbol{m}^1, \mathcal{U}, \ell_M)$ and generates decryption keys for $\boldsymbol{y} = (\boldsymbol{k}^b, \mathbf{0}^d)$ to answer any query on $(\mathsf{pk}, \boldsymbol{k}^0, \boldsymbol{k}^1, \mathcal{U}, \ell_K)$.
Game $\mathbf{G}^{\mathsf{in}}_2$: The change is that the challenger encrypts $\boldsymbol{x} = (\boldsymbol{m}^b, \boldsymbol{m}^0)$.
Game $\mathbf{G}^{\mathsf{in}}_3$: The change is that the challenger generates decryption keys for $\boldsymbol{y} = (\mathbf{0}^d, \boldsymbol{k}^0)$.
Game $\mathbf{G}^{\mathsf{in}}_4$: The change is that the challenger encrypts $\boldsymbol{x} = (\mathbf{0}^d, \boldsymbol{m}^0)$. As $\boldsymbol{y} = (\mathbf{0}^d, \boldsymbol{k}^0)$ in this game, there is no more challenge bit b, so the adversary's advantage is 0. which is identical to that in $\mathbf{G}_3$.

Inspired by the proof strategy relying on AoNE and complete-queries DDFE in [SV23], we use the following intermediate games for the transition from $\mathbf{G}^{\mathsf{in}}_1$ to $\mathbf{G}^{\mathsf{in}}_2$. For every $(\mathcal{U}_M, \ell_M) \in \mathcal{Q}_M$,

Games $\mathbf{G}^{\mathsf{in}}_{1.(\mathcal{U}_M,\ell_M)}$: In this game, to answer any query on $(\mathsf{pk}, \boldsymbol{m}^0, \boldsymbol{m}^1, \mathcal{U}, \ell)$ the challenger encrypts $\boldsymbol{x} = (\boldsymbol{m}^b, \boldsymbol{m}^0)$ if $(\mathcal{U}, \ell) < (\mathcal{U}_M, \ell_M)$ and $\boldsymbol{x} = (\boldsymbol{m}^b, \mathbf{0}^d)$ if $(\mathcal{U}, \ell) \geq (\mathcal{U}_M, \ell_M)$. The only change between games of adjacent pairs $(\mathcal{U}_M, \ell_M)$ and $(\mathcal{U}_M, \ell_M) + 1$ is the selection of message $\boldsymbol{x}$ to encrypt for encryption queries on $(\mathsf{pk}, \boldsymbol{m}^0, \boldsymbol{m}^1, \mathcal{U}_M, \ell_M)$.

An adversary $\mathcal{A}$ that distinguishes between two games can be used by an adversary $\mathcal{B}$ to play against either DDFE or AoNE as follows.

– $\mathcal{B}$ first guesses if $\mathcal{A}$ sends complete queries at $(\mathcal{U}_M, \ell_M)$ or not. If yes, it starts a sta-sym-IND game of DDFE for complete queries. If no, it starts an sym-IND game of AoNE. If in the end the guess is wrong, it aborts the underlying security game. In both games, $\mathcal{B}$ receives encryption queries $(\mathsf{pk}, \boldsymbol{m}^0, \boldsymbol{m}^1, \mathcal{U}, \ell)$ from $\mathcal{A}$ and prepares challenge messages and keys as follows:
  - $\boldsymbol{y} = \boldsymbol{y}^0 = \boldsymbol{y}^1 = (\boldsymbol{k}^b, \mathbf{0}^d)$;
  - $\boldsymbol{x}^0 = \boldsymbol{x}^1 = (\boldsymbol{m}^b, \boldsymbol{m}^0)$ if $(\mathcal{U}, \ell) < (\mathcal{U}_M, \ell_M)$;
  - $\boldsymbol{x}^0 = \boldsymbol{x}^1 = (\boldsymbol{m}^b, \mathbf{0}^d)$ if $(\mathcal{U}, \ell) > (\mathcal{U}_M, \ell_M)$;
  - $\boldsymbol{x}^0 = (\boldsymbol{m}^b, \mathbf{0}^d)$ and $\boldsymbol{x}^1 = (\boldsymbol{m}^b, \boldsymbol{m}^0)$ if $(\mathcal{U}, \ell) = (\mathcal{U}_M, \ell_M)$.
– If $\mathcal{B}$ is playing against DDFE, then $\mathcal{B}$ sends the queries $(\mathsf{pk}, \boldsymbol{x}^0, \boldsymbol{x}^1, \mathcal{U}, \ell)$ and $(\mathsf{pk}, \boldsymbol{y}, \boldsymbol{y}, \mathcal{U}, \ell)$ to DDFE oracles. Note that if encryption queries on $(\mathcal{U}, \ell) \neq (\mathcal{U}_M, \ell_M)$ and decryption-key queries sent by $\mathcal{A}$ are not complete, $\mathcal{B}$ can complete them by sending itself queries for identical messages and keys on missing $\mathsf{pk} \in \mathcal{U} \cap \mathcal{H}$ to DDFE oracles. If $\mathcal{A}$ sends complete queries at $(\mathcal{U}_M, \ell_M)$, $\mathcal{B}$ will be an adversary against DDFE with all complete queries.
– If $\mathcal{B}$ is playing against AoNE, then $\mathcal{B}$ uses IPE to encrypt $(\boldsymbol{x}^0, \boldsymbol{x}^1)$ into $(\mathsf{ict}^0, \mathsf{ict}^1)$, and sends $(\mathsf{pk}, \mathsf{ict}^0, \mathsf{ict}^1, \mathcal{U}, \ell \| "ct")$ to AoNE encryption oracle. For decryption-key queries, it uses IPE to generate keys idk for $\boldsymbol{y}$ and sends $(\mathsf{pk}, \mathsf{idk}, \mathsf{idk}, \mathcal{U}, \ell \| "dk")$ to AoNE decryption-key oracle. If $\mathcal{A}$ sends incomplete queries at $(\mathcal{U}_M, \ell_M)$, $\mathcal{B}$ will be an admissible adversary against AoNE.

The advantage of $\mathcal{A}$ between each pair of games $\mathbf{G}^{\mathsf{in}}_{1.(\mathcal{U}_M,\ell_M)}$ and $\mathbf{G}^{\mathsf{in}}_{1.(\mathcal{U}_M,\ell_M)+1}$ is then upper-bounded by the maximum of best advantages in complete-queries DDFE and AoNE, which is negligible. We apply a similar strategy of using complete-queries DDFE and AoNE to the transitions between $\mathbf{G}^{\mathsf{in}}_2$, $\mathbf{G}^{\mathsf{in}}_3$, and $\mathbf{G}^{\mathsf{in}}_4$.

C Attribute-Weighted-Sum DDFE

Correctness. From the scheme in Figure 11, one can always recover AWIPE ciphertexts $(\mathsf{aict}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}$ and decryption keys $(\mathsf{aidk}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}$ from AoNE. The correctness is then implied by the correctness of the AWIPE scheme and the UZS scheme:

$$\begin{aligned}
&\sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{aiDec}(\mathsf{aict}_{\mathsf{pk}}, \mathsf{aidk}_{\mathsf{pk}})\\
=&\sum_{\mathsf{pk}\in\mathcal{U}} [\sum_{j\in[N_{\mathsf{pk}}]} f_{\mathsf{pk}}(\boldsymbol{x}_{\mathsf{pk},j})^\top \cdot \boldsymbol{z}_{\mathsf{pk},j} + {\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}]_T\\
=&[\sum_{\mathsf{pk}\in\mathcal{U}} \sum_{j\in[N_{\mathsf{pk}}]} f_{\mathsf{pk}}(\boldsymbol{x}_{\mathsf{pk},j})^\top \cdot \boldsymbol{z}_{\mathsf{pk},j}]_T\\
&+e\left([1]_1, \sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{uShareEval}(\mathsf{uSeedUpt}(\mathsf{seed}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}, \mathsf{token}^{\ell_M}_{\mathsf{pk},\mathcal{U}}))\right)\\
=&[\sum_{\mathsf{pk}\in\mathcal{U}} \sum_{j\in[N_{\mathsf{pk}}]} f_{\mathsf{pk}}(\boldsymbol{x}_{\mathsf{pk},j})^\top \cdot \boldsymbol{z}_{\mathsf{pk},j}]_T + e\left([1]_1, \sum_{\mathsf{pk}\in\mathcal{U}} \mathsf{share}^{\ell_{\boldsymbol{f}}||\ell_M}_{\mathsf{pk},\mathcal{U}}\right)\\
=&[\sum_{\mathsf{pk}\in\mathcal{U}} \sum_{j\in[N_{\mathsf{pk}}]} f_{\mathsf{pk}}(\boldsymbol{x}_{\mathsf{pk},j})^\top \cdot \boldsymbol{z}_{\mathsf{pk},j}]_T + e([1]_1, [0]_2)\\
=&[\sum_{\mathsf{pk}\in\mathcal{U}} \sum_{j\in[N_{\mathsf{pk}}]} f_{\mathsf{pk}}(\boldsymbol{x}_{\mathsf{pk},j})^\top \cdot \boldsymbol{z}_{\mathsf{pk},j}]_T.
\end{aligned}$$

□

Remark 6 (Size of Ciphertext/Decryption Key). In the AWS-DDFE construction (Figure 11), if one uses the AWSw/IP-FE that is constructed based on the MDDH_k assumption as in [ATY23] and the AoNE that is constructed from a rate-1 identity-based encryption and employed in the hybrid-encryption mode as in [CDSG+20], then the complexity for the size each DDFE ciphertext will be $O_\lambda(N(kn_0 + n_1 + n))$, and the size of each DDEE decryption key will be $O_\lambda(t(kn_0 + n_1 + n))$. Here (t, n_0, n_1) are parameters related to ABPs in $\mathcal{F}^{\mathsf{ABP}}_{n_0,n_1}$.

Like FH-IP-DDFE, we have a remark from the admissibility of AWS-DDFE.

Remark 7 (Invariance in AWS-DDFE). Given any admissible adversary against AWS-DDFE and any challenge bit b, Condition $(*)$ in Definition 5 implies that among encryption queries under $(\ell_M,\mathcal{U})$ and decryption-key queries under $(\ell_{\boldsymbol{f}},\mathcal{U})$, the equalities

$$\begin{aligned}
&\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}\setminus\mathsf{pk}^\star} \sum_{j\in[N_{\mathsf{pk}}]} f_{\mathsf{pk}}(\boldsymbol{x}_{\mathsf{pk},j})^\top \boldsymbol{z}^b_{\mathsf{pk},j} + \sum_{j\in[N_{\mathsf{pk}^\star}]} f_{\mathsf{pk}^\star}(\boldsymbol{x}^\tau_{\mathsf{pk}^\star,j})^\top \boldsymbol{z}^{b,\tau}_{\mathsf{pk}^\star,j}\\
=&\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}\setminus\mathsf{pk}^\star} \sum_{j\in[N_{\mathsf{pk}}]} f_{\mathsf{pk}}(\boldsymbol{x}_{\mathsf{pk},j})^\top \boldsymbol{z}^0_{\mathsf{pk},j} + \sum_{j\in[N_{\mathsf{pk}^\star}]} f_{\mathsf{pk}^\star}(\boldsymbol{x}^\tau_{\mathsf{pk}^\star,j})^\top \boldsymbol{z}^{0,\tau}_{\mathsf{pk}^\star,j}
\end{aligned}$$

holds for each $\mathsf{pk}^\star \in \mathcal{H}\cap\mathcal{U}$ and each index τ that numerates repetitions of encryption queries under $(\mathsf{pk}^\star, \ell_M, \mathcal{U})$. Then one has the value

$$\begin{aligned}
\Delta^b_{\mathsf{pk}^\star} :=& \sum_{j\in[N_{\mathsf{pk}^\star}]} f_{\mathsf{pk}^\star}(\boldsymbol{x}^\tau_{\mathsf{pk}^\star,j})^\top \boldsymbol{z}^{0,\tau}_{\mathsf{pk}^\star,j} - \sum_{j\in[N_{\mathsf{pk}^\star}]} f_{\mathsf{pk}^\star}(\boldsymbol{x}^\tau_{\mathsf{pk}^\star,j})^\top \boldsymbol{z}^{b,\tau}_{\mathsf{pk}^\star,j}\\
=& \sum_{j\in[N_{\mathsf{pk}^\star}]} f_{\mathsf{pk}^\star}(\boldsymbol{x}^1_{\mathsf{pk}^\star,j})^\top \boldsymbol{z}^{0,1}_{\mathsf{pk}^\star,j} - \sum_{j\in[N_{\mathsf{pk}^\star}]} f_{\mathsf{pk}^\star}(\boldsymbol{x}^1_{\mathsf{pk}^\star,j})^\top \boldsymbol{z}^{b,1}_{\mathsf{pk}^\star,j}
\end{aligned}$$

be an invariance across τ. Furthermore, each variance is an additive share of zero, i.e. $\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} \Delta^b_{\mathsf{pk}} = 0$.

*Proof (***Theorem 3***).* This is a selective-static game where any adversary has to send encryption queries $\mathcal{Q}_M$, decryption-key queries $\mathcal{Q}_K$ and corruption queries for $\mathcal{C} = \mathcal{PK} \setminus \mathcal{H}$ in one shot. For brevity, we use the notations $\hat{x} := (\boldsymbol{x}_j, \boldsymbol{z}_j)_{j\in[N]}$ and $\hat{f}$ for each ABP function f such that $\hat{f}(\hat{x}) := \sum_{j\in[N]} f(\boldsymbol{x}_j)^\top \boldsymbol{z}_j$. We proceed via a hybrid argument: we describe the global changes in the IND game by using the games $\mathbf{G}_0$, $\mathbf{G}_1$, $\mathbf{G}_2$ and $\mathbf{G}_3$ that are described below; the transition between $\mathbf{G}_2$ and $\mathbf{G}_3$ requires intermediate games $(\mathbf{G}_{2.(\mathcal{U},\ell_M).i})_{i\in[5]}$ (see Figure 14) for each pair $(\mathcal{U}, \ell_M) \in \mathcal{Q}_M$. Notably, the game $\mathbf{G}_0$ corresponds to sel-sta-sym-IND security game, and the game $\mathbf{G}_3$ corresponds to the case where adversary's advantage is 0 since there is no challenge bit b.

Game $\mathbf{G}_1$: The change is that the challenger uses a random function $\mathsf{RF}_{\mathsf{pk}}$ instead of $\mathsf{PRF}_{k_{\mathsf{pk}}}$ for $\mathsf{pk} \in \mathcal{H}$. The indistinguishability is implied by the security of the pseudorandom functions.

Game $\mathbf{G}_2$: When $\mathsf{pk} \in \mathcal{H}$, a decryption key query $(\mathsf{pk}, (\hat{f}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_K}, \mathcal{U}_K) \in \mathcal{Q}_K$ is said to be incomplete if there exists $\mathsf{pk}' \in \mathcal{H} \cap \mathcal{U}_K$ such that there is no key query $(\mathsf{pk}', (\hat{f}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_K}, \mathcal{U}_K) \in \mathcal{Q}_K$. For that query, $\mathsf{act}_{\mathsf{pk}}$ is changed to the encapsulation of $(0, \mathcal{U}_K, \ell_{\boldsymbol{f}} || "dk")$. Similarly, when $\mathsf{pk} \in \mathcal{H}$, an encryption query $(\mathsf{pk}, \hat{x}^0_{\mathsf{pk}}, \hat{x}^1_{\mathsf{pk}}, \mathcal{U}_M, \ell_M) \in \mathcal{Q}_M$ is said to be incomplete if there exists $\mathsf{pk}' \in \mathcal{H} \cap \mathcal{U}_M$ such that there is no encryption query $(\mathsf{pk}', \hat{x}^0_{\mathsf{pk}'}, \hat{x}^1_{\mathsf{pk}'}, \mathcal{U}_M, \ell_M) \in \mathcal{Q}_M$. For that query, $\mathsf{act}_{\mathsf{pk}}$ is changed to the encapsulation of $(0, \mathcal{U}_M, \ell_M || "ct")$. The indistinguishability is implied by the security of the AoNE scheme.

Game $\mathbf{G}_3$: In this game, for every complete encryption query on the tuple $(\mathsf{pk}, \hat{x}^0_{\mathsf{pk}}, \hat{x}^1_{\mathsf{pk}}, \mathcal{U}_M, \ell_M)$, the challenger chooses $\hat{x}^0_{\mathsf{pk}}$ to encrypt.

Lemmas for transitions between the intermediate games $(\mathbf{G}_{2.(\mathcal{U},\ell_M).i})_{i\in[5]}$ (see Figure 14) for each pair $(\mathcal{U}, \ell_M) \in \mathcal{Q}_M$ are provided in the following section, which complete the transition from $\mathbf{G}_2$ to $\mathbf{G}_3$. Therefore, the theorem is complete. □

Lemma 9 (AWS-DDFE: **Transition from $\mathbf{G}_{2.(\mathcal{U},\ell_M).0}$ to $\mathbf{G}_{2.(\mathcal{U},\ell_M).1}$**). *For any PPT adversary $\mathcal{A}$, the advantage in distinguishing two games is*

$$\left|\mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).0}} - \mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).1}}\right| \leq (q_{\mathsf{QNewHon}} - q_{\mathsf{QCor}}) \cdot \mathsf{Adv}^{\text{sel-sym}}_{\mathsf{AWIPE}}.$$

Proof. On a fixed user set $\mathcal{U}$, we proceed by using multiple hybrid games for each $\mathsf{pk} \in \mathcal{H}$. We build an adversary $\mathcal{B}$ against the sel-sym security of AWIPE from an adversary $\mathcal{A}$ that distinguishes between two games in the transition. To simulate a AWS-DDFE challenger, $\mathcal{B}$ uses the AWIPE oracles to handle all AWIPE related operations for the reply of each $(\mathsf{pk}, \mathcal{U})$-involved query from $\mathcal{A}$.

– For each complete $\mathsf{QDKGen}(\mathsf{pk}, (\hat{f}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}, \mathcal{U})$ query, $\mathcal{B}$ prepares the AWIPE key as

$$k^0 = (\hat{f}_{\mathsf{pk}}, [\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}, 0]_2),$$
$$k^1 = (\hat{f}_{\mathsf{pk}}, [\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}, {\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}]_2)$$

where $\boldsymbol{f} = (\hat{f}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}$ and sends (pk, k^0, k^1) to the AWIPE decryption-key oracle. It uses the returned decryption key to complete the reply to $\mathcal{A}$.

– For each complete $\mathsf{QEnc}(\mathsf{pk}, \hat{x}^0_{\mathsf{pk}}, \hat{x}^1_{\mathsf{pk}}, \mathcal{U}, \ell_M)$ query, $\mathcal{B}$ prepares the AWIPE message as

$$m^0 = (\hat{x}^b_{\mathsf{pk}}, [\boldsymbol{b}_{\mathsf{pk},\mathcal{U},\ell_M}, 0]_1),$$
$$m^1 = (\hat{x}^b_{\mathsf{pk}}, [\mathbf{0}^{|\mathcal{U}|-1}, 1]_1);$$

and sends (pk, m^0, m^1) to the AWIPE encryption oracle. It uses the returned ciphertext to complete the reply to $\mathcal{A}$.

– $\mathcal{B}$ outputs $\mathcal{A}$'s guess for the challenge bit $\mathsf{AWIPE}.b$.

Let F_{AWIPE} be the functionality defined in Definition 8 for AWIPE. The admissibility condition $(*)$ of AWIPE in each transition holds since one always has

$$F_{\mathsf{AWIPE}}(k^0, m^0) = [\hat{f}_{\mathsf{pk}}(\hat{x}^b_{\mathsf{pk}}) + {\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}]_T = F_{\mathsf{AWIPE}}(k^1, m^1).$$

Game	Adjustment	Assumption
$\mathbf{G}_{2.(\mathcal{U},\ell_M).0}$	aiEnc: $(\mathcal{U}',\ell'_M) < (\mathcal{U},\ell_M)$: $(\hat{x}^0_{\mathsf{pk}}, \boldsymbol{b}^{\ell'_M}_{\mathsf{pk},\mathcal{U}'}, 0)$ $(\mathcal{U}',\ell'_M) \geq (\mathcal{U},\ell_M)$: $(\hat{x}^b_{\mathsf{pk}}, \boldsymbol{b}^{\ell'_M}_{\mathsf{pk},\mathcal{U}'}, 0)$ aiDKGen: same as in $\mathbf{G}_2$	Hybrids on $(\mathcal{U}',\ell'_M) < (\mathcal{U},\ell_M)$
$\mathbf{G}_{2.(\mathcal{U},\ell_M).1}$	aiEnc: $(\mathcal{U}',\ell'_M) = (\mathcal{U},\ell_M) : (\hat{x}^b_{\mathsf{pk}}, \mathbf{0}^{\lvert U \rvert -1}, 1)$ aiKeyGen: $\mathcal{U}' = \mathcal{U}$: $(\hat{f}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}, {\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}}^{\top} \cdot \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}})$	IND of AWIPE
$\mathbf{G}_{2.(\mathcal{U},\ell_M).2}$	aiEnc: same as in $\mathbf{G}_{2.(\mathcal{U},\ell_M).1}$ aiDKGen: $\mathcal{U}' = \mathcal{U}$: $(\hat{f}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}, R^{\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}})$ where $\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} R^{\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}} = -\sum_{\mathsf{pk}\in\mathcal{C}\cap\mathcal{U}} {\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}}^{\top} \cdot \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}$	IND of UZS
$\mathbf{G}_{2.(\mathcal{U},\ell_M).3}$	aiEnc: same as in $\mathbf{G}_{2.(\mathcal{U},\ell_M).1}$ aiDKGen: $\mathcal{U}' = \mathcal{U}$: $(\hat{f}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}, R^{\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}} + \Delta^{b,\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}})$ where $\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} R^{\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}} = -\sum_{\mathsf{pk}\in\mathcal{C}\cap\mathcal{U}} {\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}}^{\top} \cdot \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}$ $\Delta^{b,\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}} = \hat{f}_{\mathsf{pk}}(\hat{x}^0_{\mathsf{pk}}) - \hat{f}_{\mathsf{pk}}(\hat{x}^b_{\mathsf{pk}})$	Statistical
$\mathbf{G}_{2.(\mathcal{U},\ell_M).4}$	aiEnc: same as in $\mathbf{G}_{2.(\mathcal{U},\ell_M).1}$ aiDKGen: $\mathcal{U}' = \mathcal{U}$: $(\hat{f}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}, {\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}}^{\top} \cdot \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}} + \Delta^{b,\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}})$ where $\Delta^{b,\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}} = \hat{f}_{\mathsf{pk}}(\hat{x}^0_{\mathsf{pk}}) - \hat{f}_{\mathsf{pk}}(\hat{x}^b_{\mathsf{pk}})$	IND of UZS
$\mathbf{G}_{2.(\mathcal{U},\ell_M)+1.0} :=$ $\mathbf{G}_{2.(\mathcal{U},\ell_M).5}$	aiEnc: $(\mathcal{U}',\ell'_M) = (\mathcal{U},\ell_M) : (\hat{x}^0_{\mathsf{pk}}, \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}, 0)$ aiKeyGen: $\mathcal{U}' = \mathcal{U}$: $(\hat{f}_{\mathsf{pk}}, \boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}, 0)$	IND of AWIPE

Fig. 14. Intermediate hybrids for the transition from $\mathbf{G}_2$ to $\mathbf{G}_3$ in Theorem 3. We denote by $(\mathcal{U}',\ell'_M) < (\mathcal{U},\ell_M)$ when $(\mathcal{U}',\ell'_M)$ is a previous pair of $(\mathcal{U},\ell_M)$ in the encryption-query set $\mathcal{Q}_M$.

Therefore, in each transition of the multiple hybrid games for each $\mathsf{pk} \in \mathcal{H}$, when $\mathsf{AWIPE}.b = 0$, $\mathcal{A}$ is playing the previous game; when $\mathsf{AWIPE}.b = 1$, $\mathcal{A}$ is playing the subsequent game. Since the number of $\mathsf{pk} \in \mathcal{H}$ queried to either QEnc or QDKGen is bounded by $(q_{\mathsf{QNewHon}} - q_{\mathsf{QCor}})$, one has

$$\left|\mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).0}} - \mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).1}}\right| \leq (q_{\mathsf{QNewHon}} - q_{\mathsf{QCor}}) \cdot \mathsf{Adv}^{\text{sel-sym}}_{\mathsf{AWIPE}}.$$

□

Lemma 10 **(AWS-DDFE: Transition from $\mathbf{G}_{2.(\mathcal{U},\ell_M).1}$ to $\mathbf{G}_{2.(\mathcal{U},\ell_M).2}$).** *For any PPT adversary $\mathcal{A}$, the advantage in distinguishing two games is*

$$\left|\mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).1}} - \mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).2}}\right| \leq \mathsf{Adv}^{\text{otu-sta}}_{\mathsf{UZS}}.$$

Proof. We build an adversary $\mathcal{B}$ against the otu-sta-IND security of UZS from an adversary $\mathcal{A}$ that distinguishes between two games in the transition. To simulate a AWS-DDFE challenger, $\mathcal{B}$ uses the UZS oracles to handle all UZS related operations.

– For each complete $\mathsf{QEnc}(\mathsf{pk}, \hat{x}^0_{\mathsf{pk}}, \hat{x}^1_{\mathsf{pk}}, \mathcal{U}', \ell'_M)$ query, $\mathcal{B}$ prepares the AWIPE message as
  - If $(\mathcal{U}', \ell'_M) \neq (\mathcal{U}, \ell_M)$: it obtains $\boldsymbol{b}_{\mathsf{pk},\mathcal{U}',\ell'_M} \leftarrow \mathsf{QTokGen}(\mathsf{pk}, \mathcal{U}', \ell'_M)$ to complete m.
  - If $(\mathcal{U}', \ell'_M) = (\mathcal{U}, \ell_M)$: it does not have to obtain $\boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}$ as the message to AWIPE encryption is $(\hat{x}^b, [\mathbf{0}^{|\mathcal{U}|-1}, 1]_1)$ in this case.

– For each complete $\mathsf{QDKGen}(\mathsf{pk}, (\hat{f}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}, \mathcal{U})$ query, $\mathcal{B}$ prepares the AWIPE key as
  - $\mathcal{B}$ obtains $[\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}]_2 \leftarrow \mathsf{QSeedGen}(\mathsf{pk}, \mathcal{U}, \ell_{\boldsymbol{f}})$;
  - $\mathcal{B}$ obtains $([R^{\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}]_2)_{\mathsf{pk}\in\mathcal{U}\cap\mathcal{H}} \leftarrow \mathsf{QShare}(\mathcal{U}, \ell_{\boldsymbol{f}} || \ell_M)$;

  where $\boldsymbol{f} = (f_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}$ and completes the key with the inner-product input as

$$k = (\hat{f}_{\mathsf{pk}}, [\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}, R^{\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}]_2).$$

The admissibility condition $(*)$ of UZS holds since

– all the corruption queries in AWS-DDFE are sent in one shot;

– $\mathsf{QShare}(\mathcal{U}, \ell_{\boldsymbol{f}} || \ell_M)$ queries are made for the same ℓ_M on every $\mathcal{U}$ and there are no $\mathsf{QTokGen}(\mathsf{pk}, \mathcal{U}, \ell_M)$ queries required.

When $\mathsf{UZS}.b = 0$, one has $[R^{\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}]_2 = [{\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}}]_2$ which corresponds to $\mathbf{G}_{2.(\mathcal{U},\ell_M).1}$; and when $\mathsf{UZS}.b = 1$, one has $([R^{\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}]_2)_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} \overset{\$}{\leftarrow} \mathcal{R}_{\mathcal{H}\cap\mathcal{U},\ell_{\boldsymbol{f}}||\ell_M}$ (as in Definition 10), which corresponds to $\mathbf{G}_{2.(\mathcal{U},\ell_M).2}$. Therefore, one has

$$\left|\mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).1}} - \mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).2}}\right| \leq \mathsf{Adv}^{\text{otu-sta}}_{\mathsf{UZS}}.$$

□

Lemma 11 **(AWS-DDFE: Transition from $\mathbf{G}_{2.(\mathcal{U},\ell_M).2}$ to $\mathbf{G}_{2.(\mathcal{U},\ell_M).3}$).** *The two games $\mathbf{G}_{2.(\mathcal{U},\ell_M).2}$ to $\mathbf{G}_{2.(\mathcal{U},\ell_M).3}$ are identical.*

Proof. By the fact that $\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} \Delta^{b,\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}} = 0$ (Remark 7), for any random shares $\left(R^{\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}\right)_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$ of the relation

$$\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} R^{\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}} = -\sum_{\mathsf{pk}\in\mathcal{C}\cap\mathcal{U}} {\boldsymbol{a}^{\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}}^\top \cdot \boldsymbol{b}^{\ell_M}_{\mathsf{pk},\mathcal{U}},$$

one has $\left(R^{\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}} + \Delta^{b,\ell_M,\ell_{\boldsymbol{f}}}_{\mathsf{pk},\mathcal{U}}\right)_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}}$ are also random shares of the same relation. □

Lemma 12 **(AWS-DDFE: Transition from $\mathbf{G}_{2.(\mathcal{U},\ell_M).4}$ to $\mathbf{G}_{2.(\mathcal{U},\ell_M).5}$).** *For any PPT adversary $\mathcal{A}$, the advantage in distinguishing two games is*

$$\left|\mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).4}} - \mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).5}}\right| \leq (q_{\mathsf{QNewHon}} - q_{\mathsf{QCor}}) \cdot \mathsf{Adv}^{\text{sel-sym}}_{\mathsf{AWIPE}}.$$

Proof. Given $\{(\mathsf{pk}, \hat{x}_{\mathsf{pk}}^{\tau,0}, \hat{x}_{\mathsf{pk}}^{\tau,1}, \mathcal{U}, \ell_M)\}_{\tau\in[\mathsf{rep}_{\ell_M}]}$ as a set of complete encryption queries and $(\mathsf{pk}, (\hat{f}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}, \mathcal{U})$ as a complete decryption-key query, one has the following facts by the Remark 7:

1. $\Delta_{\mathsf{pk},\mathcal{U}}^{b,\ell_M,\ell_{\boldsymbol{f}}} = \hat{f}_{\mathsf{pk}}(\hat{x}_{\mathsf{pk}}^{0,\tau}) - \hat{f}_{\mathsf{pk}}(\hat{x}_{\mathsf{pk}}^{b,\tau})\ \forall\tau \in [\mathsf{rep}_{\ell_M}]$,
2. $\sum_{\mathsf{pk}\in\mathcal{H}\cap\mathcal{U}} \Delta_{\mathsf{pk},\mathcal{U}}^{b,\ell_M,\ell_{\boldsymbol{f}}} = 0$.

We proceed by using multiple hybrid games for each $\mathsf{pk} \in \mathcal{H}$. We build an adversary $\mathcal{B}$ against the sel-sym security of AWIPE from an adversary $\mathcal{A}$ that distinguishes between two games in the transition. To simulate a AWS-DDFE challenger, $\mathcal{B}$ uses the AWIPE oracles to handle all AWIPE related operations for the reply of each $(\mathsf{pk}, \mathcal{U})$-involved query from $\mathcal{A}$.

– For each complete $\mathsf{QDKGen}(\mathsf{pk}, (\hat{f}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}, \mathcal{U})$ query, $\mathcal{B}$ prepares the AWIPE key as

$$k^0 = (\hat{f}_{\mathsf{pk}}, [\boldsymbol{a}_{\mathsf{pk},\mathcal{U}}^{\ell_{\boldsymbol{f}}}, {\boldsymbol{a}_{\mathsf{pk},\mathcal{U}}^{\ell_{\boldsymbol{f}}}}^{\top} \cdot \boldsymbol{b}_{\mathsf{pk},\mathcal{U}}^{\ell_M} + \Delta_{\mathsf{pk},\mathcal{U}}^{b,\ell_M,\ell_{\boldsymbol{f}}}]_2)$$
$$k^1 = (\hat{f}_{\mathsf{pk}}, [\boldsymbol{a}_{\mathsf{pk},\mathcal{U}}^{\ell_{\boldsymbol{f}}}, {\boldsymbol{a}_{\mathsf{pk},\mathcal{U}}^{\ell_{\boldsymbol{f}}}}^{\top} \cdot \boldsymbol{b}_{\mathsf{pk},\mathcal{U}}^{\ell_M}]_2)$$

where $\boldsymbol{f} = (f_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}$ and sends (pk, k^0, k^1) to the AWIPE decryption key generation oracle. It uses the returned decryption key to complete the reply to $\mathcal{A}$.

– For each complete $\mathsf{QEnc}(\mathsf{pk}, \hat{x}_{\mathsf{pk}}^0, \hat{x}_{\mathsf{pk}}^1, \mathcal{U}, \ell_M)$ query, $\mathcal{B}$ prepares the AWIPE message as

$$m^0 = (\hat{x}_{\mathsf{pk}}^b, [\mathbf{0}^{|\mathcal{U}|-1}, 1]_1);$$
$$m^1 = (\hat{x}_{\mathsf{pk}}^0, [\boldsymbol{b}_{\mathsf{pk},\mathcal{U}}^{\ell_M}, 0]_1);$$

and sends $(\mathsf{pk}, \hat{m}^0, \hat{m}^1)$ to the AWIPE encryption oracle. It uses the returned ciphertext to complete the reply to $\mathcal{A}$.

– $\mathcal{B}$ outputs $\mathcal{A}$'s guess for the challenge bit $\mathsf{AWIPE}.b$.

Let F_{AWIPE} be the functionality defined in Definition 8 for AWIPE. The admissibility condition $(*)$ of AWIPE in each transition holds since one always has

$$\begin{aligned} F_{\mathsf{AWIPE}}(k^0, m^0) &= [(\hat{f}_{\mathsf{pk}}(\hat{x}_{\mathsf{pk}}^b) + \Delta_{\mathsf{pk},\mathcal{U}}^{b,\ell_M,\ell_{\boldsymbol{f}}}) + {\boldsymbol{a}_{\mathsf{pk},\mathcal{U}}^{\ell_{\boldsymbol{f}}}}^{\top} \cdot \boldsymbol{b}_{\mathsf{pk},\mathcal{U}}^{\ell_M}]_T \\ &= [\hat{f}_{\mathsf{pk}}(\hat{x}_{\mathsf{pk}}^0) + {\boldsymbol{a}_{\mathsf{pk},\mathcal{U}}^{\ell_{\boldsymbol{f}}}}^{\top} \cdot \boldsymbol{b}_{\mathsf{pk},\mathcal{U}}^{\ell_M}]_T \text{ (by the above fact 1)} \\ &= F_{\mathsf{AWIPE}}(k^1, m^1) \end{aligned}$$

The index τ is omitted in the above equalities as $\Delta_{\mathsf{pk},\mathcal{U}}^{b,\ell_M,\ell_{\boldsymbol{f}}}$ applies to all pairs of $\tau \in [\mathsf{rep}_{\ell_M}]$ and $\boldsymbol{f} = (\hat{f}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}}$. Therefore, in each transition of the multiple hybrid games for each $\mathsf{pk} \in \mathcal{H}$, when $\mathsf{AWIPE}.b = 0$, $\mathcal{A}$ is playing the previous game; when $\mathsf{AWIPE}.b = 1$, $\mathcal{A}$ is playing the subsequent game. Since the number of pairs $\mathsf{pk} \in \mathcal{H}$ queried to either QEnc or QDKGen is bounded by $(q_{\mathsf{QNewHon}} - q_{\mathsf{QCor}})$, one has

$$\left|\mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).4}} - \mathsf{Adv}_{\mathbf{G}_{2.(\mathcal{U},\ell_M).5}}\right| \le (q_{\mathsf{QNewHon}} - q_{\mathsf{QCor}}) \cdot \mathsf{Adv}_{\mathsf{AWIPE}}^{\text{sel-sym}}.$$

□