Ming Wan, Geng Wang, Shi-Feng Sun *et al.* Efficient Function-Hiding Inner Product Functional Encryption and Its Application to Fine-grained Data Sharing. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY DOI 10.1007/s11390-024-3670-y

Efficient Function-Hiding Inner Product Functional Encryption and Its Application to Fine-grained Data Sharing

Ming Wan (万 明), Geng Wang* (王 更), Shi-Feng Sun (孙士锋), Da-Wu Gu* (谷大武), *Distinguished Member, CCF, Member, ACM*, and Gong-Yu Shi (石攻玉)

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

E-mail: wanming@sjtu.edu.cn; wanggxx@sjtu.edu.cn; shifeng.sun@sjtu.edu.cn; dwgu@sjtu.edu.cn
gy_shi@sjtu.edu.cn

Received August 11, 2023; accepted June 13, 2024.

Abstract In a function-hiding inner product functional encryption (FH-IPFE) scheme, both secret keys and ciphertexts are associated with vectors. Given a secret key for n-dimensional vector $\boldsymbol{x}$, and a ciphertext for n-dimensional vector $\boldsymbol{y}$, a decryptor learns the inner product value $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ and nothing else about both $\boldsymbol{x}$ and $\boldsymbol{y}$. FH-IPFE has been shown to be very useful in privacy-preserving computation. In this paper, we first propose a new (secret-key) FH-IPFE scheme and prove it secure in generic group model. Compared with the state-of-the-art scheme (Kim *et al*, SCN18), the proposed scheme has comparable performance in decryption and reduces 1) the size of master key from n^2 to $3n-1$, 2) the setup complexity from $O\left(n^3\right)$ to $O\left(n\right)$, and 3) the encryption and key generation complexities from $O(n^2)$ to $O(n \log n)$. To the best of our knowledge, this is the most efficient construction based on pairings to date. Moreover, we apply our FH-IPFE scheme to build a fine-grained data sharing system, where data owners store their encrypted data on an untrusted server. Our design supports not only basic database operations but also statistical analyses on encrypted data. To achieve this goal, we also introduce a new security notion, partial-key exposure-resilient simulation-based security (PK-ER-SIM), for FH-IPFE, which enables lightweight clients to securely delegate heavy computations to powerful server and may be independent of interest.

Keywords function-hiding, functional encryption, fine-grained data sharing, generic group model, inner product

1 Introduction

Functional encryption (FE) [1–4] is a powerful generalization of traditional encryption, which enables fine-grained information disclosure to a secret key holder. In an FE scheme, the holder of the master key is able to delegate arbitrary secret keys that allow users to learn specific functions of the data, and nothing else. Specifically, given an encryption of a message x and a secret key for a function f in some function class $\mathcal{F}$, a decryptor only learns the value $f\left(x\right)$.

Inner product functional encryption (IPFE) [5–7], is a special case of FE supporting inner product functionality. IPFE has been shown to be expressive and efficient enough to support many interesting applications such as biometric authentication and queries on encrypted data [8, 9]. In an IPFE scheme, both secret keys and ciphertexts are associated with vectors $\boldsymbol{x} \in \mathbb{Z}^n$ and $\boldsymbol{y} \in \mathbb{Z}^n$. Given a secret key $\mathsf{sk}_{\boldsymbol{x}}$ for $\boldsymbol{x}$ and a ciphertext $\mathsf{ct}_{\boldsymbol{y}}$ for $\boldsymbol{y}$, the decryption algorithm outputs the inner product value $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \mathbb{Z}$.

The basic security notion of IPFE guarantees that an encryption of $\boldsymbol{y}$ reveals only the inner product value $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ (but potentially $\boldsymbol{x}$) to an adversary who has a secret key $\mathsf{sk}_{\boldsymbol{x}}$. However, for certain applications, the ba-

Regular Paper

This work was supported by the National Natural Science Foundation of China under Grant Nos. 62202294 and 62272294, and the National Key Research and Development Program of China under Grant No. 2020YFA0712300.

*Corresponding Author

sic security notion may not be enough and it is further required that secret keys reveal nothing about the underlying functions. For example, let us suppose that a data owner stores sensitive data on an untrusted server in the encrypted form (say $\mathsf{ct}_{\boldsymbol{y}}$) and the server wants to make some specific statistical analyses (with $\mathsf{sk}_{\boldsymbol{x}}$) on the encrypted data. For certain statistical calculations such as variance or covariance, the vector $\boldsymbol{x}$ may be related to the data vector $\boldsymbol{y}$. It is easy to imagine that the owner may also want to hide the particular choice of $\boldsymbol{x}$ from the server. IPFE with such a function-hiding property is defined as function-hiding inner product functional encryption (FH-IPFE)[7, 10–12].

In the last decade, owing to the fast development of FE and homomorphic encryption (HE) [13, 14] , the concept of computation on encrypted data has drawn lots of attentions. In an FE scheme, an owner of the master key can delegate secret keys to an untrusted server that allows to provide fine-grained control over encrypted data, while in an HE scheme, only the master key holder can decrypt the evaluated ciphertext. Therefore, in contrast to the general-purpose leveraging of HE, FE allows the server to gain access to the evaluation results. To capture this scenario, we apply our FH-IPFE scheme to build a fine-grained data sharing system. Let us imagine that each data owner is given a master key for an FH-IPFE scheme and the server stores an encryption $\mathsf{ct}_{\boldsymbol{y}}$ of each owner's sensitive data under the master key. The server will gain access to statistical analyses on the encrypted data by decryption, using delegated secret keys $\mathsf{sk}_{\boldsymbol{x}}$.

On the other hand, some data owners may also want to learn particular statistical results on their own sensitive data and keep those results secret from the server. Here we also consider the problem that the resource of every data owner is limited and must rely on the server for computation. We point out that the standard security notion of FH-IPFE cannot capture the requirement. For such a requirement, we need an additional security notion that if the server is given a partial secret key, it can only help the data owner to calculate an encrypted statistical result (by doing partial decryption). Meanwhile, the server gains no additional information about the actual value. We call such security notion as partial-key exposure-resilient simulation-based security (PK-ER-SIM).

1.1 Our Contributions

In this work, we focus on the construction of more efficient FH-IPFE and its application to fine-grained data sharing system. Our main contributions include:

First, we give a new construction of FH-IPFE scheme where the master key contains just $(3n-1)$ field elements and prove the simulation-based security in the generic group model. This corresponds to a noticeable reduction in the size of the master key compared with all existing schemes. Meanwhile, in our scheme, secret keys and ciphertexts contain $(n+1)$ group elements. This also corresponds to a reduction (by a factor of 2, 4, and 2, respectively) compared with the existing schemes of Bishop *et al.* [10], Datta *et al.* [11], and Tomida *et al.* [12], and has comparable performance with Kim *et al.* [7] (See **Table** 1). As a result, the run-time complexities of all algorithms (except for decryption) are also reduced asymptotically (See **Table** 2). Furthermore, our experiment results also show that over the same pairing curves, the running time of algorithms in our FH-IPFE scheme is much shorter than those in the state-of-the-art construction given by Kim *et al.* [7].

Second, we formalize an additional security notion, PK-ER-SIM, for FH-IPFE. The start-point is that, in our construction, the secret key contains two parts (i.e., K_1 and K_2) and the ciphertext also contains two parts (i.e., C_1 and C_2). We want to demonstrate that with K_2 but not K_1, a decryptor can only get an encryption form of inner product value $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ but not the value itself. The property is necessary for designing fine-gained data sharing application to support statistical analyses on the data owners side.

At last, based on the proposed FH-IPFE scheme, we present a fine-grained data sharing system. Our designed system not only supports basic database operations but also achieves statistical analyses for both

Table 1. Comparison of Parameter Sizes in FH-IPFE Schemes

Scheme	Master Key	Ciphertext	Secret Key
Bishop *et al.* [10]	$(8n^2+8)\ell_{\mathbb{Z}_q}$	$(2n+2)\ell_{\mathbb{G}_1}$	$(2n+2)\ell_{\mathbb{G}_2}$
Datta *et al.* [11]	$(8n^2+12n+28)\ell_{\mathbb{Z}_q}$	$(4n+8)\ell_{\mathbb{G}_1}$	$(4n+8)\ell_{\mathbb{G}_2}$
Tomida *et al.* [12]	$(4n^2+18n+20)\ell_{\mathbb{Z}_q}$	$(2n+5)\ell_{\mathbb{G}_1}$	$(2n+5)\ell_{\mathbb{G}_2}$
Kim *et al.* [7]	$(n^2)\ell_{\mathbb{Z}_q}$	$(n+1)\ell_{\mathbb{G}_1}$	$(n+1)\ell_{\mathbb{G}_2}$
This work	$(3n-1)\ell_{\mathbb{Z}_q}$	$(n+1)\ell_{\mathbb{G}_1}$	$(n+1)\ell_{\mathbb{G}_2}$

Note: All schemes employ asymmetric bilinear maps over two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order q.
n: the dimension of vector. ℓ_G: the bit length to represent an element in group G. $\ell_{\mathbb{Z}_q}$: the bit length to represent an element in field $\mathbb{Z}_q$.

Table 2. Comparison of Run-time Complexities in FH-IPFE Schemes

Scheme	Setup	Key Generation	Encryption	Decryption
Bishop *et al.* [10]	$O\left(n^3\right)$	$O\left(n^2\right)$	$O\left(n^2\right)$	$O(n)$
Datta *et al.* [11]	$O\left(n^3\right)$	$O\left(n^2\right)$	$O\left(n^2\right)$	$O(n)$
Tomida *et al.* [12]	$O\left(n^3\right)$	$O\left(n^2\right)$	$O\left(n^2\right)$	$O(n)$
Kim *et al.* [7]	$O\left(n^3\right)$	$O\left(n^2\right)$	$O\left(n^2\right)$	$O(n)$
This work	$O\left(n\right)$	$O\left(n\log n\right)$	$O\left(n\log n\right)$	$O(n)$

data owners and server. An overview of the system is illustrated in Fig.1. To put it in more detail, we achieve two main functionalities depending on who initiates the request.

1) If the server tries to make a specific statistical analysis on an owner's data, the owner first generates the corresponding secret key $\mathsf{sk}_{\boldsymbol{x}}$ and sends it to the server. Then the server computes the result with the secret key $\mathsf{sk}_{\boldsymbol{x}}$. In this case, we require that the sensitive data $\boldsymbol{x}$ and $\boldsymbol{y}$ are both hidden from the server and it can be guaranteed by the standard security of FH-IPFE since both secret key and ciphertext hide the underlying information.

2) If a data owner wants to learn a particular statistical result on his/her own sensitive data and keeps it secret from the server, the owner first generates the corresponding secret key $\mathsf{sk}_{\boldsymbol{x}}$ and sends partial secret key to the server. The server uses this partial key to get an intermediate value of final result and returns it to the owner. In our design, this procedure will deal with the most part of heavy computation, then the owner only need to do minor operations with the rest part of $\mathsf{sk}_{\boldsymbol{x}}$ to learn the statistical result. In this case, we require that the actual result is hidden from the server and it can be guaranteed by our PK-ER-SIM security notion.

2 Related Works

FH-IPFE was first considered by Bishop *et al.* [10], they gave a direct construction of secret-key function-hiding inner product functional encryption (FH-IPFE) scheme under an indistinguishability-based definition from the symmetric external Diffie-Hellman (SXDH) assumption in bilinear groups. However, they defined a relaxed security model that imposes a somewhat unrealistic admissibility constraint on the adversary's queries. This was further improved upon by Datta *et al.* [11] who showed how to construct a secret-key FH-IPFE scheme from the SXDH assumption that removes the need for that additional constraint on the adversary's queries. Subsequently, Tomida *et al.* [12] gave a construction of secret-key FH-IPFE scheme from the decision linear (DLIN) assumption where the secret keys and ciphertexts consist of $(2n+5)$ group elements. Kim *et al.* [7] also gave a construction of secret-key FH-IPFE scheme in the generic group model where the secret keys and ciphertexts contain only $(n+1)$ group elements. Notably, in all previous constructions, the master key contains $O\left(n^2\right)$ field elements.

The main idea in all those constructions is to use one or several uniformly random matrices and their inverse to hide the information of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. This leads an $O(n^3)$ complexity in setup phase since we need to calculate inverse matrices with Gaussian elimination method. Furthermore, in the key generation and encryption phases, $\boldsymbol{x}$ and $\boldsymbol{y}$ are multiplied by uni-

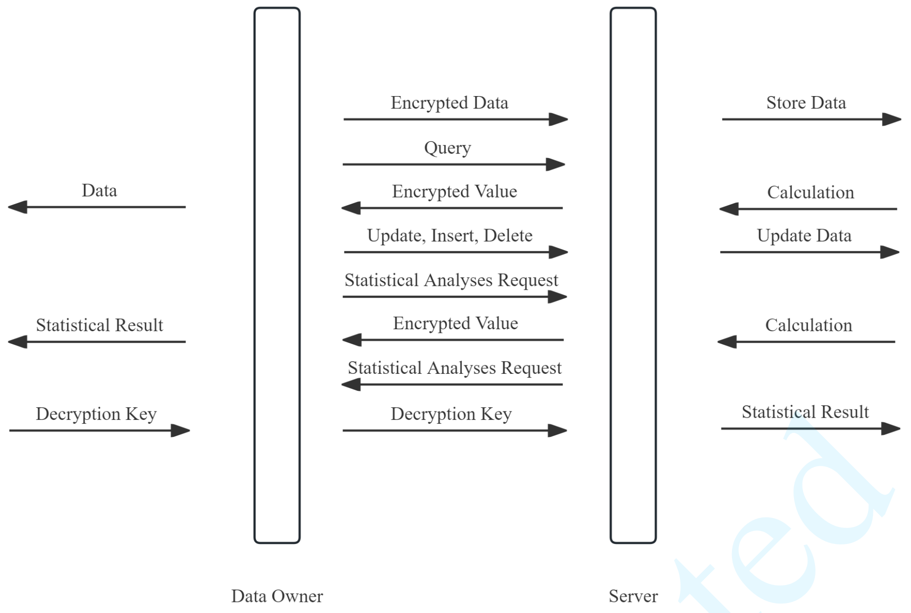

Fig.1. Overview of proposed fine-gained data sharing system.

formly random matrices and it leads an $O(n^2)$ complexity. Thus, the size of master key affects performance heavily since the complexities of setup, encryption and key generation algorithms depend on it. In fact, when considering practical applications enabled by FH-IPFE, the $O(n^3)$ complexity in setup phase is unrealistic due to the inefficient finite field arithmetic computation in $\mathbb{Z}_q$, where $q = 2^\lambda$ and λ is a security parameter. This motivates us to design more efficient FH-IPFE schemes with smaller master key size, which will also reduce the run-time complexities.

3 Preliminaries

3.1 Notations

In **Table** 3, we summarize some notations used in this paper.

3.2 Number Theoretic Transform

The number theoretic transform (NTT) [16] is a specialized version of the Fast Fourier Transform (FFT) over a finite field.

Let q be a prime number and $n = 2^\kappa$ be a power of two such that $q \mod 2n \equiv 1$. Let ω be an n-th primitive root of unity in $\mathbb{Z}_q$. Then for a vector $\boldsymbol{f}$, for $1 \le i \le n$, we define the forward transformation $\bar{f}_i = \sum_{j=1}^n f_j \omega^{(i-1)(j-1)} \mod q$ and the inverse transformation $f_i = n^{-1} \sum_{j=1}^n \bar{f}_j \omega^{-(i-1)(j-1)} \mod q$. For simplicity, we write $\bar{\boldsymbol{f}} = \mathsf{NTT}_\omega(\boldsymbol{f}) = \boldsymbol{T} \cdot \boldsymbol{f}$ and $\boldsymbol{f} = \mathsf{NTT}_\omega^{-1}(\bar{\boldsymbol{f}}) = \boldsymbol{T}^{-1} \cdot \bar{\boldsymbol{f}}$, where $\boldsymbol{T}$ is defined as follows:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)^2} \end{pmatrix} \in \mathbb{Z}_q^{n\times n}.$$

3.3 Bilinear Groups

We briefly recall the basic definition of an asymmetric bilinear group [17, 18]. Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be three distinct groups of prime order q, and let $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ be a function mapping two elements from $\mathbb{G}_1$ and $\mathbb{G}_2$ onto the target group $\mathbb{G}_T$. Let g_1, g_2, and g_T be generators of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, respectively. We write the group operation in groups multiplicatively and write 1 to denote their multiplicative identity. We say that the tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ is an asymmetric

Table 3. Notations in This Paper

Notations	Representations
α or a	scalar
$\boldsymbol{v}$	column vector
$\boldsymbol{M}$	matrix
λ	the security parameter
$\epsilon(\lambda)$	$\epsilon(\lambda)$ is negligible in λ if $\epsilon(\lambda) = o\left(\frac{1}{\lambda^c}\right)$ for all positive integers c
$[n]$	the set $\{1, 2, ..., n\}$
$x \leftarrow S$	the operation of sampling a uniformly random element x from a set S
$x \leftarrow D$	the operation of sampling a random x according to the distribution D
$\boldsymbol{v}^{-1}$	$\left(v_1^{-1}, ..., v_n^{-1}\right)^T$ for $\boldsymbol{v} \in \mathbb{Z}_q^n$ that is non-zero in every coordinate
$\boldsymbol{M}^T$	the transpose of $\boldsymbol{M}$
$\det(\boldsymbol{M})$	the determinant of $\boldsymbol{M}$
$\mathbb{GL}_n(\mathbb{Z}_q)$	the general linear group of $(n \times n)$ matrices over $\mathbb{Z}_q$ (i.e., invertible matrices over $\mathbb{Z}_q$)
$\boldsymbol{u} \odot \boldsymbol{v}$	$(u_1 v_1, ..., u_n v_n)^T$
$g^{\boldsymbol{v}}$	$(g^{v_1}, ..., g^{v_n})^T$

bilinear group if the following properities hold:

- **Efficiency**: The group operations in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ as well the map $e(\cdot, \cdot)$ are all efficiently computable.
- **Non-degeneracy**: $e(g_1, g_2) = g_T \neq 1$.
- **Bilinearity**: $e\left(g_1^a, g_2^b\right) = g_T^{ab}$ for all $a, b \in \mathbb{Z}_q$.

In this work, we additionally let $[a]_1, [b]_2$, and $[c]_T$ denote encodings of a, b, c in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, i.e., g_1^a, g_2^b, and g_T^c, respectively. For a vector $\boldsymbol{v}$ or matrix $\boldsymbol{M}$, we use the shorthand $[\boldsymbol{v}]$ or $[\boldsymbol{M}]$ (for any of the three groups) to denote the group elements obtained by encoding each entry of $\boldsymbol{v}$ and $\boldsymbol{M}$ respectively. Furthermore, for any scalar $k \in \mathbb{Z}_q$ and vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{Z}_q^n$, we write $[\boldsymbol{v}]^k = [k\boldsymbol{v}]$ and $[\boldsymbol{v}][\boldsymbol{w}] = [\boldsymbol{v} + \boldsymbol{w}]$. The pairing operation over groups is also extended to vectors and matrices as follows, $e\left(\left[\boldsymbol{v}^T\right]_1, [\boldsymbol{w}]_2\right) = [\langle \boldsymbol{v}, \boldsymbol{w} \rangle]_T$ and $e\left(\left[\boldsymbol{v}^T\right]_1, [\boldsymbol{M}]_2\right) = \left[\boldsymbol{v}^T \boldsymbol{M}\right]_T$. It is not hard to see that the above operations are all efficiently computable.

3.4 Generic Bilinear Group Model

We prove the security of our construction in a generic model of bilinear groups [19, 20], which is an extension of generic group model [21, 22] adapted to bilinear groups. In the generic group model, access to the group elements is replaced by "handles." In this case, an adversary in the generic group model is also only given access to a stateful oracle which carries out the group operations, and in the bilinear group setting, the pairing operation. The generic group oracle maintains internally a list mapping from handles to group elements, which is used to answer the oracle queries. Thus, when a cryptographic scheme is shown to satisfy some security property in the generic group model, it means that no efficient adversary can break that security property applying the group operations as a black-box oracle. The following definition is taken verbatim from [15] and originally appeared in [7].

Definition 1 (Generic Bilinear Group Oracle)**.** *A generic bilinear group oracle is a stateful oracle* BG *that responds to queries as follows:*

- *On a query* $\mathsf{BG.Setup}(1^\lambda)$*, the oracle generates two fresh handles* $\mathsf{pp}, \mathsf{sp} \leftarrow \{0,1\}^\lambda$ *and a prime* q*. It outputs* $(\mathsf{pp}, \mathsf{sp}, q)$*. It stores the generated values, initializes an empty table* $T \leftarrow \{\}$*, and sets the internal state so subsequent invocations of* BG.Setup *fail.*
- *On a query* $\mathsf{BG.Encode}(k, x, i)$*, where* $k \in \{0,1\}^\lambda$*,* $x \in \mathbb{Z}_q$ *and* $i \in \{1, 2, T\}$*, the oracle checks that* $k = \mathsf{sp}$ *(return* $\perp$ *otherwise). The oracle then generates a fresh handle* $h \leftarrow \{0,1\}^\lambda$*, adds the entry* $h \mapsto (x, i)$ *to table* T*, and outputs* h*.*
- *On a query* $\mathsf{BG.Add}(k, h_1, h_2)$*, where* $k, h_1, h_2 \in \{0,1\}^\lambda$*, the oracle checks that* $k = pp$*, that the handles* h_1, h_2 *are present in its internal table* T *and are mapped to the values* (x_1, i_1) *and* (x_2, i_2)*, respectively, with* $i_1 = i_2$ *(return* $\perp$ *otherwise).*

The oracle then generates a fresh handle $h \leftarrow \{0,1\}^\lambda$, computes $x = x_1 + x_2 \in \mathbb{Z}_q$, adds the entry $h \mapsto (x, i_1)$ to T, and outputs h.

- *On a query* $\mathsf{BG.Pair}(k, h_1, h_2)$, *where* k, h_1, h_2 , *the oracle checks that $k = pp$, that the handles h_1, h_2 are present in T and are mapped to values $(x_1, 1)$ and $(x_2, 2)$, respectively (returning $\perp$ otherwise). The oracle then generates a fresh handle $h \leftarrow \{0,1\}^\lambda$, computes $x = x_1 x_2 \in \mathbb{Z}_q$, adds the entry $h \mapsto (x, T)$ to T, and outputs h.*
- *On a query* $\mathsf{BG.ZeroTest}(k, h)$ *where $k, h \in \{0,1\}^\lambda$, the oracle checks that $k = pp$, that the handle h is present in T and it maps to some value (x, i) (returning $\perp$ otherwise). The oracle then outputs "zero" if $x = 0 \in \mathbb{Z}_q$ and "non-zero" otherwise.*

We notice that in the generic group model, the random elements will often be substituted to formal variables when analyzing the security of constructed schemes. In order to distinguish between a specific value and a formal variable, we will explicitly write x if it is a specific value and $\hat{x}$ if it is a formal variable. This notation will also naturally extend to vectors $\hat{\boldsymbol{v}}$ and matrices $\hat{\boldsymbol{M}}$ where their each entry is a formal variable.

We also use the Schwartz-Zippel lemma [23, 24] in the security proof, stated below.

Lemma 2 (Schwartz-Zippel [23, 24], adapted)**.** *Fix a prime q and let $f \in \mathbb{F}_q[\hat{x}_1, ..., \hat{x}_n]$ be an n-variate polynomial with degree at most d and which is not identically zero. Then,*

$$\Pr[x_1, ..., x_n \leftarrow \mathbb{F}_q : f(x_1, ..., x_n) = 0] \leq \frac{d}{q}.$$

4 Function-Hiding Inner Product Functional Encryption

4.1 Syntax

We recall the definition of secret-key FH-IPFE schemes.

Definition 3 ([7])**.** *A secret-key FH-IPFE scheme is a tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ defined over a message space $\mathbb{Z}_q^n \backslash \{0^n\}$ with the following properties:*

- $\mathsf{Setup}(1^\lambda, S) \mapsto (\mathsf{pp}, \mathsf{msk})$*: On input a security parameter λ and a set $S \subseteq \mathbb{Z}_q$, the setup algorithm* Setup *outputs the public parameter* pp *and the master key* msk.
- $\mathsf{KeyGen}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{x}) \mapsto \mathsf{sk}_{\boldsymbol{x}}$*: On input the public parameter* pp, *the master key* msk *and a vector $\boldsymbol{x} \in \mathbb{Z}_q^n \backslash \{0^n\}$, the key generation algorithm* KeyGen *outputs a secret key* $\mathsf{sk}_{\boldsymbol{x}}$.
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{y}) \mapsto \mathsf{ct}_{\boldsymbol{y}}$*: On input the public parameter* pp, *the master key* msk, *and a vector $\boldsymbol{y} \in \mathbb{Z}_q^n \backslash \{0^n\}$, the encryption algorithm* Enc *outputs a ciphertext* $\mathsf{ct}_{\boldsymbol{y}}$.
- $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}_{\boldsymbol{y}}, \mathsf{sk}_{\boldsymbol{x}}) \mapsto z/\perp$*: On input the public parameter* pp, *a secret key* $\mathsf{sk}_{\boldsymbol{x}}$ *and a ciphertext* $\mathsf{ct}_{\boldsymbol{y}}$, *the decryption algorithm* Dec *outputs a result $z \in S$ or a special symbol $\perp$.*

An FH-IPFE scheme $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ defined above is correct if for all sets S where $|S| = poly(\lambda)$, and all non-zero vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_q^n \backslash \{0^n\}$, where $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in S$, the following condition holds. Let $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, S)$, $\mathsf{sk}_{\boldsymbol{x}} \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{x})$, and $\mathsf{ct}_{\boldsymbol{y}} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{y})$, then

$$\Pr[\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_{\boldsymbol{x}}, \mathsf{ct}_{\boldsymbol{y}}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle] = 1 - \epsilon(\lambda),$$

where the probability is taken over the internal randomness of the Setup, Enc and KeyGen algorithms, and $\epsilon(\lambda)$ is a negligible function.

4.2 Simulation-Based Security

In the simulation-based definition, we require that every efficient adversary that interacts with the real encryption and key generation oracles can be simulated given only oracle access to the inner products between each pair of vectors that the adversary submits to the key generation and encryption oracles. The following definition is taken verbatim from [7].

Definition 4. *Let $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an FH-IPFE scheme. Then Π is SIM-secure if for all PPT adversaries $\mathcal{A}$, there exists an efficient simulator $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$ such that the outputs of the following experiments are computationally indistinguishable:*
Real experiment $\mathsf{Real}_{\mathcal{A}}(1^\lambda)$*:*

- $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$
- $b \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{pp},\mathsf{msk},\cdot),\mathcal{O}_{\mathsf{Enc}}(\mathsf{pp},\mathsf{msk},\cdot)}(1^\lambda)$
- *output* b

Ideal experiment $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$*:*

- *initialize* $\mathcal{X} \leftarrow \emptyset$ *and* $\mathcal{Y} \leftarrow \emptyset$
- $(\mathsf{pp}, \mathsf{st}) \leftarrow \mathcal{S}_1(\lambda)$
- $b \leftarrow \mathcal{A}^{\mathcal{O}'_{\mathsf{KeyGen}}(\mathsf{pp},\cdot),\mathcal{O}'_{\mathsf{Enc}}(\mathsf{pp},\cdot)}(1^\lambda)$
- *output* b

where the oracles $\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{pp}, \mathsf{msk}, \cdot)$, $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pp}, \mathsf{msk}, \cdot)$, $\mathcal{O}'_{\mathsf{KeyGen}}(\mathsf{pp}, \cdot)$ *and* $\mathcal{O}'_{\mathsf{Enc}}(\mathsf{pp}, \cdot)$ *are defined as follows:*

1) *Oracles* $\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{pp}, \mathsf{msk}, \cdot)$ *and* $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pp}, \mathsf{msk}, \cdot)$ *represent the real encryption and key generation oracles, respectively. Specifically,* $\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{x}) = \mathsf{KeyGen}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{x})$ *and* $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{y}) = \mathsf{Enc}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{y})$.

2) *Oracles* $\mathcal{O}'_{\mathsf{KeyGen}}(\mathsf{pp}, \cdot)$ *and* $\mathcal{O}'_{\mathsf{Enc}}(\mathsf{pp}, \cdot)$ *represent the ideal encryption and key generation oracles, respectively. The two oracles are stateful and share counters i and j (initialized to 0 at the beginning of the experiment), a simulator sate* st *(initialized to state output by $\mathcal{S}_1$), and a collection of mappings:*

$$\mathcal{C}_{\mathsf{ip}} = \{(i', j') \mapsto \langle \boldsymbol{x}^{(i')}, \boldsymbol{y}^{(j')} \rangle : (i', j') \in [i] \times [j]\},$$

where $\boldsymbol{x}^{(i)} \in \mathbb{Z}_q^n$ and $\boldsymbol{y}^{(j)} \in \mathbb{Z}_q^n$ are the inputs for the i-th invocation of $\mathcal{O}'_{\mathsf{KeyGen}}(\mathsf{pp}, \cdot)$ *and the j-th invocation of* $\mathcal{O}'_{\mathsf{Enc}}(\mathsf{pp}, \cdot)$ *by the adversary, respectively. At the beginning of the experiment, the set $\mathcal{C}_{\mathsf{ip}}$ is initialized to the empty set.*

a) *On the adversary's i-th invocation of* $\mathcal{O}'_{\mathsf{KeyGen}}(\mathsf{pp}, \cdot)$ *with input vector $\boldsymbol{x}^{(i)} \in \mathbb{Z}_q^n$, the oracle* $\mathcal{O}'_{\mathsf{KeyGen}}$ *sets $i \leftarrow i + 1$, updates the collection of mapping $\mathcal{C}_{\mathsf{ip}}$, and invokes the simulator $\mathcal{S}_2$ on inputs $\mathcal{C}_{\mathsf{ip}}$ and* st*. The simulator responds with a tuple* $(\mathsf{sk}_{\boldsymbol{x}^{(i)}}, \mathsf{st}') \leftarrow \mathcal{S}_2(\mathcal{C}_{\mathsf{ip}}, \mathsf{st})$*. The oracle updates the state* $\mathsf{st} \leftarrow \mathsf{st}'$ *and replies to the adversary with the secret key* $\mathsf{sk}_{\boldsymbol{x}^{(i)}}$*.*

b) *On the adversary's j-th invocation of* $\mathcal{O}'_{\mathsf{Enc}}(\mathsf{pp}, \cdot)$ *with input vector $\boldsymbol{y}^{(j)} \in \mathbb{Z}_q^n$, the oracle* $\mathcal{O}'_{\mathsf{Enc}}$ *sets $j \leftarrow j + 1$, updates the collection of mapping $\mathcal{C}_{\mathsf{ip}}$, and invokes the simulator $\mathcal{S}_3$ on inputs $\mathcal{C}_{\mathsf{ip}}$ and* st*. The simulator responds with a tuple* $(\mathsf{ct}_{\boldsymbol{y}^{(j)}}, \mathsf{st}') \leftarrow \mathcal{S}_3(\mathcal{C}_{\mathsf{ip}}, \mathsf{st})$*. The oracle updates the state* $\mathsf{st} \leftarrow \mathsf{st}'$ *and replies to the adversary with the ciphertext* $\mathsf{ct}_{\boldsymbol{y}^{(j)}}$*.*

4.3 Partial-Key Exposure-Resilient Simulation-Based Security

For our purpose, here we present a new security notion called PK-ER-SIM, for FH-IPFE. In our construction, the key generation algorithm KeyGen outputs a secret key $\mathsf{sk}_{\boldsymbol{x}}$ consisting of two components K_1 and K_2. The PK-ER-SIM security shows that without knowing K_1, the adversary learns no information about the inner product value even if it holds K_2.

Definition 5. *Let $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an FH-IPFE scheme. Then Π is PK-ER-SIM-secure if for all PPT adversaries $\mathcal{A}$, there exists an efficient simulator $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$ such that the outputs of the following experiments are computationally indistinguishable:*
Real experiment $\mathsf{Real}_{\mathcal{A}}(1^\lambda)$*:*

- $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$
- $b \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}'}(\mathsf{pp},\mathsf{msk},\cdot),\mathcal{O}_{\mathsf{Enc}}(\mathsf{pp},\mathsf{msk},\cdot)}(1^\lambda)$
- *output* b

Ideal experiment $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$*:*

- *initialize* $\mathcal{X} \leftarrow \emptyset$ *and* $\mathcal{Y} \leftarrow \emptyset$

- $(\mathsf{pp}, \mathsf{st}) \leftarrow \mathcal{S}_1(\lambda)$
- $b \leftarrow \mathcal{A}^{\mathcal{O}'_{\mathsf{KeyGen}'}(\mathsf{pp},\cdot),\mathcal{O}'_{\mathsf{Enc}}(\mathsf{pp},\cdot)}(1^\lambda)$
- *output* b

where the oracles $\mathcal{O}_{\mathsf{KeyGen}'}(\mathsf{pp}, \mathsf{msk}, \cdot)$, $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pp}, \mathsf{msk}, \cdot)$, $\mathcal{O}'_{\mathsf{KeyGen}'}(\mathsf{pp}, \cdot)$ *and* $\mathcal{O}'_{\mathsf{Enc}}(\mathsf{pp}, \cdot)$ *are defined as follows:*

1) *Oracle* $\mathcal{O}_{\mathsf{KeyGen}'}(\mathsf{pp}, \mathsf{msk}, \cdot)$ *behaves the same as* $\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{pp}, \mathsf{msk}, \cdot)$ *except that it only returns* K_2. *Oracle* $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pp}, \mathsf{msk}, \cdot)$ *represents the real encryption oracle. Specifically,* $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{y}) = \mathsf{Enc}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{y})$.

2) *Oracles* $\mathcal{O}'_{\mathsf{KeyGen}'}(\mathsf{pp}, \cdot)$ *and* $\mathcal{O}'_{\mathsf{Enc}}(\mathsf{pp}, \cdot)$ *represent the ideal encryption and key generation oracles, respectively. The two oracles are stateful and share counters* i *and* j *(initialized to* 0 *at the beginning of the experiment), a simulator sate* st *(initialized to state output by* $\mathcal{S}_1$*), and a collection of mappings:*

$$\mathcal{C}_{\mathsf{ip}} = \{(i', j') \mapsto \alpha_{i'} \langle \boldsymbol{x}^{(i')}, \boldsymbol{y}^{(j')} \rangle : (i', j') \in [i] \times [j]\},$$

where α_i *is a random value tied by* $\boldsymbol{x}^{(i)}$. *Moreover,* $\boldsymbol{x}^{(i)} \in \mathbb{Z}_q^n$ *and* $\boldsymbol{y}^{(j)} \in \mathbb{Z}_q^n$ *are the inputs for the* i*-th invocation of* $\mathcal{O}'_{\mathsf{KeyGen}'}(\mathsf{pp}, \cdot)$ *and the* j*-th invocation of* $\mathcal{O}'_{\mathsf{Enc}}(\mathsf{pp}, \cdot)$ *by the adversary, respectively. At the beginning of the experiment, the set* $\mathcal{C}_{\mathsf{ip}}$ *is initialized to the empty set.*

   a) *On the adversary's* i*-th invocation of* $\mathcal{O}'_{\mathsf{KeyGen}'}(\mathsf{pp}, \cdot)$ *with input vector* $\boldsymbol{x}^{(i)} \in \mathbb{Z}_q^n$, *the oracle* $\mathcal{O}'_{\mathsf{KeyGen}'}$ *sets* $i \leftarrow i + 1$, *updates the collection of mapping* $\mathcal{C}_{\mathsf{ip}}$, *and invokes the simulator* $\mathcal{S}_2$ *on inputs* $\mathcal{C}_{\mathsf{ip}}$ *and* st. *The simulator responds with a tuple* $(\mathsf{sk}_{\boldsymbol{x}^{(i)}}, \mathsf{st}') \leftarrow \mathcal{S}_2(\mathcal{C}_{\mathsf{ip}}, \mathsf{st})$. *The oracle updates the state* $\mathsf{st} \leftarrow \mathsf{st}'$ *and replies to the adversary with the second part of secret key* $\mathsf{sk}_{\boldsymbol{x}^{(i)}}$, *i.e.,* $K_2^{(i)}$.

   b) *On the adversary's* j*-th invocation of* $\mathcal{O}'_{\mathsf{Enc}}(\mathsf{pp}, \cdot)$ *with input vector* $\boldsymbol{y}^{(j)} \in \mathbb{Z}_q^n$, *the oracle* $\mathcal{O}'_{\mathsf{Enc}}$ *sets* $j \leftarrow j + 1$, *updates the collection of mapping* $\mathcal{C}_{\mathsf{ip}}$, *and invokes the simulator* $\mathcal{S}_3$ *on inputs* $\mathcal{C}_{\mathsf{ip}}$ *and* st. *The simulator responds with a tuple* $(\mathsf{ct}_{\boldsymbol{y}^{(j)}}, \mathsf{st}') \leftarrow \mathcal{S}_3(\mathcal{C}_{\mathsf{ip}}, \mathsf{st})$. *The oracle updates the state* $\mathsf{st} \leftarrow \mathsf{st}'$ *and replies to the adversary with the ciphertext* $\mathsf{ct}_{\boldsymbol{y}^{(j)}}$.

Remark. In this definition, the inner product $\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$ is hidden from the simulator by α_i if and only if when $\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle \neq 0$. Therefore, in our construction below, we require that $\boldsymbol{x}$ and $\boldsymbol{y}$ contain only positive elements to meet the condition, which means that we may need a preprocessing phase on the data.

5 Our Efficient FH-IPFE Scheme

In this section, we give our construction of secret-key FH-IPFE scheme. We then prove that the scheme is simulation-based secure and PK-ER-SIM secure in the generic bilinear group model.

5.1 Technical Overview

Following the basic idea of designing FH-IPFE scheme mentioned by Kim *et al.* [7] in generic group model, the main idea is to use a random matrix $\boldsymbol{R}$ to hide the information of $\boldsymbol{x}$ and $\boldsymbol{y}$. In the construction of Kim *et al.*, the master key is $\boldsymbol{R}$ and $\boldsymbol{R}^* = \det(\boldsymbol{R}) \cdot (\boldsymbol{R}^{-1})^T$. The ciphertext $\mathsf{ct}_{\boldsymbol{y}} = (C_1, C_2)$ contains two parts $C_1 = g_1^\beta$ and $C_2 = g_1^{\beta \cdot \boldsymbol{R}^* \cdot \boldsymbol{y}}$ and the secret key $\mathsf{sk}_{\boldsymbol{x}} = (K_1, K_2)$ also contains two parts $K_1 = g_2^{\alpha \cdot \det(\boldsymbol{R})}$ and $K_2 = g_2^{\alpha \cdot \boldsymbol{R} \cdot \boldsymbol{x}}$. Note that α and β are random values, generated while running the corresponding algorithms. As shown, in their scheme, the master key is composed of n^2 field elements and the run-time complexities of setup, encryption and key generation algorithms are $O(n^3)$, $O(n^2)$, and $O(n^2)$, respectively. To reduce the size of parameters, we choose a structured matrix that can be generated with only $(3n - 1)$ field elements to replace the original random matrix and the complexity of such structured matrix inversion is $O(n)$. Our matrix $\boldsymbol{R}$ is composed of three vectors $\boldsymbol{r}$, $\boldsymbol{s}$, and $\boldsymbol{t}$ that are nonzero in every coordinate and $\boldsymbol{R}^{-1}$ can be calculated by $\boldsymbol{r}^{-1}$, $\boldsymbol{s}$, and $\boldsymbol{t}^{-1}$ where $\boldsymbol{r}^{-1} = \left(r_1^{-1}, ..., r_n^{-1}\right)^T$ and the same case for $\boldsymbol{t}^{-1}$.

However, the security of the scheme should be reconsidered since the structured matrix cannot guarantee that the underlying vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are well hidden. To solve this problem, we utilize the technical of number theoretic transform (NTT) to convert our structured matrix to be non-sparse to provide sufficient randomness and improve the efficiency. Generally, let ω be an n-th primitive root of unity in $\mathbb{Z}_q$ and we define a public Vandermonde matrix $\boldsymbol{T}$ and its inverse $\boldsymbol{T}^{-1}$ generated by ω to multiply the structured $\boldsymbol{R}$ while in key generation and encryption algorithms. By the properties of Vandermonde matrix $\boldsymbol{T}$, the result new matrix is non-sparse and the procedures of generating K_2 and C_2 can be iterated using NTT and its inverse instead of matrix multiplication. We state an important lemma, which is a generalization of zeroing lemma appeared in [15] and prove that it also holds for our structured matrix (See Appendix). This allows us to hide the underlying vectors and reduce the run-time complexities of key generation and encryption algorithms from $O(n^2)$ to $O(n \log n)$.

5.2 Construction

Fix a security parameter λ, and let $n = n(\lambda)$ be a power-of-two positive integer, $q = q(\lambda)$ be a positive prime integer such that $q \mod 2n \equiv 1$. Let S be a polynomial-sized subset of $\mathbb{Z}_q$. We construct a FH-IPFE scheme $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as follows.

- $\mathsf{Setup}(1^\lambda, S)$: On input the security parameter λ, and the set S, the setup algorithm samples an asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ and chooses generators $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and $\omega \in \mathbb{Z}_q \backslash \{0\}$. It samples $\boldsymbol{r}, \boldsymbol{t} \leftarrow (\mathbb{Z}_q^*)^n$ and $\boldsymbol{s} \leftarrow (\mathbb{Z}_q^*)^{n-1}$. Then it computes $\boldsymbol{r}^{-1}$ and $\boldsymbol{t}^{-1}$. Finally, the setup algorithm outputs the public parameters $\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, S)$ and master $\mathsf{msk} = (\boldsymbol{r}, \boldsymbol{t}, \boldsymbol{s}, \boldsymbol{r}^{-1}, \boldsymbol{t}^{-1})$.

- $\mathsf{KeyGen}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{x})$: On input the public parameters pp, the master key msk and a vector $\boldsymbol{x} \in \mathbb{Z}_q^n \backslash \{0^n\}$, the key generation algorithm computes $\boldsymbol{x}' = \boldsymbol{x} \odot \boldsymbol{t}$ and $\bar{\boldsymbol{x}} = \mathsf{NTT}_\omega(\boldsymbol{x}')$. Then, it computes $\boldsymbol{x}^* = \bar{\boldsymbol{x}} \odot \boldsymbol{r} + (0 \| \bar{x}_1 s_1, ..., \bar{x}_{n-1} s_{n-1})^T$, chooses a uniformly random element $\alpha \in \mathbb{Z}_q$ and outputs $\mathsf{sk}_{\boldsymbol{x}} = (K_1, K_2)$, where
$$K_1 = [\alpha]_1 \quad \text{and} \quad K_2 = [\alpha \cdot \boldsymbol{x}^*]_1 .$$

- $\mathsf{Enc}(\mathsf{pp}, \mathsf{msk}, \boldsymbol{y})$: On input the public parameters pp, the master key msk and a vector $\boldsymbol{y} \in \mathbb{Z}_q^n \backslash \{0^n\}$, the encryption algorithm computes $\boldsymbol{y}' = \boldsymbol{y} \odot \boldsymbol{t}^{-1}$ and $\bar{\boldsymbol{y}} = \mathsf{NTT}_\omega^{-1}(\boldsymbol{y}')$. Then, it computes $\boldsymbol{y}^* = (y_1^*, ..., y_n^*)^T$ as follows:
  1) $y_n^* = \bar{y}_n r_n^{-1}$,
  2) for i from $n-1$ to 1, $y_i^* = r_i^{-1}\left(\bar{y}_i - s_i y_{i+1}^*\right)$.

  Finally, it chooses a uniformly random element $\beta \in \mathbb{Z}_q$ and outputs $\mathsf{ct}_{\boldsymbol{y}} = (C_1, C_2)$, where
$$C_1 = [\beta]_2 \quad \text{and} \quad C_2 = [\beta \cdot \boldsymbol{y}^*]_2 .$$

- $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}_{\boldsymbol{y}}, \mathsf{sk}_{\boldsymbol{x}})$: On input the public parameters pp, a secret key $\mathsf{sk}_{\boldsymbol{x}} = (K_1, K_2)$ and a ciphertext $\mathsf{ct}_{\boldsymbol{y}} = (C_1, C_2)$, the decryption algorithm computes
$$D_1 = e(K_1, C_1) \quad \text{and} \quad D_2 = e(K_2, C_2) .$$
  Then, it calculates the discrete logarithm to check whether there exists $z \in S$ such that $(D_1)^z = D_2$. If so, the decryption algorithm outputs z. Otherwise, it outputs $\perp$. Note that this algorithm is efficient since $|S| = poly(\lambda)$.

5.3 Correctness

Let $\boldsymbol{R} = \begin{pmatrix} r_1 & s_1 & 0 & \dots & 0 \\ 0 & r_2 & s_2 & \dots & 0 \\ 0 & 0 & r_3 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & r_n \end{pmatrix}$ and $\boldsymbol{D} = diag(t_1, ..., t_n)$. It is easy to verify that $\boldsymbol{x}^* = \boldsymbol{R}^T \cdot \bar{\boldsymbol{x}}$ and $\boldsymbol{y}^* = \boldsymbol{R}^{-1} \cdot \bar{\boldsymbol{y}}$. Since $\boldsymbol{D}$ and $\boldsymbol{T}$ are both symmetric invertible matrices, we can rewrite D_2 as follows:
$$D_2 = e(K_2, C_2) = [\alpha\beta\langle \boldsymbol{x}^*, \boldsymbol{y}^* \rangle]_T = \left[\alpha\beta\bar{\boldsymbol{x}}^T \cdot \boldsymbol{R} \cdot \boldsymbol{R}^{-1} \cdot \bar{\boldsymbol{y}}\right]_T$$
$$= \left[\alpha \boldsymbol{x}^T \cdot \boldsymbol{D}^T \cdot \boldsymbol{T}^T \cdot \boldsymbol{R} \cdot \boldsymbol{R}^{-1} \cdot \boldsymbol{T}^{-1} \cdot \boldsymbol{D}^{-1} \cdot \boldsymbol{y}\right]_T$$
$$= [\alpha\beta\langle \boldsymbol{x}, \boldsymbol{y} \rangle]_T .$$

On the other hand, $D_1 = e(K_1, C_1) = [\alpha\beta]_T$. Then if $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in S$, the decryption algorithm will correctly output $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$.

5.4 Simulation-Based Security

To prove the simulation-based security of Π in the generic group model, we construct a simulator that, given only the inner products of the vectors corresponding to the key generation and encryption queries, is able to correctly simulate the real distribution of the secret keys and ciphertext. We first state a lemma that plays important roles in our security proof and the proof details can be founded in the Appendix.

The following lemma is inspired by a variant of lemma due to [25] that originally appeared in [26]. We adapt their results to our designed structural matrix $\hat{\boldsymbol{R}}$.

Lemma 6. *Let* $\hat{\boldsymbol{R}} = \begin{pmatrix} \hat{r}_1 & \hat{s}_1 & 0 & \dots & 0 \\ 0 & \hat{r}_2 & \hat{s}_2 & \dots & 0 \\ 0 & 0 & \hat{r}_3 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & \hat{r}_n \end{pmatrix}$ *be an $n \times n$ matrix of variables $\hat{r}_i$ and $\hat{s}_j$, and $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}_q^n$ be two arbitrary vectors that are non-zero in every coordinate. Let $\hat{\boldsymbol{u}}^T = \boldsymbol{u}^T \cdot \hat{\boldsymbol{R}}^{-1}$ and $\hat{\boldsymbol{v}} = \hat{\boldsymbol{R}} \cdot \boldsymbol{v}$ be two vectors of rational functions over the $\hat{r}_i$ and $\hat{s}_j$ formal variables. Let P be a polynomial over the entries of $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{v}}$ such that each monomial contains exactly one entry from $\hat{\boldsymbol{u}}$ and one from $\hat{\boldsymbol{v}}$. Then if P is identically a constant over the $\hat{r}_i$ and $\hat{s}_j$ variables, it must be a constant multiple of the inner product of $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{v}}$.*

Theorem 7. *The FH-IPFE scheme Π is SIM-secure in the generic group model.*

Proof. We start by defining the simulation-based security in the generic group model.
1) *Real Experiment* $\mathsf{Real}_{\mathcal{A}}\left(1^\lambda\right)$:

- $\mathsf{Setup}\left(1^\lambda\right)$: for generators g_1, g_2, and g_T, the challenger queries the oracle $\mathsf{BG.Encode}$ and returns the handles $\{h_i\}_{i\in\{1,2,T\}}$. For convenience, we denote handle h_i as $[1]_i$.

- $\mathcal{O}_{\mathsf{KeyGen}}\left(\mathsf{pp}, \mathsf{msk}, \cdot\right)$: for any vector $\boldsymbol{x}_i$ corresponding to the i-th key generation queries, the challenger queries the oracle $\mathsf{BG.Encode}$ and returns the handles $[\alpha_i]_1$ and $[\alpha_i \boldsymbol{x}_i^*]_1$.

- $\mathcal{O}_{\mathsf{Enc}}\left(\mathsf{pp}, \mathsf{msk}, \cdot\right)$: for any vector $\boldsymbol{y}_j$ corresponding to the j-th encryption queries, the challenger queries the oracle $\mathsf{BG.Encode}$ and returns the handles $[\beta_j]_2$ and $\left[\beta_j \boldsymbol{y}_j^*\right]_2$.

- The adversary also has the ability to query other GGM oracles such as $\mathsf{BG.Add}$, $\mathsf{BG.Pair}$, and $\mathsf{BG.ZeroTest}$, the challenger just simply forwards those queries and returns the results.

2) *Ideal Experiment* $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}\left(1^\lambda\right)$:

- The simulator $\mathcal{S}_1$ generates a random handle for each group generator as the public parameters, and returns all the handles to the adversary.

- $\mathcal{O}'_{\mathsf{KeyGen}}\left(\mathsf{pp}, \cdot\right)$: On input a vector $\boldsymbol{x}_i$, the simulator $\mathcal{S}_2$ receives as input a new collect $\mathcal{C}'_{\mathsf{IP}}$ of inner products (instead of real $\boldsymbol{x}_i$) and updates $\mathcal{C}_{\mathsf{ip}} \leftarrow \mathcal{C}'_{\mathsf{ip}}$. Then $\mathcal{S}_2$ generates fresh handles $[\hat{\alpha}_i]_1$, $[\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1$ and responds with a secret key $\mathsf{sk}_{\boldsymbol{x}_i} = \left([\hat{\alpha}_i]_1, [\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1\right)$.

- $\mathcal{O}'_{\mathsf{Enc}}\left(\mathsf{pp}, \cdot\right)$: On input a vector $\boldsymbol{y}_j$, the simulator $\mathcal{S}_3$ receives as input a new collect $\mathcal{C}'_{\mathsf{IP}}$ of inner products (instead of real $\boldsymbol{y}_j$) and updates $\mathcal{C}_{\mathsf{ip}} \leftarrow \mathcal{C}'_{\mathsf{ip}}$. Then $\mathcal{S}_3$ generates fresh handles $\left[\hat{\beta}_j\right]_2$, $\left[\hat{\beta}_j \hat{\boldsymbol{y}}_j^*\right]_2$ and responds with a ciphertext $\mathsf{ct}_{\boldsymbol{y}_j} = \left(\left[\hat{\beta}_j\right]_2, \left[\hat{\beta}_j \hat{\boldsymbol{y}}_j^*\right]_2\right)$.

- The adversary also has the ability to query other GGM oracles. For $\mathsf{BG.Add}$, $\mathsf{BG.Pair}$, the simulator $\mathcal{S}$ honestly returns the corresponding handles but for $\mathsf{BG.ZeroTest}$, the simulator $\mathcal{S}$ returns the output *zero* if and only if

$$\sum_{i,j} \hat{\alpha}_i \hat{\beta}_j \left(\kappa_{i,j} + c_{i,j} \mathcal{C}_{\mathsf{ip}}\left(i, j\right)\right) = 0,$$

where $\kappa_{i,j}$ and $c_{i,j}$ represent coefficients submitted by a $\mathcal{A}$ and $\mathcal{C}_{\mathsf{ip}}\left(i, j\right)$ are the collection of mappings defined before.

Let Q_1 and Q_2 be the total number of key generation and encryption oracle queries made by $\mathcal{A}$, respectively. We present two hybrid experiments, beginning with the original real experiment.

- Exp_0: Exp_0 is the real experiment and the adversary has access to the handles of elements $\{[\alpha_i]_1\}_i$, $\{[\alpha_i \boldsymbol{x}_i^*]_1\}_i$, $\{[\beta_j]_2\}_j$, and $\{[\beta_j \boldsymbol{y}_j^*]_2\}_j$.

- Exp_1: Exp_1 is obtained from Exp_0 by modifying the random elements chosen by challenger $\mathcal{C}$ to formal variables. We can imagine replacing $\{\alpha_i\}_i$, $\{\beta_j\}_j$, $\boldsymbol{r}$, $\boldsymbol{t}$, and $\boldsymbol{s}$ with formal variables. In this case, for every zero-test query submitted by $\mathcal{A}$, the resulting zero-test expressions are substituted by rational functions of above variables. By construction, this results in a polynomial of degree at most $(n+1)$ over the formal variables. Hence, for each zero-test, the difference between Exp_0 and Exp_1 is $\frac{n+1}{q}$, due to the Schwartz-Zippel lemma. Assuming that $\mathcal{A}$ makes zero-tests for p (which is a polynomial size) times, by union bound, the difference between Exp_0 and Exp_1 can be bounded by $\frac{p(n+1)}{q}$, which is negligible. In other words, $\mathcal{A}$ cannot distinguish this switch except with negligible probability.

Now in Exp_1, we rewrite the handles of the elements given to $\mathcal{A}$: $\{[\hat{\alpha}_i]_1\}_i$, $\{[\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1\}_i$, $\{\left[\hat{\beta}_j\right]_2\}_j$, and $\{\left[\hat{\beta}_j \hat{\boldsymbol{y}}_j^*\right]_2\}_j$. We consider all the possible ways that $\mathcal{A}$ can produce elements in the target group (i.e., $\mathbb{G}_T$) and write a general linear combinations of such elements (as a polynomial expression), where ρ, γ_i, $\kappa_{i,j}$, $\theta_{i,j,l}$, $\psi_{i,k}$, $\tau_{i,j,k}$, $\eta_{i,j,k,l}$, δ_j, and $\mu_{j,l}$ represent coefficients submitted by $\mathcal{A}$. Firstly, we list all the elements that may appear in the final expression:

- Adding by a constant in $\mathbb{G}_T$: ρ;
- Pairing $\{[\hat{\alpha}_i]_1\}_i$ by a constant in $\mathbb{G}_2$: $\sum_i \gamma_i \hat{\alpha}_i$;
- Pairing $\{[\hat{\alpha}_i]_1\}_i$ with $\{\left[\hat{\beta}_j\right]_2\}_j$: $\sum_{i,j} \kappa_{i,j} \hat{\alpha}_i \hat{\beta}_j$;
- Pairing $\{[\hat{\alpha}_i]_1\}_i$ with $\{\left[\hat{\beta}_j \hat{\boldsymbol{y}}_j^*\right]_2\}_j$: $\sum_{i,j,l} \theta_{i,j,l} \hat{\alpha}_i \hat{\beta}_j \left(\hat{\boldsymbol{y}}_j^*\right)_l$;
- Pairing $\{[\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1\}_i$ by a constant in $\mathbb{G}_2$: $\sum_{i,k} \psi_{i,k} \hat{\alpha}_i \left(\hat{\boldsymbol{x}}_i^*\right)_k$;
- Pairing $\{[\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1\}_i$ with $\{\left[\hat{\beta}_j\right]_2\}_j$: $\sum_{i,j,k} \tau_{i,j,k} \hat{\alpha}_i \hat{\beta}_j \left(\hat{\boldsymbol{x}}_i^*\right)_k$;
- Pairing $\{[\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1\}_i$ with $\{\left[\hat{\beta}_j \hat{\boldsymbol{y}}_j^*\right]_2\}_j$: $\sum_{i,j,k,l} \eta_{i,j,k,l} \hat{\alpha}_i \hat{\beta}_j \left(\hat{\boldsymbol{x}}_i^*\right)_k \left(\hat{\boldsymbol{y}}_j^*\right)_l$;
- Pairing $\{\left[\hat{\beta}_j\right]_2\}_j$ by a constant in $\mathbb{G}_1$: $\sum_j \delta_j \hat{\beta}_j$;
- Pairing $\{\left[\hat{\beta}_j \hat{\boldsymbol{y}}_j^*\right]_2\}_j$ by a constant in $\mathbb{G}_1$: $\sum_{j,l} \mu_{j,l} \hat{\beta}_j \left(\hat{\boldsymbol{y}}_j^*\right)_l$.

Secondly, we write the following expression, which is obtained by a linear combination of all the listed elements:

$$\rho + \sum_i \gamma_i \hat{\alpha}_i + \sum_{i,j} \kappa_{i,j} \hat{\alpha}_i \hat{\beta}_j + \sum_{i,j,l} \theta_{i,j,l} \hat{\alpha}_i \hat{\beta}_j \left(\hat{\boldsymbol{y}}_j^*\right)_l + \sum_{i,k} \psi_{i,k} \hat{\alpha}_i \left(\hat{\boldsymbol{x}}_i^*\right)_k + \sum_{i,j,k} \tau_{i,j,k} \hat{\alpha}_i \hat{\beta}_j \left(\hat{\boldsymbol{x}}_i^*\right)_k + \sum_{i,j,k,l} \eta_{i,j,k,l} \hat{\alpha}_i \hat{\beta}_j \left(\hat{\boldsymbol{x}}_i^*\right)_k \left(\hat{\boldsymbol{y}}_j^*\right)_l + \sum_j \delta_j \hat{\beta}_j + \sum_{j,l} \mu_{j,l} \hat{\beta}_j \left(\hat{\boldsymbol{y}}_j^*\right)_l .$$

Rewrite it as

$$\rho + \sum_i \hat{\alpha}_i \left(\gamma_i + \sum_k \psi_{i,k} \left(\hat{\boldsymbol{x}}_i^*\right)_k \right) + \sum_j \hat{\beta}_j \left(\delta_j + \sum_l \mu_{j,l} \left(\hat{\boldsymbol{y}}_j^*\right)_l \right) + \sum_{i,j} \hat{\alpha}_i \hat{\beta}_j \left(\kappa_{i,j} + \sum_l \theta_{i,j,l} \left(\hat{\boldsymbol{y}}_j^*\right)_l + \sum_k \tau_{i,j,k} \left(\hat{\boldsymbol{x}}_i^*\right)_k \right) + \sum_{i,j} \hat{\alpha}_i \hat{\beta}_j \left(\sum_{k,l} \eta_{i,j,k,l} \left(\hat{\boldsymbol{x}}_i^*\right)_k \left(\hat{\boldsymbol{y}}_j^*\right)_l \right) . \tag{1}$$

Now for any potentially successful zero-test queried by $\mathcal{A}$, this must result in an identically zero rational function for expression (1). Thus, the following conditions must hold:

1) Obviously, $\rho = 0$.

2) For the terms that only contain $\hat{\alpha}_i$, we can get that for all $i \in [Q_1]$, $\gamma_i + \sum_k \psi_{i,k} \left(\hat{\boldsymbol{x}}_i^*\right)_k = 0$. Note that $\hat{\boldsymbol{x}}_i^* = \hat{\boldsymbol{R}}^T \cdot \bar{\boldsymbol{x}}_i$ and $(\bar{\boldsymbol{x}}_i)_k = \sum_{j=1}^n x_j \hat{t}_j \omega^{(k-1)(j-1)}$. In this case, if $\boldsymbol{x}_i$ are non-zero vectors, we have that every entry in $\bar{\boldsymbol{x}}_i$ is non-zero. Therefore, the formal variable $\hat{r}_k$ only appears in $\psi_{i,k} \left(\hat{\boldsymbol{x}}_i^*\right)_k$. Then $\psi_{i,k}$ must be 0 for all $i \in [Q_1], k \in [n]$, and hence $\gamma_i = 0$ for $i \in [Q_1]$.

3) For the terms that only contain $\hat{\beta}_j$, we can get that for all $j \in [Q_2]$, $\delta_j + \sum_l \mu_{j,l} \left(\hat{\boldsymbol{y}}_j^*\right)_l = 0$. Similar to 2), for all $j \in [Q_2]$ and $l \in [n]$, $\delta_j = 0$ and $\mu_{j,l} = 0$.

4) For the terms contain $\hat{\alpha}_i \hat{\beta}_j$, we can get that for all $i \in [Q_1], j \in [Q_2]$,

$$\kappa_{i,j} + \sum_l \theta_{i,j,l} \left(\hat{\boldsymbol{y}}_j^*\right)_l + \sum_k \tau_{i,j,k} \left(\hat{\boldsymbol{x}}_i^*\right)_k + \sum_{k,l} \eta_{i,j,k,l} \left(\hat{\boldsymbol{x}}_i^*\right)_k \left(\hat{\boldsymbol{y}}_j^*\right)_l = 0.$$

Similar to 2) and 3), we can get that for $i \in [Q_1], j \in [Q_2], k \in [n], l \in [n]$, $\theta_{i,j,l} = 0$ and $\tau_{i,j,k} = 0$. In this case, the initial expression (1) now can be rewritten as,

$$\sum_{i,j} \hat{\alpha}_i \hat{\beta}_j \left(\kappa_{i,j} + \sum_{k,l} \eta_{i,j,k,l} \left(\hat{\boldsymbol{x}}_i^*\right)_k \left(\hat{\boldsymbol{y}}_j^*\right)_l \right) \quad (2)$$

By Lemma 6, for each i, j, the coefficients $\{\eta_{i,j,k,l}\}_{k,l}$ must be set to induce a scaling of the inner product of $\hat{\boldsymbol{x}}_i^*$ and $\hat{\boldsymbol{y}}_j^*$. Supposing that $c_{i,j}$ is the scaling, we rewrite the expression (2) as follows:

$$\begin{aligned} &\sum_{i,j} \hat{\alpha}_i \hat{\beta}_j (\kappa_{i,j} + c_{i,j} \langle \bar{\boldsymbol{x}}_i, \bar{\boldsymbol{y}}_j \rangle) \\ =& \sum_{i,j} \hat{\alpha}_i \hat{\beta}_j (\kappa_{i,j} + c_{i,j} \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle) \\ =& \sum_{i,j} \hat{\alpha}_i \hat{\beta}_j (\kappa_{i,j} + c_{i,j} \mathcal{C}_{\mathsf{ip}}(i,j)), \end{aligned}$$

where $\mathcal{C}_{\mathsf{ip}}(i,j)$ is the collection of mappings defined before.

Observe that in Exp_1, the challenger $\mathcal{C}$ behaves as the simulator we described at the beginning of the proof without knowing information about the actual value of $\{\boldsymbol{x}_i\}_i$ and $\{\boldsymbol{y}_j\}_j$ but only $\{\langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle\}_{i,j}$. Overall, $\mathsf{Real}_{\mathcal{A}}(1^\lambda) = \mathsf{Exp}_0 \approx \mathsf{Exp}_1 = \mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$. This completes the proof. □

5.5 PK-ER-SIM Security

Here we prove the PK-ER-SIM security of Π in the generic group model.

Theorem 8. *The FH-IPFE scheme Π is PK-ER-SIM-secure in the generic group model.*

Proof. The proof is similar to the generic group proofs given in Theorem 7. We start by defining the simulation-based security in the generic group model.

1) *Real Experiment* $\mathsf{Real}_{\mathcal{A}}(1^\lambda)$:

- $\mathsf{Setup}(1^\lambda)$: for generators g_1, g_2, and g_T, the challenger queries the oracle $\mathsf{BG.Encode}$ and returns the handles $\{h_i\}_{i \in \{1,2,T\}}$. For convenience, we denote handle h_i as $[1]_i$.
- $\mathcal{O}_{\mathsf{KeyGen}'}(\mathsf{pp}, \mathsf{msk}, \cdot)$: for any vector $\boldsymbol{x}_i$ corresponding to the i-th key generation queries, the challenger queries the oracle $\mathsf{BG.Encode}$ and returns the handles $[\alpha_i \boldsymbol{x}_i^*]_1$.
- $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pp}, \mathsf{msk}, \cdot)$: for any vector $\boldsymbol{y}_j$ corresponding to the j-th encryption queries, the challenger queries the oracle $\mathsf{BG.Encode}$ and returns the handles $[\beta_j]_2$ and $\left[\beta_j \boldsymbol{y}_j^*\right]_2$.
- The adversary also has the ability to query other GGM oracles such as $\mathsf{BG.Add}$, $\mathsf{BG.Pair}$, and $\mathsf{BG.ZeroTest}$, the challenger just simply forwards those queries and returns the results.

2) *Ideal Experiment* $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$:

- The simulator $\mathcal{S}_1$ generates a random handle for each group generator as the public parameters, and returns all the handles to the adversary.
- $\mathcal{O}'_{\mathsf{KeyGen}'}(\mathsf{pp}, \cdot)$: On input a vector $\boldsymbol{x}_i$, the simulator $\mathcal{S}_2$ receives as input a new collect $\mathcal{C}'_{\mathsf{ip}}$ of mappings (instead of real $\boldsymbol{x}_i$) and updates $\mathcal{C}_{\mathsf{ip}} \leftarrow \mathcal{C}'_{\mathsf{ip}}$. Then $\mathcal{S}_2$ generates fresh handles $[\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1$ and responds with the second part of a secret key $K_2^{(i)} = [\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1$.
- $\mathcal{O}'_{\mathsf{Enc}}(\mathsf{pp}, \cdot)$: On input a vector $\boldsymbol{y}_j$, the simulator $\mathcal{S}_3$ receives as input a new collect $\mathcal{C}'_{\mathsf{ip}}$ of inner products (instead of real $\boldsymbol{y}_j$) and updates $\mathcal{C}_{\mathsf{ip}} \leftarrow \mathcal{C}'_{\mathsf{ip}}$. Then $\mathcal{S}_3$ generate fresh handles $\left[\hat{\beta}_j\right]_2$, $\left[\hat{\beta}_j \hat{\boldsymbol{y}}_j^*\right]_2$ and responds with a ciphertext $\mathsf{ct}_{\boldsymbol{y}_j} = \left(\left[\hat{\beta}_j\right]_2, \left[\hat{\beta}_j \hat{\boldsymbol{y}}_j^*\right]_2\right)$.

- The adversary also has the ability to query other GGM oracles. For $\mathsf{BG.Add}$, $\mathsf{BG.Pair}$, the simulator $\mathcal{S}$ honestly returns the corresponding handles but for $\mathsf{BG.ZeroTest}$, the simulator $\mathcal{S}$ returns the output *zero* if and only if

$$\sum_j \hat{\beta}_j \sum_i c_{i,j} \mathcal{C}_{\mathsf{ip}}(i,j) = 0,$$

where $c_{i,j}$ represent coefficients submitted by a $\mathcal{A}$ and $\mathcal{C}_{\mathsf{ip}}(i,j)$ are the collection of mappings defined before.

Let Q_1 and Q_2 be the total number of key generation and encryption oracle queries made by $\mathcal{A}$, respectively. We present two hybrid experiments, beginning with the original real experiment.

- Exp_0: Exp_0 is the real experiment and the adversary has access to the handles of elements $\{[\alpha_i \boldsymbol{x}_i^*]_1\}_i$, $\{[\beta_j]_2\}_j$, and $\{[\beta_j \boldsymbol{y}_j^*]_2\}_j$.
- Exp_1: Exp_1 is obtained from Exp_0 by modifying the random elements chosen by challenger $\mathcal{C}$ to formal variables. We can imagine replacing $\{\alpha_i\}_i$, $\{\beta_j\}_j$, $\boldsymbol{r}$, $\boldsymbol{t}$, and $\boldsymbol{s}$ with formal variables. In this case, for every zero-test query submitted by $\mathcal{A}$, the resulting zero-test expressions are substituted by rational functions of above variables. By construction, this results in a polynomial of degree at most $(n+1)$ over the formal variables. Hence, for each zero-test, the difference between Exp_0 and Exp_1 is $\frac{n+1}{q}$, due to the Schwartz-Zippel lemma. Assuming that $\mathcal{A}$ makes zero-tests for p (which is a polynomial size) times, by union bound, the difference between Exp_0 and Exp_1 can be bounded by $\frac{p(n+1)}{q}$, which is negligible. In other words, $\mathcal{A}$ cannot distinguish this switch except with negligible probability.

Now in Exp_1, we rewrite the handles of the elements given to $\mathcal{A}$: $\{[\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1\}_i$, $\{\left[\hat{\beta}_j\right]_2\}_j$, and $\{\left[\hat{\beta}_j \hat{\boldsymbol{y}}_j^*\right]_2\}_j$. We consider all the possible ways that $\mathcal{A}$ can produce elements in the target group and write a general linear combinations of such elements, where ρ, $\psi_{i,k}$, $\tau_{i,j,k}$, $\eta_{i,j,k,l}$, δ_j, and $\mu_{j,l}$ represent coefficients submitted by $\mathcal{A}$. As proofs given in Theorem 7, we list the following elements:

- Adding by a constant in $\mathbb{G}_T$: ρ;
- Pairing $\{[\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1\}_i$ by a constant in $\mathbb{G}_2$: $\sum_{i,k} \psi_{i,k} \hat{\alpha}_i (\hat{\boldsymbol{x}}_i^*)_k$;
- Pairing $\{[\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1\}_i$ with $\{\left[\hat{\beta}_j\right]_2\}_j$: $\sum_{i,j,k} \tau_{i,j,k} \hat{\alpha}_i \hat{\beta}_j (\hat{\boldsymbol{x}}_i^*)_k$;
- Pairing $\{[\hat{\alpha}_i \hat{\boldsymbol{x}}_i^*]_1\}_i$ with $\{\left[\hat{\beta}_j \hat{\boldsymbol{y}}_j^*\right]_2\}_j$: $\sum_{i,j,k,l} \eta_{i,j,k,l} \hat{\alpha}_i \hat{\beta}_j (\hat{\boldsymbol{x}}_i^*)_k (\hat{\boldsymbol{y}}_j^*)_l$;
- Pairing $\{\left[\hat{\beta}_j\right]_2\}_j$ by a constant in $\mathbb{G}_1$: $\sum_j \delta_j \hat{\beta}_j$;
- Pairing $\{\left[\hat{\beta}_j \hat{\boldsymbol{y}}_j^*\right]_2\}_j$ by a constant in $\mathbb{G}_1$: $\sum_{j,l} \mu_{j,l} \hat{\beta}_j (\hat{\boldsymbol{y}}_j^*)_l$.

The following expression is obtained by a linear combination of all the above listed elements:

$$\rho + \sum_{i,k} \psi_{i,k} \hat{\alpha}_i (\hat{\boldsymbol{x}}_i^*)_k + \sum_{i,j,k} \tau_{i,j,k} \hat{\alpha}_i \hat{\beta}_j (\hat{\boldsymbol{x}}_i^*)_k + \sum_{i,j,k,l} \eta_{i,j,k,l} \hat{\alpha}_i \hat{\beta}_j (\hat{\boldsymbol{x}}_i^*)_k (\hat{\boldsymbol{y}}_j^*)_l + \sum_j \delta_j \hat{\beta}_j + \sum_{j,l} \mu_{j,l} \hat{\beta}_j (\hat{\boldsymbol{y}}_j^*)_l .$$

Rewrite it as

$$\rho + \sum_i \hat{\alpha}_i \left(\sum_k \psi_{i,k} (\hat{\boldsymbol{x}}_i^*)_k \right) + \sum_j \hat{\beta}_j \left(\delta_j + \sum_l \mu_{j,l} (\hat{\boldsymbol{y}}_j^*)_l \right) + \sum_{i,j} \hat{\alpha}_i \hat{\beta}_j \left(\sum_k \tau_{i,j,k} (\hat{\boldsymbol{x}}_i^*)_k + \sum_{k,l} \eta_{i,j,k,l} (\hat{\boldsymbol{x}}_i^*)_k (\hat{\boldsymbol{y}}_j^*)_l \right) . \tag{3}$$

Now for any potentially successful zero-test queried by $\mathcal{A}$, this must result in an identically zero rational function for expression (3). It implies the following conditions:

1) Obviously, $\rho = 0$.
2) For the terms that only contain $\hat{\alpha}_i$, we can get that for all $i \in [Q_1]$, $\sum_k \psi_{i,k} (\hat{\boldsymbol{x}}_i^*)_k = 0$. Note that $(\hat{\boldsymbol{x}}_i^*) = \hat{\boldsymbol{R}}^T \cdot \bar{\boldsymbol{x}}_b^i$ and $(\bar{\boldsymbol{x}}_i)_k = \sum_{j=1}^n x_j \hat{t}_j \omega^{(k-1)(j-1)}$. In this case, if $\boldsymbol{x}_i$ are non-zero vectors, we have that every entry in $\bar{\boldsymbol{x}}_i$ is non-zero. Therefore, the formal variable $\hat{r}_k$ only appears in $\psi_{i,k} (\hat{\boldsymbol{x}}_i^*)_k$. Then $\psi_{i,k}$ must be 0 for all $i \in [Q_1], k \in [n]$.

3) For the terms that only contain $\hat{\beta}_j$, we can get that for all j, $\delta_j + \sum_l \mu_{j,l}\left(\hat{\boldsymbol{y}}_j^*\right)_l = 0$. Similar to 2), for all $j \in [n]$ and $l \in [n]$, $\mu_{j,l} = 0$ and hence $\delta_j = 0$ for $j \in [n]$.

4) For the terms contains $\hat{\alpha}_i\hat{\beta}_j$, we can get that for all $i \in [n], j \in [n]$,

$$\sum_k \tau_{i,j,k}\left(\hat{\boldsymbol{x}}_i^*\right)_k + \sum_{k,l} \eta_{i,j,k,l}\left(\hat{\boldsymbol{x}}_i^*\right)_k\left(\hat{\boldsymbol{y}}_j^*\right)_l = 0.$$

Similar to 2) and 3), we first get that for $i \in [n], j \in [n], k \in [n]$, $\tau_{i,j,k} = 0$. The expression (3) now can be rewritten as,

$$\sum_{i,j} \hat{\alpha}_i\hat{\beta}_j\Big(\sum_{k,l} \eta_{i,j,k,l}\left(\hat{\boldsymbol{x}}_i^*\right)_k\left(\hat{\boldsymbol{y}}_j^*\right)_l\Big). \tag{4}$$

By Lemma 6, for each i, j, the coefficients $\{\eta_{i,j,k,l}\}_{k,l}$ must be set to induce a scaling of the inner product of $\hat{\boldsymbol{x}}_i^*$ and $\hat{\boldsymbol{y}}_j^*$. Supposing that $c_{i,j}$ is the scaling, we rewrite the expression (4) as follows:

$$\sum_{i,j} \hat{\alpha}_i\hat{\beta}_j c_{i,j}\langle \bar{\boldsymbol{x}}_i, \bar{\boldsymbol{y}}_j\rangle = \sum_{i,j} c_{i,j}\langle \boldsymbol{x}_i, \boldsymbol{y}_j\rangle. \tag{5}$$

Now the expression (5) for zero-test can be shown as

$$\begin{aligned}\sum_{i,j} \hat{\alpha}_i\hat{\beta}_j c_{i,j}\langle \boldsymbol{x}_i, \boldsymbol{y}_j\rangle &= \sum_j \hat{\beta}_j \sum_i c_{i,j}\hat{\alpha}_i\langle \boldsymbol{x}_i, \boldsymbol{y}_j\rangle \\ &= \sum_j \hat{\beta}_j \sum_i c_{i,j}\mathcal{C}_{\mathsf{ip}}(i,j).\end{aligned}$$

Observe that now in Exp_1, the challenger $\mathcal{C}$ behaves as the simulator we described at the beginning of the proof without knowing information about the actual inner product value of $\{\boldsymbol{x}_i\}_i$ and $\{\boldsymbol{y}_j\}_j$ but only $\{\alpha_i\langle \boldsymbol{x}_i, \boldsymbol{y}_i\rangle\}_{i,j}$. Overall, $\mathsf{Real}_{\mathcal{A}}(1^\lambda) = \mathsf{Exp}_0 \approx \mathsf{Exp}_1 = \mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$. This completes the proof.

□

5.6 Discussion

It is very interesting to consider whether our NTT technique can be applied to other function-hiding functional encryption. Bartusek *et al.* [15] constructed a public-key function-hiding predicate encryption (FH-PE) for "small superset predicates", where they also use a uniformly random matrix $\boldsymbol{R}$ to hide the information of underlying attributes and predicates. However, in their construction, the public parameters contain an encode of $\boldsymbol{R}$ as the form $[\boldsymbol{R}]_2$, which will be delivered to adversary $\mathcal{A}$ in the security games. When considering the expressions that $\mathcal{A}$ submits to the zero-test oracle in generic group model, there will be additional monomials consisting of $c_{i,j}\hat{r}_{i,j}$. In this case, our technique fails since we cannot deal with those monomials with the structured $\boldsymbol{R}$. Note that in the secret-key setting, the issues do not exist due to the fact that $\mathcal{A}$ has no access to any encodes of $\boldsymbol{R}$ in $\mathbb{G}_1$ or $\mathbb{G}_2$.

6 Application to Fine-grained Data Sharing

In this section, we show the application of our scheme in fine-grained data sharing system.

6.1 Overview of Our Fine-Grained Data Sharing System

Our system is divided into data owners and an untrusted server. Each data owner is provided with a master key for the FH-IPFE scheme, and the server stores sensitive data in the encrypted form (say $\mathsf{ct}_{\boldsymbol{y}}$) under the master key.

On the data owners side, our system supports basic operations such as insert, delete, query and update. Furthermore, a data owner can also make some statistical analyses such as mean, weighted mean and variance values. Those can be done by issuing partial secret key (K_2) for the server to process (by doing partial decryption) on encrypted data. The server gets an intermediate value of decryption and returns it to the owner. Then the data owner uses K_1 for final decryption and gets the actual value. With appropriately issued keys (K_1, K_2), the data owner can achieve various operations. In more detail, the data owner generates different secret keys by embedding corresponding vectors. For any fixed data vector $\boldsymbol{y}$, we can choose appropriate vector $\boldsymbol{x}$ and the inner product value $\langle \boldsymbol{x}, \boldsymbol{y}\rangle$ helps us to implement various operations (See Section 6.3 for more details). Meanwhile, the security is guaranteed by the PK-ER-SIM security of FH-IPFE.

On the server side, our system also supports above statistical analyses with the data owner's delegated secret keys (both K_1 and K_2 are sent to the server) and the security is guaranteed by the standard security of FH-IPFE.

6.2 Initialization

Let $N = 2^m$ denote the maximal dimension of the data vector. The initialization steps of the interaction between each data owner and the server are as follows:

Step 1. For the data $\boldsymbol{y}$ (viewed as a vector with only positive elements), let $n = |\boldsymbol{y}|$ and $y_i = 0$[1] for $i \in [n+1, N]$.

Step 2. Each data owner runs the setup algorithm $\mathsf{Setup}\left(1^\lambda, S\right)$ to get the public parameter pp and the master key msk, where $\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, S)$ and $\mathsf{msk} = (\boldsymbol{r}, \boldsymbol{t}, \boldsymbol{s})$.

Step 3. Each data owner runs the encryption algorithm $\mathsf{Enc}\left(\mathsf{msk}, \boldsymbol{y}\right)$ to get the ciphertext $\mathsf{ct}_{\boldsymbol{y}}$ and the key generation algorithm $\mathsf{KeyGen}\left(\mathsf{msk}, \boldsymbol{y}\right)$ to get the secret key $\mathsf{sk}_{\boldsymbol{y}}$, where $\mathsf{ct}_{\boldsymbol{y}} = (C_1, C_2)$ and $\mathsf{sk}_{\boldsymbol{y}} = (J_1, J_2)$.

Step 4. Each data owner sends $\mathsf{pp}, n, n_{max} = n, C_1$, C_2, and J_2[2] to the server. The owner holds msk and J_1 on his/her own side.

After the initialization steps are completed, we now describe how the proposed system supports the various operations. We describe the system for the data owners side and the server side in Section 6.3 and Section 6.4, respectively.

6.3 Data Owners Side

6.3.1 Basic Operations

We implement all the basic operations by choosing appropriate vectors to calculate the inner product values (without leaking the vectors themselves). Hence, we first construct some vectors for different types of operations.

1) For query operation in position i, let vector $\boldsymbol{x} = (0, ..., 0, 1, 0, ..., 0)$ where i-th entry is 1 and other entries are all zero.

2) For update operation in position i, let vector $\boldsymbol{y}' = (0, ..., 0, y - y_i, 0, ..., 0)$ where y_i is the value returned by the query operation and y is the value to be updated.

3) For insert operation in position $(n_{max}+1)$, let vector $\boldsymbol{y}' = (0, ..., 0, y_{n_{max}+1}, 0, ..., 0)$ and set $n = n+1$, $n_{max} = n_{max}+1$. Here $y_{n_{max}+1}$ is the value to be inserted.

4) For delete operation in position i, let vector $\boldsymbol{y}' = (0, ..., 0, -y_i, 0, ..., 0)$ where y_i is also the value returned by the query operation. Additionally, set $n = n - 1$.

Now we show that how the proposed system support various basic operations. We start with the query operation. Subsequently, it works as a subroutine to implement operations insert, update, and delete.

1) **Query:** If a data owner wants to query the data that indexed by i (i.e., the i-th entry), the interaction between the data owner and the server is as follows:

Step 1. The data owner runs the key generation algorithm $\mathsf{KeyGen}\left(\mathsf{msk}, \boldsymbol{x}\right)$ to get the secret key $\mathsf{sk}_{\boldsymbol{x}}$, where $\mathsf{sk}_{\boldsymbol{x}} = (K_1, K_2)$. Then it holds K_1 and sends K_2 to the server.

Step 2. The server computes $D_2 = e\left(K_2, C_2\right)$ and returns D_2 to the data owner.

Step 3. The data owner computes $D_1 = e\left(K_1, C_1\right)$, finds z such that $D_1^z = D_2$ and outputs z.

It is easy to verify that $D_1 = [\alpha\beta]_T$, $D_2 = [\alpha\beta y_i]_T$, and $z = y_i$ due to the correctness of

[1] For an entry that is null, we set it as 0.
[2] Note that each data owner does not store the original data $\boldsymbol{y}$, and J_2 is used to calculate variance. Thus J_2 is generated in this phase.

our FH-IPFE scheme in Section 5. For security, we have proven the PK-ER-SIM security of the proposed FH-IPFE scheme. It guarantees that even if the server receives K_2, it learns nothing about $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = y_i$ as long as it does not hold K_1. In other words, in the query operation, the data owner securely gets y_i from the server.

2) **Update, Insert and Delete:** Note that in our system, the operations insert, update, and delete can all be seen as the update operation. The main idea is that we view 0 as useless data, thus if we want to delete a data, we update it to be 0 and to insert a data, we update 0 to a new value. The only difference between them is the construction of vectors $\boldsymbol{y}'$. Here we only describe the update operation. If a data owner wants to update $\boldsymbol{y}$ in position i (i.e., the i-th entry) with value y, the interaction between the data owner and the server is as follows:

Step 1. The data owner queries the element y_i indexed by i using the query operation described before. Note that for insert operation, this step is omitted and just set $i = n_{max} + 1$.

Step 2. The data owner selects corresponding $\boldsymbol{y}'$ according to operations.

Step 3. The data owner computes $C_2' = C_1^{\boldsymbol{R}^T \cdot \boldsymbol{T} \cdot \boldsymbol{D} \cdot \boldsymbol{y}'}$ and $J_2' = J_1^{\boldsymbol{R}^{-1} \cdot \boldsymbol{T}^{-1} \cdot \boldsymbol{D}^{-1} \cdot \boldsymbol{y}'}$, and sends them to the server.

Step 4. Upon receiving C_2' and J_2', the server updates $C_2 = C_2 \cdot C_2'$ and $J_2 = J_2 \cdot J_2'$[3].

The correctness follows by the fact that

$$\begin{aligned} C_2 &= \left[\beta \boldsymbol{R}^T \cdot \boldsymbol{T} \cdot \boldsymbol{D} \cdot \boldsymbol{y}'\right]_2 \cdot \left[\beta \boldsymbol{R}^T \cdot \boldsymbol{T} \cdot \boldsymbol{D} \cdot \boldsymbol{y}\right]_2 \\ &= \left[\beta \boldsymbol{R}^T \cdot \boldsymbol{T} \cdot \boldsymbol{D} \cdot (\boldsymbol{y} + \boldsymbol{y}')\right]_2, \end{aligned}$$

and

$$\begin{aligned} J_2 &= \left[\alpha \boldsymbol{R}^{-1} \cdot \boldsymbol{T}^{-1} \cdot \boldsymbol{D}^{-1} \cdot \boldsymbol{y}'\right]_1 \\ &\quad \cdot \left[\alpha \boldsymbol{R}^{-1} \cdot \boldsymbol{T}^{-1} \cdot \boldsymbol{D}^{-1} \cdot \boldsymbol{y}\right]_1 \\ &= \left[\alpha \boldsymbol{R}^{-1} \cdot \boldsymbol{T}^{-1} \cdot \boldsymbol{D}^{-1} \cdot (\boldsymbol{y}' + \boldsymbol{y})\right]_1. \end{aligned}$$

It is easy to see that C_2 and J_2 are generated by the encryption and key generation algorithms of our FH-IPFE scheme, respectively. That is, the vector $(\boldsymbol{y}' + \boldsymbol{y})$ is embed in C_2 and J_2. Furthermore, the security is simply followed by the PK-ER-SIM security of underlying FH-IPFE scheme.

6.3.2 *Statistical Analyses*

As above, we first construct two vectors for calculating inner product values. Those vectors help us to implement statistical analyses for mean and weighted mean. See below:

1) For calculating mean value, let vector $\boldsymbol{x} = (1, ..., 1, ..., 1)$ where all the entries are 1.

2) For calculating weighted mean value, let vector $\boldsymbol{x} = (w_1, ..., w_{n_{max}}, 0, ..., 0)$ where $w_1, ..., w_{n_{max}}$ are all weight values.

On the other hand, for calculating variance values, we utilize the calculation of mean value as a subroutine to process on the encrypted data. And we have no need to construct any additional vectors. The point is that in the initialization phase, the data owner has already sent J_2 to the server. It implies that the data owner can get $\boldsymbol{y}^2$ using J_1 and he/she can compute $\mathsf{var} = \frac{\boldsymbol{y}^2}{n} - \mathsf{avg}^2$ as the variance value.

1) **Mean and Weighted Mean:** The only difference between operations calculating mean and weighted mean values is located in choosing the vector $\boldsymbol{x}$ and the rest of steps are the same. Hence, we describe them together. If a data owner wants to calculate mean or weighted mean values of all data (i.e., the full data vector), the interaction between the data owner and the server is as follows:

Step 1. The data owner selects $\boldsymbol{x}$ depending on calculating mean or weighted mean values.

Step 2. The data owner runs the key generation algorithm $\mathsf{KeyGen}(\mathsf{msk}, \boldsymbol{x})$ to get the secret key

[3] J_2 is also updated for calculating variance values.

$\mathsf{sk}_{\boldsymbol{x}}$, where $\mathsf{sk}_{\boldsymbol{x}} = (K_1, K_2)$. Then it holds K_1 and sends K_2 to the server.

Step 3. The server computes $D_2 = e(K_2, C_2)$ and returns D_2 to the data owner.

Step 4. The data owner computes $D_1 = e(K_1, C_1)$ and finds z such that $D_1^z = D_2$. Then the mean or weighted mean value is outputted by $\mathsf{avg} = \frac{z}{n}$.

Here $D_1 = [\alpha\beta]_T$ and $D_2 = [\alpha\beta\langle \boldsymbol{x}, \boldsymbol{y}\rangle]_T$. For appropriately chosen $\boldsymbol{x}$, the correctness follows.

2) **Variances:** If a data owner wants to calculate variance value of the full data vector, the interaction between the data owner and the server is as follows:

Step 1. The data owner calculates the mean value avg by the method described above.

Step 2. The server computes $D_2 = e(J_2, C_2)$ and returns D_2 to the data owner.

Step 3. The data owner computes $D_1 = e(J_1, C_1)$ and finds z such that $D_1^z = D_2$.

Step4. Finally, the variance value is outputted by $\frac{z}{n} - \mathsf{avg}^2$.

It is easy to verify that $z = \langle \boldsymbol{y}, \boldsymbol{y}\rangle$ and $\mathsf{var} = \frac{\langle \boldsymbol{y},\boldsymbol{y}\rangle}{n} - (\frac{\sum y_i}{n})^2$. The correctness follows.

For all statistical analyses on the data owners side, the security can also be guaranteed by the PK-ER-SIM security of underlying FH-IPFE schemes.

6.4 Server Side

6.4.1 Statistical Analyses

On the server side, the proposed system can also support various statistical analyses. Based on the statistical analyses given in Section 6.3, if the server requests for a statistical analysis, the data owner delegates the full secret key K_1 and K_2 (instead of only K_2) to the server. In this case, the server has the ability to do complete decryption and receives the actual statistical result.

As before, we first construct two vectors for calculating mean and weighted mean values. See below:

1) For calculating mean value, let vector $\boldsymbol{x} = (1, ..., 1, ..., 1)$ where all the entries are 1.

2) For calculating weighted mean value, let vector $\boldsymbol{x} = (w_1, ..., w_{n_{max}}, 0, ..., 0)$ where $w_1, ..., w_{n_{max}}$ are all weight values.

Similar to Section 6.3.2, there is no need to construct vectors for calculating variance values and we utilize the calculation of mean values as a subroutine. Now we start with describing how the server can compute mean or weighted values.

1) **Mean and Weighted Mean:** If the server wants to calculate mean or weighted mean values of all data (i.e., the full vector), the interaction between the data owner and the server is as follows:

Step 1. The data owner selects $\boldsymbol{x}$ depending on calculating mean or weighted mean values.

Step 2. The data owner runs the key generation algorithm $\mathsf{KeyGen}(\mathsf{msk}, \boldsymbol{x})$ to get the secret key $\mathsf{sk}_{\boldsymbol{x}}$, where $\mathsf{sk}_{\boldsymbol{x}} = (K_1, K_2)$. Then it sends K_1 and K_2 to the server.

Step 3. The server computes $D_1 = e(K_1, C_1)$, $D_2 = (K_2, C_2)$ and finds z such that $D_1^z = D_2$. Then the mean or weighted value is outputted by $\mathsf{avg} = \frac{z}{n}$.

The correctness is similar to the case on the data owners side, for appropriately chosen $\boldsymbol{x}$.

2) **Variance:** On the other hand, if the server wants to calculate variance values, the interaction between the data owner and the server is as follows:

Step 1. The server calculates the mean value avg by the method described above.

Step 2. The data owner sends J_1 to the server.

Step 3. The server computes $D_1 = e(J_1, C_1)$, $D_2 = e(J_2, C_2)$, and finds z such that $D_1^z = D_2$. Note that J_2 is sent to the server in the initialization phase.

Table 4. Running Time(s) on Several Different Values of n and Comparison with Kim *et al.* [7]

Dimension n	Setup(s)		Key Generation(s)		Enc(s)		Dec(s)	
	[7]	This work	[7]	This work	[7]	This work	[7]	This work
512	≥ 100	0.039	0.508	0.317	2.539	2.328	2.243	2.163
1024	≥ 800	0.051	1.357	0.637	5.389	4.632	4.152	4.010
2048	≥ 1600	0.055	4.268	1.292	12.322	9.344	8.065	7.965

Table 5. Running Time(s) of Proposed Fine-grained Data Sharing System

Operations		Data Owner(s)	Server(s)
Data Owners Side	Initialization	0.191	N/A
	Query	0.147	0.263
	Update, Insert, or Delete	0.181	0.002
	Mean or Weighted Mean	0.155	0.264
	Variance	0.333	0.527
Server Side	Mean or Weighted Mean	0.148	0.267
	Variance	0.278	0.581

Step 4. Finally, the variance value is outputted by $\frac{z}{n} - \mathsf{avg}^2$.

The correctness is also similar to the case on the data owners side.

For all statistical analyses on the server side, the security is followed by the SIM security of underlying FH-IPFE schemes. The server gets full secret key K_1, K_2, J_1, J_2 and ciphertext C_1, C_2 but no additional information about $\boldsymbol{y}$.

7 Experiments

To evaluate the practicality of our main construction, we implement our FH-IPFE as well as fine-grained data sharing system. Our implementation uses the RELIC [4] library to implement the pairing group operations and the finite field arithmetic in $\mathbb{Z}_q$. In our experiments, we measure the time needed to run Setup (generate master key), Enc (encrypt), Key Generation (issue secret keys), and Dec (decrypt the inner product) algorithms for n-dimensional binary vectors for several different values of n.

We run all of our experiments on a Linux laptop with a 6-core Intel Core i5-10500H 2.50GHz processor and 16GB of RAM. We also implement the previously state-of-the-art FH-IPFE scheme constructed by Kim *et al.* [7] and the experiment results show that our running time of various algorithms is much shorter. We run experiments over the curve MNT224 and assume the bound of inner products as 3×10^9 when solving discrete logarithm.

For the FH-IPFE scheme proposed by Kim *et al.* [7], the running time of the Setup algorithm is dominated by the inversion of a random $n \times n$ matrix in $\mathbb{Z}_q$ where q is a 224-bit prime, corresponding to 112 bits of security. The inverse computation is done in $O\left(n^3\right)$ time and when $n \geq 512$, the Setup algorithm in the FH-IPFE scheme given by Kim *et al.* [7] is agnostic to the actual values in $\boldsymbol{R}$ and $\boldsymbol{R}^*$. Following the suggestion in [7], we measure the performance with respect to matrices $\boldsymbol{R}$ and $\boldsymbol{R}^*$ that are sampled uniformly at random. Using simulated rather than real matrices has no effect on the experiment results when focusing on Key Generation, Enc and Dec algorithms[5]. The concrete performance numbers of FH-IPFE schemes are summarized in **Table 4**.

Based on our FH-IPFE scheme with maximal dimension $n = 1024$, we also implement our fine-grained data sharing system and the concrete performance numbers are summarized in **Table 5**. It can be seen that after putting the most parts of pairing operations on a server, the data owners save lot of time when carrying out statistical analyses.

[4] Aranha D F, Gouva C P L, Markmann T, Wahby R S, Liao K. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[5] When running experiments on their Setup algorithm, we do not use this simulation method.

8 Conclusion

In this work, we construct a more efficient (secret-key) FH-IPFE scheme. We prove the security of our construction in a generic model of bilinear maps. We also formalize an additional notion of security as PK-ER-SIM for FH-IPFE and design a fine-grained data sharing system based on our construction. It supports not only basic database operations but also statistical analyses on encrypted data. The experiment results also show that our FH-IPFE scheme and designed system are efficient and practical.

Acknowledgement(s)

We thank the anonymous JCST reviewers for many helpful comments.

Conflict of Interest

The authors declare that they have no conflict of interest.

References

[1] O'Neill A. Definitional issues in functional encryption. Cryptology ePrint Archive, Paper 2010/556. https://eprint.iacr.org/2010/556.

[2] Boneh D, Sahai A, Waters B. Functional Encryption: Definitions and Challenges. In *Proc. the 8th Theory of Cryptography Conference*, Mar. 2011, pp.253-273. DOI:10.1007/978-3-662-48797-6_20.

[3] Abdalla M, Catalano D, Gay R, Ursu B. Inner-Product Functional Encryption with Fine-Grained Access Control. In *Proc. the 26th International Conference on the Theory and Application of Cryptology and Information Security*, Dec. 2020, pp.467-497. DOI:10.1007/978-3-030-64840-4_16.

[4] Lai Q Q, Liu F H, Wang Z D. New Lattice Two-Stage Sampling Technique and Its Applications to Functional Encryption - Stronger Security and Smaller Ciphertexts. In *Proc. the 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Oct. 2021, pp.498-527. DOI:10.1007/978-3-030-77870-5_18.

[5] Abdalla M, Bourse F, Caro A D, Pointcheval D. Simple Functional Encryption Schemes for Inner Products. In *Proc. the 18th IACR International Conference on Practice and Theory in Public-Key Cryptography*, Apr. 2015, pp.733-751. DOI:10.1007/978-3-662-46447-2_33.

[6] Agrawal S, Libert B, and Stehlé D. Fully secure functional encryption for inner products, from standard assumptions. In *Proc. the 36th Annual International Cryptology Conference*, Aug. 2016, pp.333-362. DOI:10.1007/978-3-662-53015-3_12.

[7] Kim S, Lewi K, Mandal A, Montgomery H, Roy A, Wu D J. Function-Hiding Inner Product Encryption Is Practical. In *Proc. the 11th International Conference on Security and Cryptography for Networks*, Sep. 2018, pp.544-562. DOI:10.1007/978-3-319-98113-0_29.

[8] Ryu D H, Jeon S Y, Hong J, Lee MK. Efficient lp distance computation using function-hiding inner product encryption for privacy-preserving anomaly detection. In *Sensors*, 2023, 23(8): 4169. DOI:10.3390/s23084169.

[9] Zheng Y D, Lu R X. Efficient Privacy-Preserving Similarity Range Query based on Pre-Computed Distances in eHealthcare. In *Proc. the 2020 IEEE Global Communications Conference*, Dec. 2020, pp.1-6. DOI:10.1109/GLOBECOM42002.2020.9322502.

[10] Bishop A, Jain A, Kowalczyk L. Function-hiding inner product encryption. In *Proc. the 21st International Conference on the Theory and Application of Cryptology and Information Security*, Dec. 2015, pp.470-491. DOI:10.1007/978-3-662-48797-6_20.

[11] Datta P, Dutta R, Mukhopadhyay S. Functional encryption for inner product with full function privacy. In *Proc. the 19th IACR International Conference on Practice and Theory in Public-Key Cryptography*, Mar. 2016, pp.164-195. DOI:10.1007/978-3-662-49384-7_7.

[12] Tomida J, Abe M, Okamoto T. Efficient Functional Encryption for Inner-Product Values with Full-Hiding Security. In *Proc. the 19th Information Security Conference*, Sep. 2016, pp.408-425. DOI:10.1007/978-3-319-45871-7_24.

[13] Brakerski Z, Gentry C, and Vaikuntanathan V. (leveled) fully homomorphic encryption without bootstrapping. In *Proc. the Innovations in Theoretical Computer Science 2012*, Jan. 2012, pp.309-325. DOI:10.1145/2090236.2090262.

[14] Gentry C, Sahai A, Waters B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Proc. the 33rd Annual Cryptology Conference*, Aug. 2013, pp.75-92. DOI:10.1007/978-3-642-40041-4_5.

[15] Bartusek J, Carmer B, Jain A, Jin Z, Lepoint T, Ma F, Malkin T, Malozemoff A J, Raykova M. Public-key function-private hidden vector encryption (and more). In *Proc. the 25th International Conference on the Theory and Application of Cryptology and Information Security*, Dec. 2019, pp.489-519. DOI:10.1007/978-3-030-34618-8_17.

[16] Good I J. Introduction to Cooley and Tukey (1965) An Algorithm for the Machine Calculation of Complex Fourier Series. In *Breakthroughs in Statistics*, Kotz S, Johnson N L (eds.), Springer Series in Statistics, 1997, pp.201-216.

[17] Joux A. A one round protocol for tripartite diffie–hellman. In *J. Cryptology*, 2003, 17(4): 263-276. DOI:10.1007/s00145-004-0312-y.

[18] Boneh D, Franklin M. Identity-based encryption from the weil pairing. In *Proc. the 21st Annual International Cryptology Conference*, Aug. 2001, pp.213-229. DOI:10.1007/3-540-44647-8_13.

[19] Boneh D, Boyen X, Shacham H. Short group signatures. In *Proc. the 24th Annual International Cryptology Conference*, Aug. 2004, pp.41-55. DOI:10.1007/978-3-540-28628-8_3.

[20] Boneh D, Boyen X, Goh E J. Hierarchical identity based encryption with constant size ciphertext. In *Proc. the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, May. 2005, pp.440-456. DOI:10.1007/11426639_26.

[21] Nechaev V I. Complexity of a determinate algorithm for the discrete logarithm. In *Math Notes*, 1994, 17: 165-172. DOI:10.1007/BF02113297.

[22] Shoup V. Lower bounds for discrete logarithms and related problems. In *Proc. the 1997 International Conference on the Theory and Application of Cryptographic Techniques*, May. 1997, pp.256-266. DOI:10.1007/3-540-69053-0_18.

[23] Schwartz J T. Fast probabilistic algorithms for verification of polynomial identities. In *J. ACM*, 1980, 27(4): 701-717. DOI:10.1145/322217.322225.

[24] Zippel R. Probabilistic algorithms for sparse polynomials. In *Proc. the 1979 International Symposium on Symbolic and Algebraic Manipulation*, Jun. 1979, pp.216-226. DOI:10.1007/3-540-09519-5_73.

[25] Bartusek J, Guan J, Ma F, Zhandry M. Return of GGH15: Provable Security Against Zeroizing Attacks. In *Proc. the 18th Theory of Cryptography Conference*, Nov. 2018, pp.544-574. DOI:10.1007/978-3-030-03810-6_20.

[26] Badrinarayanan S, Miles E, Sahai A, Zhandry M. Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. In *Proc. the 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, May. 2016, pp.764-791. DOI:10.1007/978-3-662-49896-5_27.

Ming Wan received his B.S. degree in computer science from Northwestern Polytechnical University, Xi'an, in 2017. He is currently a Ph.D. candidate at the school of electronic information and electrical engineering, Shanghai Jiao Tong University, Shanghai. His research interests include functional encryption and group-based cryptography.

Geng Wang received his Ph.D. degree in applied mathematics from Peking University, Beijing, in 2013. He is currently an assistant researcher at the school of electronic information and electrical engineering, Shanghai Jiao Tong University, Shanghai. His research interests include functional encryption and lattice-based cryptography.

Shi-Feng Sun received his Ph.D. degree in computer science from Shanghai Jiao Tong University, Shanghai, in 2016. He is currently an associate professor at the school of electronic information and electrical engineering, Shanghai Jiao Tong University, Shanghai. His research interests include cryptography and data privacy, particularly provably secure cryptosystems against physical attacks, data privacy-preserving technology in cloud storage, and privacy-enhancing technology in blockchain.

Da-Wu Gu is a distinguished professor at School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University (SJTU), Shanghai. He received from Xidian University of China, Xi'an, his B.S. degree in applied mathematics in 1992, and his M.S. degree in 1995 and Ph.D. degree in 1998, both in cryptography. His research interests include crypto algorithms, crypto engineering, and system security. He leads the Laboratory of Cryptology and Computer Security (LoCCS) at SJTU, Shanghai. He was the winner of Chang Jiang Scholars Distinguished Professors Program in 2014 by Ministry of Education of China. He won the National Award of Science and Technology Progress in 2017. He has got over 150 scientific papers in academic journals and conferences, and owned 28 innovation patents.

Gong-Yu Shi received his B.S. degree from Zhejiang University, Hangzhou, in 2020. He is currently a Master student at the school of electronic information and electrical engineering, Shanghai Jiao Tong University, Shanghai. His research interests include cryptanalysis and cryptography engineering.

Appendix

Proof of Lemma 6

Proof. We firstly write

$$\hat{\boldsymbol{R}} = \begin{pmatrix} \hat{r}_1 & \hat{s}_1 & 0 & \dots & 0 \\ 0 & \hat{r}_2 & \hat{s}_2 & \dots & 0 \\ 0 & 0 & \hat{r}_3 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & \hat{r}_n \end{pmatrix}.$$

Hence, for $\hat{\boldsymbol{R}}^{-1}$,

$$\hat{R}^{-1}_{i,j} = \begin{cases} (-1)^{j-i} \prod_{k=i}^{j} \hat{r}_k^{-1} \prod_{k=i}^{j-1} \hat{s}_k, & j \geq i \\ 0, & j < i \end{cases}.$$

Let $\hat{z}_i = \hat{r}_i^{-1}\hat{s}_i$, then we can rewrite $\hat{R}^{-1}_{i,j} = \begin{cases} (-1)^{j-i} \hat{r}_j^{-1} \prod_{k=i}^{j-1} \hat{z}_k, & j \geq i \\ 0, & j < i \end{cases}$. Now for $\hat{\boldsymbol{u}}$, we write $\hat{\boldsymbol{u}}^T = \boldsymbol{u}^T \cdot \hat{\boldsymbol{R}}^{-1} = \left(\sum_j u_j \hat{R}^{-1}_{j,1}, ..., \sum_j u_j \hat{R}^{-1}_{j,n}\right)$, we have $\hat{u}_i = \sum_j u_j \hat{R}^{-1}_{j,i} = \hat{r}_i^{-1} \sum_{j=1}^{i} (-1)^{i-j} u_j \prod_{k=j}^{i-1} \hat{z}_k$. On the other hand, for $\hat{\boldsymbol{v}}$, $\hat{v}_i$ can be written as $\hat{v}_i = \begin{cases} \hat{r}_i v_i + \hat{s}_i v_{i+1} = \hat{r}_i (v_i + v_{i+1}\hat{z}_i), & 0 \leq i \leq n-1 \\ \hat{r}_n v_n, & i = n \end{cases}$.

Since P is a polynomial over the entries of $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{v}}$ such that each monomial contains exactly one entry from $\hat{\boldsymbol{u}}$ and one from $\hat{\boldsymbol{v}}$, we have

$$P = \sum_{i,j} M_{i,j} \hat{u}_i \hat{v}_j = \hat{\boldsymbol{u}}^T \cdot \boldsymbol{M} \cdot \hat{\boldsymbol{v}}.$$

Observe that the formal variables $\hat{r}_i\hat{r}_j$ only exists in $M_{i,j}\hat{u}_i\hat{v}_j$ and the coefficient is $M_{i,j}u_i v_j$, we can get $M_{i,j} = 0$ if $i \neq j$ since $u_i v_j \neq 0$ following the assumption. Now we rewrite

$$\begin{aligned} P &= \sum_i M_{i,i} \hat{u}_i \hat{v}_i \\ &= \sum_i M_{i,i} \left(\sum_{j=1}^{i} (-1)^{i-j} u_j \prod_{k=j}^{i-1} \hat{z}_k \right) (v_i + v_{i+1}\hat{z}_i). \end{aligned}$$

Observe that the monomial $c\hat{z}_i$ for some constant c exists in two cases, one in $M_{i,i}\hat{u}_i\hat{v}_i$ and another in $M_{i+1,i+1}\hat{u}_{i+1}\hat{v}_{i+1}$. The coefficients are $M_{i,i}u_i v_{i+1}$ and $-M_{i+1,i+1}u_i v_{i+1}$, respectively. Again, $M_{i,i}$ must be equal to $M_{i+1,i+1}$ since $u_i v_{i+1} \neq 0$ following the assumption. Therefore, the matrix $\boldsymbol{M}$ can be viewed as the form $\boldsymbol{M} = m\boldsymbol{I}$ for some constant m and $P = m\langle \hat{\boldsymbol{u}}, \hat{\boldsymbol{v}} \rangle$. This completes the proof. □