Carnegie Mellon University

CARNEGIE INSTITUTE OF TECHNOLOGY

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

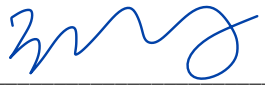
FOR THE DEGREE OF Master of Science

TITLE Multi-Input Inner Product Encryption: Function-hiding instantiations without Random Oracles

PRESENTED BY Nikhil Vanjani

ACCEPTED BY THE DEPARTMENT OF

Information Networking Institute



THESIS ADVISOR — DATE 12/7/21

ACADEMIC ADVISOR — DATE

DEPARTMENT HEAD — DATE

APPROVED BY THE COLLEGE COUNCIL

DEAN — DATE

Multi-Input Inner Product Encryption: Function-hiding instantiations without Random Oracles

Submitted in partial fulfillment of the requirements for

the degree of

Master of Science

in

Information Security

Nikhil Vanjani

B. Tech., Computer Science and Engineering
Indian Institute of Technology Kanpur, India

Carnegie Mellon University
Pittsburgh, PA

December, 2021

Acknowledgements

I would like to thank my committee members:

- Elaine - for being an inspiring and supportive mentor and for helping me understand subtle yet crucial aspects of cryptography.
- Ke - for helping me kickstart with the research done during this thesis.

I am thankful to the Information Networking Institute at Carnegie Mellon University for supporting this research in part through a tuition scholarship.

I would also like to thank my family and friends for their support throughout this journey.

Abstract

In a Multi-Input Functional Encryption (MIFE) scheme, n clients each obtain a secret encryption key from a trusted authority. Each client i can encrypt its data using its secret key. The authority can use its master secret key to compute a functional key given a function f, and the functional key can be applied to a collection of n clients' ciphertexts, resulting in the outcome of f on the clients' data. If an MIFE scheme hides not only the clients' data but also the function f, we say it is function hiding. In this work, we study function-hiding security of two variants of MIFE for inner-product computations.

Multi-Client Functional Encryption (MCFE) is a strengthening of MIFE where clients associate their encrypted data with some time step t and the outcome of f can be computed only on ciphertexts encrypted to the same time step. Although MCFE for inner-product computation has been extensively studied, most earlier works on MCFE do not achieve function privacy. The recent work by Agrawal et al. showed how to construct a function-hiding MCFE for inner-product from standard assumptions and the existence of a random oracle. An intriguing open question is whether we can achieve the same without random oracles. In this work, we are the first to show such a function-hiding MCFE for inner products, relying on the standard Decisional Linear assumption. Our main technical contribution is a new upgrade from single-input functional encryption for inner-products to a multi-client one; and, if the single-input scheme is function-hiding, so is the resulting multi-client scheme.

Ad Hoc MIFE (AMIFE) is a decentralized version of MIFE. In AMIFE, the users can jointly decide in a decentralized way what function they would allow to be evaluated on their joint data. The aforementioned work by Agrawal et al. also showed how to construct a function-hiding AMIFE scheme for inner-products, relying on standard bilinear group assumptions, and without random oracles. We construct a new AMIFE scheme that provides the same security guarantees as this earlier work but our construction provides a nicer abstraction making the scheme and the security proofs conceptually simpler.

Table of Contents

List of Tables

1

Introduction

Multi-Input Functional Encryption (MIFE), first proposed by Goldwasser et al. [GGG+14, GGJS13, GKL+13], allows us to evaluate certain functions on multiple users' encrypted data. In MIFE, a trusted setup gives an encryption key to each of n users, and then each user i can use its encryption key to encrypt some value x_i. A data analyst can ask the trusted setup for a cryptographic token to evaluate a specific function f. Equipped with the token, the data analyst can evaluate the outcome $f(x_1, \ldots, x_n)$ when presented with n ciphertexts each encoding $x_1, \ldots, x_n$, respectively.

It is also well-understood that the MIFE formulation suffers from a couple limitations. The first notable limitation is that it does not make any attempt to limit the mix-and-match of ciphertexts. The evaluator can take any combination of users' ciphertexts, one from each user, to evaluate the function f. As a simple example, imagine that two users each encrypted two values, x_0, x_1 and y_0, y_1, respectively. Then, the evaluator can learn the outcome of $f(x_{b_0}, y_{b_1})$ for any combination of $b_0, b_1 \in \{0, 1\}$. In some applications, this may be too much leakage, and we want to limit the extent of mix-and-match. As a result, a related notion called Multi-Client Functional Encryption (MCFE) was introduced [SCR+11, GGG+14].

An MCFE scheme works well in a "streaming" setting: imagine that in every time step t, each user i encrypts a value $x_{i,t}$. Given the ciphertexts, the evaluator can evaluate

$f(x_{1,t}, \ldots, x_{n,t})$ for each time step t, but it cannot mix-and-match the ciphertexts across different time steps and combine them in the evaluation. This greatly restricts the inherent leakage of the scheme. More generally, MCFE schemes allow users to encrypt to a label t, and only ciphertexts encrypted to the same label t can be used together during the functional evaluation.

Another limitation of MIFE is that it encourages centralization as a trusted third party is needed for setup phase, and the same trusted party can decide what function to compute on the users' data. Therefore, to overcome these limitations an elegant new notion called Ad Hoc MIFE (AMIFE) was introduced [ACF+20].

In AMIFE, every user locally runs the setup algorithm and posts their public keys to a public bulletin board. Later, the users can jointly decide what function they would allow to be evaluated on their joint data. To authorize a function f to be evaluated, each user creates a piece of a token for the function f, and posts it to a public bulletin board. When we collect all pieces from a set $\mathcal{U}$ of users, we can then evaluate the function f on the inputs from users in $\mathcal{U}$.

In this paper, we explore the following question left open by these lines of work.

> Can we construct a *function-hiding* MCFE and AMIFE scheme for inner product evaluation from *standard* assumptions, *without random oracles*?

An MCFE or AMIFE scheme is said to be function hiding, iff the token additionally hides the function f being evaluated.

Prior results. To put our work in context, it is helpful to first review the prior results in this space. For general functions, we currently do not know how to construct MIFE schemes (including MCFE/AMIFE) without relying on indistinguishability obfuscation. However, for inner-product computation, prior work constructed MCFE from standard assumptions without the function hiding requirement. For example, function-revealing MCFE for inner products can be constructed from either bilinear group assumptions [CDG+18, ABG19] or lattices [LT19]. Function-revealing AMIFE for inner products can also be constructed from

standard assumptions [ACF+20]. When it comes to function hiding, we know how to construct function-hiding MIFE for inner-products from standard bilinear group assumptions, due to the elegant work by Abdalla et al. [ACF+18]. However, for the strengthened abstraction MCFE, a function-hiding construction is unknown. Unfortunately, the function-hiding techniques of [ACF+18] are not compatible with MCFE or AMIFE — we explain the technical difficulties that arise in more detail in Section 1.2, and perhaps for these reasons, how to construct function-hiding MCFE from standard assumptions still eludes us. It is worth noting that if we are allowed to assume the existence of random oracles, then the recent work of Agrawal et al. [AGT21b] presents a solution. In fact, they construct a scheme called a Dynamic Decentralized Functional Encryption (DDFE) scheme for inner products assuming bilinear groups and the existence of random oracles. In particular, the DDFE abstraction unifies the benefits of MCFE and AMIFE; and therefore their result directly gives rise to a corresponding MCFE/AMIFE scheme for inner-product too. Moreover, the AMIFE scheme that arises as a consequence does not need random oracles.

1.1 Our Results and Contributions

We advance the state of our understanding regarding function-hiding instantiations of MCFE, AMIFE for inner products. For MCFE, we provide the first function-hiding construction from standard bilinear group assumptions. For AMIFE, we provide a conceptually simpler function-hiding construction from standard bilinear group assumptions.

Specifically, we prove the following theorems.

Theorem 1.1.1 (Function-hiding MCFE for inner-products without random oracles)**.** *Assuming the Decisional Linear assumption in bilinear groups, there exists a function-hiding MCFE scheme for inner-products that satisfies a standard selective, indistinguishability-based security notion (same as the security notion in prior works [AGT21b, SW21] and formally defined in Section 3.1).*

Theorem 1.1.2 (Function-hiding AMIFE for inner-products without random oracles)**.** *As-*

suming the Decisional Linear and Decisional Bilinear Diffie-Hellman assumptions in suitable bilinear groups, there exists a function-hiding AMIFE scheme for inner-products that satisfies a standard selective, indistinguishability-based security notion (same as the security notion in prior works [AGT21b] and formally defined in Section 3.2).

Additional contributions. Notably, both our MCFE and AMIFE constructions are conceptually simpler than some prior constructions (even when compared with some prior schemes without function-hiding). Our schemes are also asymptotically efficient and potentially implementable. The techniques we use to get our results may also be of independent interest as we explain below.

- **A new upgrade from single-input to multi-client.** To get our MCFE result, we suggest a new technique for upgrading a *single-input* function-hiding inner-product functional encryption scheme (satisfying certain properties) to a *multi-client* one. Our upgrade is new and improves the state of the art in the following senses. Abdalla et al. [ABG19] showed how to upgrade a single-input inner-product functional encryption scheme satisfying certain properties (henceforth referred to as IPE) to a multi-client one. Their upgrade does not require random oracles, but it suffers from a couple drawbacks: 1) even if the original IPE scheme is function-private, their upgrade does not preserve function privacy; and 2) their upgrade incurs a $\Theta(n)$ blowup in the per-client ciphertext size. In comparison, our work preserves the function privacy property if the original IPE is function private; moreover, we incur only constant blowup in the per-client ciphertext size. The recent construction of Agrawal et al. [AGT21b] can also be viewed as an upgrade from a function-hiding IPE to a function-hiding MCIPE scheme — however, as mentioned, their construction critically relies on a random oracle.
- **A new upgrade from single-input to ad-hoc multi-input.** To get our AMIFE result, we propose a new upgrade from a *single-input* function-hiding inner-product functional encryption scheme , to an ad-hoc, multi-input one. In comparison, the prior work of [ACF+20] suggests an approach to compile an MIFE scheme into an AMIFE scheme.

Their transformation works for general functionalities, but it suffers from the following drawbacks: 1) their approach is inherently not function-hiding; 2) their scheme is not concretely efficient and incurs a large $\mathsf{poly}(\lambda)$ blowup where λ is the security parameter, since their scheme involves evaluating an MIFE scheme under a 2-round multi-party computation protocol; and 3) their scheme does not support evaluation among a *dynamic* set of users, i.e., all users must participate to evaluate any function. In contrast, our approach preserves function privacy, is concretely efficient as it does not involve a multi-party computation protocol and enables a dynamic set of users to evaluate some inner-product function over their joint plaintexts.

1.2 Additional Related Work

Multi-input functional encryption. Multi-input functional encryption (MIFE) was first proposed by Goldwasser et al. [GGG+14], who constructed an MIFE scheme for general functions assuming indistinguishability obfuscation and other standard cryptographic assumptions. A line of subsequent works [AGRW17, ACF+18, AGT21a] explored how to get MIFE for inner products and quadratic functions from standard assumptions, without resorting to program obfuscation. Notably, the work of Abdalla et al. [ACF+18] gave a function-hiding MCFE for inner products from standard bilinear group assumptions. Unfortunately, *their function privacy techniques do not readily extend to MCFE or AMIFE* — as mentioned, MCFE and AMIFE both impose more stringent syntactic and/or security requirements than MIFE. More specifically, their function privacy techniques for MIFE fail in the context of AMIFE, because they require that all key components share the same random coins — this does not work when the key components must be generated by the parties in a distributed fashion without coordination. Their function privacy techniques fail for the MCFE context, not only because MCFE restricts the mix-and-match for ciphertexts with different labels, but also because the presence of corrupt clients in MCFE complicates the function-hiding construction and proof. In particular, when a functional key is released to the adversary, the key components corresponding to corrupt players disclose some information about the

shared secret randomness used in the functional key computation (across all coordinates). This technicality creates significant challenges in the construction and proofs.

Multi-client functional encryption. Multi-Client Functional Encryption (MCFE) was first suggested by Shi et al. [SCR+11] (referred to as "private stream aggregation" in their work), but they considered only the simple *summation* function (and thus there is only one functional key globally). Goldwasser et al. [GGG+14, GKL+13, GGJS13] generalized MCFE to support *arbitrary* polynomial-time functions, and suggested a construction relying on Indistinguishability Obfuscation (iO) [GGH+13] the existence of a random oracle, and other standard assumptions; and their scheme is *not* function hiding.

Several prior works [ABG19, ABM+20, CDG+18, LT19, AGT21b] considered MCFE for inner products. Among them, the only work that achieved function privacy is the work of Agrawal et al. [AGT21b], but their scheme relies on a random oracle. All earlier schemes do not achieve function privacy; moreover, a subset of these results [ABM+20, AGT21b, CDG+18] needed a random oracle.

Very recently, the work of Shi et al. [SW21] consider a simple special case of inner-product, that is, "selection". Selection is the task of selecting one coordinate from the plaintext vector, i.e., inner product with a special vector where one coordinate is set to 1, and all other coordinates are set to 0. Shi et al. showed how to achieve function privacy in an MCFE scheme for the special selection operator, and moreover, they used the resulting scheme to construct a non-interactive anonymous router. Their function privacy technique does not readily extend to inner products. In particular, their proof techniques are tightly coupled with the fact that the key vector is a selection vector.

Ad-hoc multi-input functional encryption. Ad hoc multi-input functional encryption (AMIFE) was first proposed by Agrawal et al. [ACF+20]. They proposed a paradigm to construct AMIFE for inner products (aka AMIIPE) from certain "ad hoc friendly" MIFE schemes. To do so, they also used 2-round MPC as an ingriedient. All of their building blocks are known

from standard assumption and hence is their result. The MPC-based paradigm of Agrawal et al. [ACF+20] inherently needs the function description in plain for the MPC protocol and thus there is no straightforward way to achieve function-hiding security for constructions based on such paradigm. Subsequently, Chotard et al. [CDSG+20] generalized the notion of AMIFE to dynamic decentralized functional encryption (DDFE) which can be seen AMIFE but in the multi-client setting. Agrawal et al. [AGT21b] constructed function-hiding DDFE for inner-products assuming random oralces. In their construction, random oracles are needed for the labels arising due to the multi-client setting. But when their construction is considered in the restricted multi-input setting (equivalently, AMIFE), random oracles are not needed anymore. While our function-hiding AMIFE construction matches the security guarantees and assumptions with this construction, it combines insights from prior works [ACF+20, CDSG+20] to provide a nicer abstraction, thus making the scheme and security proofs conceptually simpler.

2

Overview of Our Constructions and Techniques

We now give an informal overview of our construction and proof techniques. In our subsequent technical sections, we will present formal definitions, detailed scheme description, and formal proofs.

Notations. Throughout, we will use boldface letters such as $\mathbf{x}$ to denote vectors. Given a bilinear group $\mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ of prime order q, we use the notation $[\![x]\!]$ and $[\![x]\!]_T$ to denote the group encoding of $x \in \mathbb{Z}_q$; and a similar notation is used for vectors too.

2.1 Function-Hiding Multi-Client Inner-Product Encryption

2.1.1 Building Blocks

Our schemes makes use of a single-input, function-hiding functional encryption scheme for computing "inner-product in the exponent", henceforth denoted IPE. We assume that IPE is built from bilinear groups; moreover, we want the following nice property: the encryption algorithm (denoted **Enc**) and the functional key generation algorithm (denoted **KGen**) should work even when taking in the group encoding of the plaintext or key vector rather than the vector itself. Indeed, known constructions of IPE satisfy this nice property [BIK+17, ABG19, SW21] — see Section 4.4 for details.

Additionally, we make use of a *correlated pseudorandom function*, denoted CPRF. In a CPRF scheme, each client $i \in [n]$ obtains a secret key K_i from a trusted setup. Then, given a message x, the user can compute $\mathsf{CPRF.Eval}(K_i, x)$ to obtain an outcome that is computationally indistinguishable from random, subject to the constraint that

$$\sum_{i \in [n]} \mathsf{CPRF.Eval}(K_i, x) = 0 \tag{2.1}$$

Furthermore, even when a subset of the clients may be corrupted, the outcomes of the honest clients' evaluations are nonetheless pseudorandom subject to the constraint in Equation (2.1) — see Section 4.5 for the formal definition. Earlier works have shown how to construct such a CPRF assuming the existence of pseudorandom functions.

2.1.2 Our Construction

We sketch our construction below — a more formal presentation can be found in subsequent technical sections. In our scheme the public parameters are just the public parameters of the underlying IPE scheme.

- **Setup**: during a setup phase, we run n independent instances of $\mathsf{IPE.Setup}$ to sample n secret keys denoted $\mathsf{imsk}_1, \ldots, \mathsf{imsk}_n$, respectively. We also run the setup algorithm of the CPRF, and obtain $K_1, \ldots, K_n$. Finally, we generate a random $a_i \xleftarrow{\$} \mathbb{Z}_q$ for each client $i \in [n]$.

  In summary, each client's secret key is composed of the terms $(\mathsf{imsk}_i, K_i, a_i)$, and the master secret key is simply the union of all clients' secret keys.

- **KGen**: an authority with the master secret key can compute a functional key for the vector $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n) \in \mathbb{Z}_q^{m \cdot n}$ as follows:

$$\{\mathsf{isk}_i := \mathsf{IPE.KGen}(\mathsf{imsk}_i, \widetilde{\mathbf{y}}_i)\}_{i \in [n]} \quad \text{where } \widetilde{\mathbf{y}}_i = (\mathbf{y}_i, 0^m, \rho, -\rho a_i, 0) \text{ for some fresh random } \rho \xleftarrow{\$} \mathbb{Z}_q$$

- **Enc**: for a client $i \in [n]$ to encrypt a vector $\mathbf{x}_i \in \mathbb{Z}_q^m$ to some label t, it samples $\mu_{i,t} \xleftarrow{\$} \mathbb{Z}_q$ if it has not been sampled before, and outputs the following ciphertext:

$$\mathsf{IPE}.\mathbf{Enc}\left(\mathsf{imsk}_i, \widetilde{\mathbf{x}}_i\right) \text{where } \widetilde{\mathbf{x}}_i = (\mathbf{x}_i, 0^m, \mathsf{CPRF}.\mathbf{Eval}(K_i, t) + a_i\mu_{i,t}, \mu_{i,t}, 0)$$

- **Dec**: to decrypt, simply use each isk_i to decrypt the ciphertext ct_i from the i-th client and obtain a partial decryption p_i; then, output the discrete log of $\prod_{i\in[n]} p_i$. Since decryption requires computing discrete logarithm, the outcome of the inner-product computation must lie within a polynomially-bounded space for the decryption to be efficient.

It is not hard to verify correctness. Suppose that $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ are n honestly generated ciphertexts all for the same label t, and for plaintext vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$, respectively. Further, suppose that $(\mathsf{isk}_1, \ldots, \mathsf{isk}_n)$ is the functional key for the vector $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$. Then, applying isk_i to ct_i gives the partial decryption result

$$\begin{aligned} p_i &= [\![\langle \mathbf{x}_i, \mathbf{y}_i \rangle + \rho \cdot \mathsf{CPRF.Eval}(K_i, t) + \rho \cdot a_i\mu_{i,t} - \rho a_i \cdot \mu_{i,t}]\!]_T \\ &= [\![\langle \mathbf{x}_i, \mathbf{y}_i \rangle + \rho \cdot \mathsf{CPRF.Eval}(K_i, t)]\!]_T \end{aligned}$$

Therefore, when we compute the product $\prod_{i\in[n]} p_i$, the part related to the CPRF all cancel out, leaving us the term $[\![\mathbf{x}, \mathbf{y}]\!]_T$ where $\mathbf{x} := (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

2.1.3 Proof Roadmap

Table 2.1: MCFE: Sequence of hybrids, where $\star$ denotes the most technical step to be elaborate later. Here we show the vectors passed to the underlying IPE's **Enc** and **KGen** functions in each hybrid. Q_{kgen} denotes the maximum number of **KGen** queries made by the adversary. For conciseness, we write $\mathsf{CPRF}(K_i, t)$ as a shorthand for $\mathsf{CPRF}.\mathbf{Eval}(K_i, t)$. Note that the ρ term is sampled fresh at random for each **KGen** query.

Hybrid	**Enc**	**KGen**	assumption
Real_1	$\left(\mathbf{x}_i^{(1)}, \mathbf{0}, \mathsf{CPRF}(K_i,t) + a_i\mu_{i,t}, \mu_{i,t}, 0\right)$	$\left(\mathbf{y}_i^{(1)}, \mathbf{0}, \rho, -\rho a_i, 0\right)$	
Hyb_0	$\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, \mathsf{CPRF}(K_i,t) + a_i\mu_{i,t}, \mu_{i,t}, 0\right)$	$\left(\mathbf{y}_i^{(1)}, \mathbf{0}, \rho, -\rho a_i, 0\right)$	FH-IND of IPE
Hyb_ℓ $\ell \in [Q_{\text{kgen}}]$	same as Hyb_0	first ℓ: $\left(\mathbf{0}, \mathbf{y}_i^{(0)}, \rho, -\rho a_i, 0\right)$ remaining: $\left(\mathbf{y}_i^{(1)}, \mathbf{0}, \rho, -\rho a_i, 0\right)$	explained below $\star$
Hyb^*	$\left(\mathbf{0}, \mathbf{x}_i^{(0)}, \mathsf{CPRF}(K_i,t) + a_i\mu_{i,t}, \mu_{i,t}, 0\right)$	$\left(\mathbf{0}, \mathbf{y}_i^{(0)}, \rho, -\rho a_i, 0\right)$	FH-IND of IPE
Real_0	$\left(\mathbf{x}_i^{(0)}, \mathbf{0}, \mathsf{CPRF}(K_i,t) + a_i\mu_{i,t}, \mu_{i,t}, 0\right)$	$\left(\mathbf{y}_i^{(0)}, \mathbf{0}, \rho, -\rho a_i, 0\right)$	FH-IND of IPE

To prove that our scheme satisfies function-hiding indistinguishability-based security, we need to go through a sequence of hybrids as shown in Table Table 2.1. Note that Table Table 2.1 shows only how the challenger generates ciphertext and key components for an *honest* client $i \in [n]$. For a *corrupted* client i, the security game stipulates that $\mathbf{x}_i^{(0)} = \mathbf{x}_i^{(1)}$ and $\mathbf{y}_i^{(0)} = \mathbf{y}_i^{(1)}$, and thus the challenger simply runs the honest **Enc** or **KGen** algorithm as in the real world.

The steps where we apply the function-hiding indistinguishability security (denoted FH-IND in Table Table 2.1) of the underlying IPE are relatively straightforward. The most technical part in the proof is to argue that $\mathsf{Hyb}_{\ell-1}$ is computationally indistinguishable from Hyb_ℓ for $\ell \in [Q_{\text{kgen}}]$. To do this, we carry out yet another sequence of inner hybrids as shown in Table Table 2.2. In this sequence of inner hybrids, the third step that relies on Decisional Linear is the most technical one, and is proven in detail in Claim 5.2.2.

Table 2.2: MCFE: Inner hybrids to go from $\mathsf{Hyb}_{\ell-1}$ to Hyb_ℓ. The most technical steps are the ones that rely on Decisional Linear (DLin), formally proven later in Claim 5.2.2. ρ^* is the randomness used in the ℓ-th **KGen** query, and $\mathbf{y}^{*(b)} := (\mathbf{y}_1^{*(b)}, \ldots, \mathbf{y}_n^{*(b)})$ for $b \in \{0,1\}$ denote the key vectors submitted in the ℓ-th **KGen** query.

Hybrid		assumption
$\mathsf{Hyb}_{\ell-1}$	see Table Table 2.1	
$\mathsf{H}_{\ell-1,1}$	**Enc** : $\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, \mathsf{CPRF}(K_i,t) + a_i\mu_{i,t}, \mu_{i,t}, \mathsf{CPRF}(K_i,t)\cdot\rho^* + \langle \mathbf{x}_i^{(1)}, \mathbf{y}_i^{*(1)}\rangle\right)$ **KGen** : first $\ell-1$: $\left(0^m, \mathbf{y}_i^{(0)}, \rho, -\rho a_i, 0\right)$ ℓ-th: $(0^m, 0^m, 0, 0, 1)$ remaining: $\left(\mathbf{y}_i^{(1)}, 0^m, \rho, -\rho a_i, 0\right)$	FH-IND of IPE
$\mathsf{H}_{\ell-1,2}$	**Enc** : $\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, R_{i,t} + a_i\mu_{i,t}, \mu_{i,t}, R_{i,t}\cdot\rho^* + \langle \mathbf{x}_i^{(1)}, \mathbf{y}_i^{*(1)}\rangle\right)$ where $\sum_{i\in\mathcal{H}} R_{i,t} = -\sum_{i\in\mathcal{K}} \mathsf{CPRF}(K_i,t)$ **KGen** : same as $\mathsf{H}_{\ell-1,1}$	correlated pseudorand. of CPRF
$\mathsf{H}_{\ell-1,3}$	**Enc** : $\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, R_{i,t} + a_i\mu_{i,t}, \mu_{i,t}, T_{i,t} + \langle \mathbf{x}_i^{(1)}, \mathbf{y}_i^{*(1)}\rangle\right)$ where $\sum_{i\in\mathcal{H}} T_{i,t} = -\rho^* \cdot \sum_{i\in\mathcal{K}} \mathsf{CPRF}(K_i,t)$ **KGen** : same as $\mathsf{H}_{\ell-1,1}$	DLin
$\mathsf{H}'_{\ell-1,3}$	**Enc** : $\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, R_{i,t} + a_i\mu_{i,t}, \mu_{i,t}, T_{i,t} + \langle \mathbf{x}_i^{(0)}, \mathbf{y}_i^{*(0)}\rangle\right)$ where $\sum_{i\in\mathcal{H}} T_{i,t} = -\rho^* \cdot \sum_{i\in\mathcal{K}} \mathsf{CPRF}(K_i,t)$ **KGen** : same as $\mathsf{H}_{\ell-1,1}$	identically distributed
$\mathsf{H}'_{\ell-1,2}$	**Enc** : $\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, R_{i,t} + a_i\mu_{i,t}, \mu_{i,t}, R_{i,t}\cdot\rho^* + \langle \mathbf{x}_i^{(0)}, \mathbf{y}_i^{*(0)}\rangle\right)$ where $\sum_{i\in\mathcal{H}} R_{i,t} = -\sum_{i\in\mathcal{K}} \mathsf{CPRF}(K_i,t)$ **KGen** : same as $\mathsf{H}_{\ell-1,1}$	DLin
$\mathsf{H}'_{\ell-1,1}$	**Enc** : $\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, \mathsf{CPRF}(K_i,t) + a_i\mu_{i,t}, \mu_{i,t}, \mathsf{CPRF}(K_i,t)\cdot\rho^* + \langle \mathbf{x}_i^{(0)}, \mathbf{y}_i^{*(0)}\rangle\right)$ **KGen** : same as $\mathsf{H}_{\ell-1,1}$	correlated pseudorand. of CPRF
Hyb_ℓ	see Table Table 2.1	FH-IND of IPE

2.2 Function-Hiding Ad Hoc Multi-Iput Inner Product Encryption

2.2.1 Building Blocks

Our scheme uses a *decentralized secure summation* scheme, denoted DSum. In a DSum scheme, each client $i \in [n]$ generates their own public key - secret key pair $(\mathsf{dpk}_i, \mathsf{dsk}_i)$ as part of local setup. Then, a set $\mathcal{U}$ of clients agree to compute a sum of their inputs. To achieve this, each client $i \in \mathcal{U}$ encrypts a message x_i associated with the set $\mathcal{U}$ and a label t using their secret key and everyone else's public key as $\mathsf{ct}_i \leftarrow \mathbf{Enc}(\mathsf{dsk}_i, (x_i, \mathcal{U}, t), \{\mathsf{dpk}_j\}_{j \in \mathcal{U}})$. They publish their ciphertexts to a public bullentin board and if everyone's ciphertexts for the unique tuple $(\mathcal{U}, t)$ are available on the bulletin board, then, anyone can recover the sum of the underlying messages as $\sum_i x_i = \mathbf{Dec}(\mathsf{dpp}, \{\mathsf{ct}_i\}_{i \in \mathcal{U}})$.

2.2.2 Our Construction

We sketch our construction below — a more formal presentation can be found in subsequent technical sections. In our scheme the public parameters pp are just the public parameters of the underlying IPE scheme and DSum scheme.

- **USetup**: during a user i's setup phase, the user runs $\mathsf{IPE}.\mathbf{Setup}$ to sample a secret keys denoted imsk_i. It also runs the user setup algorithm of the DSum, and obtain $(\mathsf{dpk}_i, \mathsf{dsk}_i)$. In summary, each client's master secret key is composed of the terms $(\mathsf{imsk}_i, \mathsf{dsk}_i)$, and public key is $(\mathsf{pp}, \mathsf{dpk}_i)$.

- **KGen**: A user i can issue a functional key for the vector $\mathbf{y}_i \in \mathbb{Z}_q^m$ associated with a user set $\mathcal{U}$ and a label t as follows:

$$(\mathsf{isk}_i := \mathsf{IPE}.\mathbf{KGen}(\mathsf{imsk}_i, \widetilde{\mathbf{y}}_i), \mathsf{dct}_i := \mathsf{DSum}.\mathbf{Enc}(\mathsf{dsk}_i, ([\![r_i]\!], \mathcal{U}, t), \{\mathsf{dpk}_i\}_{i \in \mathcal{U}}))$$
$$\text{where } \widetilde{\mathbf{y}}_i = (\mathbf{y}_i, 0^m, r_i, 0) \text{ and } r_i \xleftarrow{\$} \mathbb{Z}_q \text{ is fresh randomness}$$

- **Enc**: for a client i to encrypt a vector $\mathbf{x}_i \in \mathbb{Z}_q^m$, it outputs the following ciphertext:

$$\mathsf{IPE}.\mathbf{Enc}\left(\mathsf{imsk}_i, \widetilde{\mathbf{x}}_i\right), \text{ where } \widetilde{\mathbf{x}}_i = (\mathbf{x}_i, 0^m, 1, 0)$$

- **Dec**: to decrypt, simply use each isk_i to decrypt the ciphertext ct_i from the i-th client and obtain a partial decryption p_i; then, use DSum decryption to obtain $[\![z]\!] = [\![\sum_i r_i]\!]$. Finally, compute $[\![v]\!]_T = \prod_{i\in[n]} p_i$ and output discrete log of $([\![v]\!]_T/[\![z]\!]_T)$. Since decryption requires computing discrete logarithm, the outcome of the inner-product computation must lie within a polynomially-bounded space for the decryption to be efficient.

To verify correctness, suppose that for a set $\mathcal{U}$ of users, $\{\mathsf{ct}_i\}_{i\in\mathcal{U}}$ are honestly generated ciphertexts for plaintext vectors $\{\mathbf{x}_i\}_{i\in\mathcal{U}}$, respectively. Further, suppose that $\{\mathsf{isk}_i, \mathsf{dct}_i\}_{i\in\mathcal{U}}$ are the functional keys for the vectors $\{\mathbf{y}_i\}_{i\in\mathcal{U}}$, all generated for the same label t and set $\mathcal{U}$. Then, applying isk_i to ct_i gives the partial decryption result $p_i = [\![\langle \mathbf{x}_i, \mathbf{y}_i\rangle + r_i]\!]_T$ and combining all the DSum ciphertexts dct_i gives the mask $[\![z]\!] = [\![\sum_{i\in\mathcal{U}} r_i]\!]$. Therefore, when we compute the product $[\![v]\!]_T = \prod_{i\in\mathcal{U}} p_i$, we get $[\![v]\!]_T = [\![\sum_{i\in\mathcal{U}}\langle \mathbf{x}_i, \mathbf{y}_i\rangle + \sum_{i\in\mathcal{U}} r_i]\!]_T$. Hence, taking discrete log of $([\![v]\!]_T/[\![z]\!]_T)$ gives us the expected result.

Our construction borrows several ideas from existing works. We start with the observation made in [ACF+20] that certain MIFE schemes are ad hoc friendly such as [AGRW17, ACF+18]. In particular, the **Setup** algorithm of both of those MIFE schemes can be trivially made to be run locally by each user. Their **Enc** algorithms are already local. Their **KGen** algorithms are not local and require non-trivial efforts to be made so and that's what we try to achieve. [ACF+20] used non-interactive multi-party computation to execute the MIFE **KGen** algorithm in a decentralized way. But, this approach inherently can't be made function-hiding secure, hence, we divert from this idea.

We take a step back and observe that in the MIFE schemes constructed by Abdalla et al. [AGRW17, ACF+18], the **KGen** algorithm outputs two components, one of which is a set of keys generated by running every user's IPE **KGen** algorithms and the other component captures a way to unmask the final output obtained by running IPE decryption for all parties as part of MIFE decryption algorithm. We note that the first components can be computed by every user locally and it is the second component whose computation needs efforts to

be made local. In the IND-secure MIFE scheme constructed by [ACF+18], this second component comprises of $\sum_i \langle \mathbf{u}_i, \mathbf{y}_i \rangle$ and it can't be made function-hiding because it leaks information about function inputs $\mathbf{y}_i$'s as pointed out by the authors. On the other hand, in the IND-secure MIFE scheme constructed by [AGRW17], this second component comprises of $\sum_i \langle \mathbf{z}_i, \mathbf{r} \rangle$, where $\mathbf{z}_i$ are part of user secret keys and $\mathbf{r}$ is shared randomness. Thus, this is not function-dependent and all we need to do is to decentralize this step. For starters, there is no way to generate the shared randomness $\mathbf{r}$ in a non-interactive way. If we overlook the idea of shared randomness and rather have each user generate randomness $\mathbf{r}_i$ locally, then, this breaks the security proof. In particular, now every user's partial inner products are revealed. Ideally, we would like a way where every user's component masks $\langle \mathbf{z}_i, \mathbf{r}_i \rangle$ are encrypted and only the sum of the masks is revealed to the decrypter. DSum protocol provides exactly this functionality and we employ it make the security proof work. Hence, in summary we follow the blueprint of [AGRW17]'s MIFE scheme and make it ad hoc by using function-hiding IPE and DSum primitives in a black box manner.

We emphasize that our construction is conceptually simpler. In [AGRW17], the mask was of the form $\langle \mathbf{z}_i, \mathbf{r} \rangle$ instead of just $\mathbf{r}$ because $\mathbf{r}$ is given out as part of the functional key in plain and hence it can't be used as a mask. We, on the other hand use a function-hiding IPE as a starting point, so $\mathbf{r}$ is not revealed in plain and hence can directly be used for masking. Further, towards our goal of making all the computation local, instead of computing shared randomness r, every user will generate their own scalar random value r_i and this is sufficient for our security proof.

2.2.3 Proof Roadmap

Table 2.3: AMIFE: Sequence of hybrids, where $\star$ denotes the most technical step to be elaborate later. Here we show the vectors passed to the underlying IPE's **Enc** and **KGen** functions as well as d_i's passed to the underlying DSum's **Enc** in each hybrid. Q_{kgen} denotes the maximum number of **KGen** queries made by the adversary. Note that the r_i terms are sampled fresh at random for each **KGen** query.

	IPE.**Enc**	IPE.**KGen**	DSum.**Enc**				
Hybrid	$\widetilde{\mathbf{x}}_i$	$\widetilde{\mathbf{y}}_i$	d_1	d_2	$\dots$	d_n	assumption
Real_1	$\left(\mathbf{x}_i^{(1)}, \mathbf{0}, 1, 0\right)$	$\left(\mathbf{y}_i^{(1)}, \mathbf{0}, r_i, 0\right)$	r_1	r_2	$\dots$	r_n	
Hyb^*	$\left(\mathbf{x}_i^{(1)}, \mathbf{0}, 1, 0\right)$	$\left(\mathbf{y}_i^{(1)}, \mathbf{0}, r_i, 0\right)$	$\sum_i r_i$	0	$\dots$	0	SEL-IND of DSum
Hyb_0	$\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, 1, 0\right)$	$\left(\mathbf{y}_i^{(1)}, \mathbf{0}, r_i, 0\right)$	$\sum_i r_i$	0	$\dots$	0	FH-IND of IPE
Hyb_ℓ $\ell \in [Q_{\text{kgen}}]$	same as Hyb_0	first ℓ: $\left(\mathbf{0}, \mathbf{y}_i^{(0)}, r_i, 0\right)$ remaining: $\left(\mathbf{y}_i^{(1)}, \mathbf{0}, r_i, 0\right)$	$\sum_i r_i$	0	$\dots$	0	explained below $\star$
$\mathsf{Hyb}^\#$	$\left(\mathbf{0}, \mathbf{x}_i^{(0)}, 1, 0\right)$	$\left(\mathbf{0}, \mathbf{y}_i^{(0)}, r_i, 0\right)$	$\sum_i r_i$	0	$\dots$	0	FH-IND of IPE
Hyb^{**}	$\left(\mathbf{0}, \mathbf{x}_i^{(0)}, 1, 0\right)$	$\left(\mathbf{0}, \mathbf{y}_i^{(0)}, r_i, 0\right)$	r_1	r_2	$\dots$	r_n	SEL-IND of DSum
Real_0	$\left(\mathbf{x}_i^{(0)}, \mathbf{0}, 1, 0\right)$	$\left(\mathbf{y}_i^{(0)}, \mathbf{0}, r_i, 0\right)$	r_1	r_2	$\dots$	r_n	FH-IND of IPE

To prove that our scheme satisfies function-hiding indistinguishability-based security, we need to go through a sequence of hybrids as shown in Table Table 2.3. Interestingly, the security hybrids blueprint is very similar to that for MCFE. Note that Table Table 2.3 shows only how the challenger generates ciphertext and key components for an *honest* client $i \in [n]$. For a *corrupted* client i, the security game stipulates that $\mathbf{x}_i^{(0)} = \mathbf{x}_i^{(1)}$ and $\mathbf{y}_i^{(0)} = \mathbf{y}_i^{(1)}$, and thus the challenger simply runs the honest **Enc** or **KGen** algorithm as in the real world.

The steps where we apply the function-hiding indistinguishability security (denoted FH-IND in Table Table 2.3) of the underlying IPE and the indistinguisability security of the underlying DSum are relatively straightforward. The most technical part in the proof is to

Table 2.4: AMIFE: Inner hybrids to go from $\mathsf{Hyb}_{\ell-1}$ to Hyb_{ℓ}. r_i^* is the randomness used in the ℓ-th **KGen** query, and $\mathbf{y}_i^{*(b)}$ for $b \in \{0,1\}$ and for $i \in \mathcal{U}$ denote the key vectors submitted in the ℓ-th **KGen** query.

Hybrid		assumption
$\mathsf{Hyb}_{\ell-1}$	see Table Table 2.3	
$\mathsf{H}_{\ell-1,1}$	**Enc** : $\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, 1, r_i^* + \langle \mathbf{x}_i^{(1)}, \mathbf{y}_i^{*(1)} \rangle\right)$ **KGen** : first $\ell-1$: $\left(0^m, \mathbf{y}_i^{(0)}, r_i, 0\right)$ ℓ-th: $(0^m, 0^m, 0, 1)$ remaining: $\left(\mathbf{y}_i^{(1)}, 0^m, r_i, 0\right)$	FH-IND of IPE
$\mathsf{H}'_{\ell-1,1}$	**Enc** : $\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, 1, r_i^* + \langle \mathbf{x}_i^{(0)}, \mathbf{y}_i^{*(0)} \rangle\right)$ **KGen** : same as $\mathsf{H}_{\ell-1,1}$	independent and identically distributed
Hyb_{ℓ}	see Table Table 2.1	FH-IND of IPE

argue that $\mathsf{Hyb}_{\ell-1}$ is computationally indistinguishable from Hyb_{ℓ} for $\ell \in [Q_{\text{kgen}}]$. To do this, we carry out yet another sequence of inner hybrids as shown in Table Table 2.4.

3

Definitions

3.1 Multi-Client Inner Product Encryption

Henceforth, we use m to denote the number of coorindates encrypted by each client, and use n to denote the number of clients. In a Multi-Client Inner-Product Functional Encryption (MCIPE) scheme, in every time step, each client $i \in [n]$ encrypts a vector $\mathbf{x}_i \in \mathbb{Z}_q^m$ using its private key ek_i. An authority holding a master secret key msk can generate a functional key $\mathsf{sk}_{\mathbf{y}}$ for a vector $\mathbf{y} \in \mathbb{Z}_q^{mn} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n)$ where each $\mathbf{y}_i \in \mathbb{Z}_q^m$. One can now apply the functional key $\mathsf{sk}_{\mathbf{y}}$ to the collection of all n clients' ciphertexts belonging to the same time step, and an evaluation procedure gives the result $\langle \mathbf{x}, \mathbf{y} \rangle$ where $\mathbf{x} := (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

We use a standard notion of selective indistinguishability for multi-client inner-product encryption [AGT21b]. In this standard definition, the time step t is generalized and encoded as an arbitrary label, and only ciphertexts encrypted to the same label can be combined during the decryption process. Mix-and-match among ciphertexts encrypted to different labels should be prevented; however, mix-and-match among the same label is allowed. Formally, an MCIPE scheme consists of the following possibly randomized algorithms:

- $\mathsf{pp} \leftarrow \mathbf{Gen}(1^\lambda)$: the parameter generation algorithm $\mathbf{Gen}$ takes in a security parameter λ and chooses parameters pp — we will assume that pp contains a λ-bit long prime number

$q \in \mathbb{N}$ and the description of a suitable cyclic group $\mathbb{G}$ of prime order q.

- $(\mathsf{mpk}, \mathsf{msk}, \{\mathsf{ek}_i\}_{i\in[n]}) \leftarrow \mathbf{Setup}(\mathsf{pp}, m, n)$: takes in the parameters q, $\mathbb{G}$, m, and n, and outputs a public key mpk, a master secret key msk, and n user secret keys needed for encryption, denoted $\mathsf{ek}_1, \ldots, \mathsf{ek}_n$, respectively. Without loss of generality, henceforth we may assume that mpk encodes pp so we need not write the parameters pp explicitly below.
- $\mathsf{sk}_{\mathbf{y}} \leftarrow \mathbf{KGen}(\mathsf{mpk}, \mathsf{msk}, \mathbf{y})$: takes in the public key mpk, the master secret key msk, and a vector $\mathbf{y} \in \mathbb{Z}_q^{mn}$, and outputs a functional secret key $\mathsf{sk}_{\mathbf{y}}$.
- $\mathsf{ct}_{i,t} \leftarrow \mathbf{Enc}(\mathsf{mpk}, \mathsf{ek}_i, \mathbf{x}_i, t)$: takes in the public key mpk, a user secret key ek_i, a plaintext $\mathbf{x}_i \in \mathbb{Z}_q^m$, and a label $t \in \{0,1\}^*$, outputs a ciphertext $\mathsf{ct}_{i,t}$.
- $v \leftarrow \mathbf{Dec}(\mathsf{mpk}, \mathsf{sk}_{\mathbf{y}}, \{\mathsf{ct}_{i,t}\}_{i\in[n]})$: takes in the public key mpk, the functional secret key $\mathsf{sk}_{\mathbf{y}}$, and a collection of ciphertexts $\{\mathsf{ct}_{i,t}\}_{i\in[n]}$, outputs a decrypted outcome $v \in \mathbb{Z}_q$.

Correctness. For correctness, we require that for any $\lambda \in \mathbb{N}$, for any $\mathsf{pp} := (q, \ldots)$ in the support of $\mathbf{Gen}(1^\lambda)$, the following holds with probability 1 for any $m, n \in \mathbb{N}$: for any $\mathbf{y} \in \mathbb{Z}_q^{mn}$, and any $\mathbf{x} := (\mathbf{x}_1, \ldots, \mathbf{x}_n) \in \mathbb{Z}_q^n$, and any $t \in \{0,1\}^*$: let $(\mathsf{mpk}, \mathsf{msk}, \{\mathsf{ek}_i\}_{i\in[n]}) \leftarrow \mathbf{Setup}(\mathsf{pp}, m, n)$, let $\mathsf{sk}_{\mathbf{y}} \leftarrow \mathbf{KGen}(\mathsf{mpk}, \mathsf{msk}, \mathbf{y})$, let $\mathsf{ct}_{i,t} \leftarrow \mathbf{Enc}(\mathsf{mpk}, \mathsf{ek}_i, \mathbf{x}_i, t)$ for $i \in [n]$, and let $v \leftarrow \mathbf{Dec}(\mathsf{mpk}, \mathsf{sk}_{\mathbf{y}}, \{\mathsf{ct}_{i,t}\}_{i\in[n]}\})$, it must be that $v = \langle \mathbf{x}, \mathbf{y} \rangle$.

Function-hiding IND-security for MCIPE. We now define function-hiding security. Consider the following experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

Experiment $\mathsf{MCIPE\text{-}Expt}^b(1^\lambda)$:

- **Setup.** $\mathcal{A}(1^\lambda)$ outputs a set of corrupted parties $\mathcal{K} \subset [n]$, as well as the parameters m and n to the challenger $\mathcal{C}$. The challenger $\mathcal{C}$ runs $\mathsf{pp} \leftarrow \mathbf{Gen}(1^\lambda)$, and $(\mathsf{mpk}, \mathsf{msk}, \{\mathsf{ek}_i\}_{i\in[n]}) \leftarrow \mathbf{Setup}(\mathsf{pp}, m, n)$; it gives mpk and $\{\mathsf{ek}_i\}_{i\in\mathcal{K}}$ to $\mathcal{A}$.
- **Query.** The adversary can make the following types of queries:
  - **KGen queries.** Whenever the adversary $\mathcal{A}$ makes a **KGen** query with two vectors $\mathbf{y}^{(0)} \in \mathbb{Z}_q^{mn}$ and $\mathbf{y}^{(1)} \in \mathbb{Z}_q^{mn}$: $\mathcal{C}$ calls $\mathsf{sk}_{\mathbf{y}^{(b)}} := \mathbf{KGen}(\mathsf{mpk}, \mathsf{msk}, \mathbf{y}^{(b)})$ and returns $\mathsf{sk}_{\mathbf{y}^{(b)}}$ to $\mathcal{A}$;

– **Enc queries.** Whenever $\mathcal{A}$ makes an **Enc** query with the tuple $(i, t, \mathbf{x}_{i,t}^{(0)}, \mathbf{x}_{i,t}^{(1)})$, the challenger $\mathcal{C}$ calls $\mathsf{ct}_{i,t} := \mathbf{Enc}(\mathsf{mpk}, \mathsf{ek}_i, \mathbf{x}_{i,t}^{(b)}, t)$ and returns $\mathsf{ct}_{i,t}$ to $\mathcal{A}$;

An adversary $\mathcal{A}$ is said to be *admissible* iff the following hold with probability 1 where $\mathcal{H} := [n] \backslash \mathcal{K}$ denotes the set of honest clients:

1. $\mathcal{A}$ always makes all **KGen** queries ahead of any **Enc** query;
2. for every label $t \in \{0,1\}^*$, either for every $i \in \mathcal{H}$, $\mathcal{A}$ has made at least one **Enc** query of the form $(i, t, \_, \_)$, or $\mathcal{A}$ made no **Enc** query of the form $(i, t, \_, \_)$ for any $i \in \mathcal{H}$.
3. if $\mathcal{A}$ ever makes an **Enc** query with the tuple $(i, t, \mathbf{x}_{i,t}^{(0)}, \mathbf{x}_{i,t}^{(1)})$ for some corrupt $i \in \mathcal{K}$, it must be that $\mathbf{x}_{i,t}^{(0)} = \mathbf{x}_{i,t}^{(1)}$;
4. for any pair $(\mathbf{y}^{(0)}, \mathbf{y}^{(1)})$ submitted in a **KGen** query where for $b \in \{0,1\}$, $\mathbf{y}^{(b)} := (\mathbf{y}_1^{(b)}, \ldots, \mathbf{y}_n^{(b)}) \in \{0,1\}^{mn}$, it must be that
   (a) for $i \in \mathcal{K}$, $\mathbf{y}_i^{(0)} = \mathbf{y}_i^{(1)}$.
   (b) for any collection $\{\mathbf{x}_{i,t}^{(0)}, \mathbf{x}_{i,t}^{(1)}\}_{i \in \mathcal{H}}$ pertaining to the same t where each pair $(\mathbf{x}_{i,t}^{(0)}, \mathbf{x}_{i,t}^{(1)})$ for $i \in \mathcal{H}$ has been submitted in an **Enc** query of the form $(i, t, \mathbf{x}_{i,t}^{(0)}, \mathbf{x}_{i,t}^{(1)})$,

$$\left\langle (\mathbf{x}_{i,t}^{(0)})_{i \in \mathcal{H}}, (\mathbf{y}_i^{(0)})_{i \in \mathcal{H}} \right\rangle = \left\langle (\mathbf{x}_{i,t}^{(1)})_{i \in \mathcal{H}}, (\mathbf{y}_i^{(1)})_{i \in \mathcal{H}} \right\rangle \tag{3.1}$$

Definition 1 (Function-hiding IND-security of MCIPE)**.** We say that an MCIPE scheme is selectively function-hiding IND-secure iff for any non-uniform probabilistic polynomial-time (PPT) admissible adversary $\mathcal{A}$, its views in $\mathsf{MCIPE\text{-}Expt}^0(1^\lambda)$ and $\mathsf{MCIPE\text{-}Expt}^1(1^\lambda)$ are computationally indistinguishable.

Note that the above is a selective, static notion, since the adversary must commit to the corrupted set of clients ahead of time, and moreover, it must make all **KGen** queries ahead of **Enc** queries.

Remark 1. Note that the second admissibility rule requires that if one ciphertext is queried for a label t, then all ciphertexts for honest clients must be queried for this label. Jumping

ahead, this rule is necessary later to show that the hybrids $\mathsf{H}_{\ell,3}$ and $\mathsf{H}'_{\ell,3}$ are identically distributed. If one further restricts the adversary to be "strongly selective", i.e., the adversary must submit submit all **Enc** and **KGen** queries in one shot, then the second admissibility rule can be removed using standard techniques by wrapping the ciphertexts with an additional layer of all-or-nothing encryption [CDSG+20]. This technique was shown in the elegant work of Chotard [CDSG+20], but their work is not function hiding. For completeness, we explicitly present this transformation in Appendix B and prove that it works for *function-hiding* MCIPE too.

3.2 Ad Hoc Multi-Input Inner Product Encryption

Ad Hoc Multi-Input Inner Product Encryption. Ad Hoc Multi-Input Inner Product Encryption (aMIIPE) is a decentralized version of Multi-Input Functional Encryption for computing inner products. In aMIIPE, any *dynamically formed* set of users can evaluate the inner product of a key vector with their joint plaintext vector. Therefore, our formulation of aMIIPE is actually more flexible than the original formulation by Agrawal et al. [ACF+20], since in their formulation, all users must participate key components to approve an evaluation.

Without loss of generality, we may assume that $[n]$ denotes the space of user identities where $n \in \mathbb{N}$ and m denotes the number of coordinates encrypted by each user. In an Ad Hoc Multi-Input Inner-Product Encryption (aMIIPE) scheme, a user i encrypts a vector $\mathbf{x}_i \in \{0,1\}^m$ using its master secret key msk_i. At any point of time, a user i can generate a functional key sk_i for a vector $\mathbf{y}_i \in \{0,1\}^m$; the functional key is also tied to a set of users $\mathcal{U}_i$ that contains i, and a label t_i. The user can publish the functional key sk_i to a public bulletin board. Note that sk_i contains the information $\mathcal{U}_i$ and t_i in plain but does not leak information about $\mathbf{y}_i$. If all the users belonging to the set $\mathcal{U}_i$ publish their functional keys for the same label t_i, then, anyone can apply this set of functional keys to any set of ciphertexts ct_i's by the same set of users to obtain the sum of their respective inner products $\sum_{j \in \mathcal{U}_i} \langle \mathbf{x}_j, \mathbf{y}_j \rangle$.

Formally, let $\mathcal{K}$, and $\mathcal{M}$ denote a key space, and a message space, respectively. Each

key object $k = (\mathbf{y}, \mathcal{U}_K, t) \in \mathcal{K}$ consists of a private key component $\mathbf{y} \in \mathbb{Z}_q^m$, a set of users $\mathcal{U}_K \subseteq [n]$, and a label $t \in \mathcal{T}$ where $\mathcal{T}$ denotes a label space. is a vector denoted $\mathbf{x} \in \mathbb{Z}_q^m$, and we use $\mathcal{M}$ to denote the message space. Let $f : \bigcup_{i\in\mathbb{N}}([n] \times \mathcal{K})^i \times \bigcup_{i\in\mathbb{N}}([n] \times \mathcal{M})^i \to \mathbb{Z}_q^m$ be a function. An Ad Hoc Multi-Input Inner Product Encryption (aMIIPE) scheme for f consists of the following, possibly randomized algorithms:

- $\mathsf{pp} \leftarrow \mathbf{Gen}(1^\lambda, n, m)$: takes a security parameter 1^λ, the space size of user identities $n \in \mathbb{N}$, the per-user message and key length $m \in \mathbb{N}$, and outputs the public parameters pp;
- $(\mathsf{pk}_i, \mathsf{msk}_i) \leftarrow \mathbf{USetup}(\mathsf{pp})$: takes in the public parameters pp, and outputs a public key and master secret key pair for a user, denoted pk_i and msk_i, respectively;
- $\mathsf{sk}_i \leftarrow \mathbf{KGen}(\mathsf{msk}_i, k = (\mathbf{y}_i, \mathcal{U}_i, t_i), \{\mathsf{pk}_j\}_{j\in\mathcal{U}_i})$: takes a user's master secret key msk_i, a key object $k = (\mathbf{y}_i, \mathcal{U}_i, t_i) \in \mathcal{K}$, a set of public keys $\{\mathsf{pk}_j\}_{j\in\mathcal{U}_i}$ for users in $\mathcal{U}_i$, and outputs a secret key sk_i.
- $\mathsf{ct}_i \leftarrow \mathbf{Enc}(\mathsf{msk}_i, \mathbf{x}_i)$: takes in a user's master secret key msk_i, and a message $\mathbf{x}_i \in \mathcal{M}$, and outputs a ciphertext ct_i.
- $v \leftarrow \mathbf{Dec}(\{\mathsf{sk}_i\}_{i\in\mathcal{U}_K}, \{\mathsf{ct}_i\}_{i\in\mathcal{U}_K})$: takes in a set of secret keys $\{\mathsf{sk}_i\}_{i\in\mathcal{U}_K}$, and a set of ciphertexts $\{\mathsf{ct}_i\}_{i\in\mathcal{U}_K}$, and outputs a decrypted value v; if decryption fails, output $v = \perp$.

The aMIIPE function f is defined as follows:

$$f\left(\{i, (\mathbf{y}_i, \mathcal{U}_i, t_i)\}_{i\in\mathcal{U}_K}, \{i, \mathbf{x}_i\}_{i\in\mathcal{U}_K}\right) = \begin{cases} \sum_{i\in\mathcal{U}_K} \langle \mathbf{x}_i, \mathbf{y}_i \rangle & \text{if the condition (*) below is satisfied} \\ \perp & \text{otherwise} \end{cases}$$

Condition (*) is defined as: there exists label $t \in \mathcal{T}$ such that $\forall i \in \mathcal{U}_K : t_i = t$, and moreover, $\mathcal{U}_i = \mathcal{U}_K$.

In other words, fix some $\mathcal{U}$, t: given a collection of ciphertexts from users in $\mathcal{U}$, and given a collection of secret keys from users in $\mathcal{U}$ all tagged with $(\mathcal{U}, t)$, then an evaluator can compute the inner-product $\sum_{i\in\mathcal{U}} \langle \mathbf{x}_i, \mathbf{y}_i \rangle$.

Correctness. We say that an aMIIPE scheme satisfies correctness, iff it satisfies the following condition. For all $\lambda, n \in \mathbb{N}, m \in \mathbb{N}$, for any $\mathcal{U}_K \subseteq [n]$ whose size is denoted $|\mathcal{U}_K| = s$, any $\mathbf{x}_1, \ldots, \mathbf{x}_s$, any $\mathbf{y}_1, \ldots, \mathbf{y}_s$, any $\mathcal{U}_1, \ldots, \mathcal{U}_s$, and any $t_1, \ldots, t_s$ chosen from the appropriate domains, the following holds with probability 1:

$$v = f\left(\{i, (\mathbf{y}_i, \mathcal{U}_i, t_i)\}_{i \in \mathcal{U}_K}, \{i, \mathbf{x}_i\}_{i \in \mathcal{U}_K}\right) : \begin{array}{l} \mathsf{pp} \leftarrow \mathbf{Gen}(1^\lambda, n, m) \\ \forall i \in [n] : (\mathsf{pk}_i, \mathsf{msk}_i) \leftarrow \mathbf{USetup}(\mathsf{pp}) \\ \forall i \in \mathcal{U}_K : \mathsf{ct}_i \leftarrow \mathbf{Enc}(\mathsf{msk}_i, \mathbf{x}_i) \\ \forall i \in \mathcal{U}_K : \mathsf{sk}_i \leftarrow \mathbf{KGen}(\mathsf{msk}_i, (\mathbf{y}_i, \mathcal{U}_i, t_i), \{\mathsf{pk}_j\}_{j \in \mathcal{U}_i}) \\ v = \mathbf{Dec}(\{\mathsf{sk}_i\}_{i \in \mathcal{U}_K}, \{\mathsf{ct}_i\}_{i \in \mathcal{U}_K}) \end{array}$$

Function-hiding IND-security of aMIIPE. Consider the following experiment $\mathsf{aMIIPE\text{-}Expt}^b(1^\lambda)$:

Experiment $\mathsf{aMIIPE\text{-}Expt}^b(1^\lambda)$:

- **Setup.** $\mathcal{A}(1^\lambda)$ outputs a set of parties to corrupt $\mathcal{K} \subset [n]$. The challenger $\mathcal{C}$ runs $\mathsf{pp} \leftarrow \mathbf{Gen}(1^\lambda, n, m)$, and for $i \in [n]$, it runs $(\mathsf{pk}_i, \mathsf{msk}_i) \leftarrow \mathbf{USetup}(\mathsf{pp})$. It returns $\{\mathsf{pk}_i\}_{i \in [n]}$ as well as $\{\mathsf{msk}_i\}_{i \in \mathcal{K}}$ to $\mathcal{A}$.
- **Query.** The adversary can make the following types of queries:
  - **KGen queries.** Whenever $\mathcal{A}$ makes a **KGen** query of the form $(i, \mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \mathcal{U}_K, t)$, the challenger $\mathcal{C}$ calls $\mathsf{sk} \leftarrow \mathbf{KGen}(\mathsf{msk}_i, (\mathbf{y}^{(b)}, \mathcal{U}_K, t), \{\mathsf{pk}_j\}_{j \in \mathcal{U}_K})$, and returns sk to $\mathcal{A}$.
  - **Enc queries.** Whenever $\mathcal{A}$ makes an **Enc** query with the tuple $(i, \mathbf{x}^{(0)}, \mathbf{x}^{(1)})$, $\mathcal{C}$ calls $\mathsf{ct} \leftarrow \mathbf{Enc}(\mathsf{msk}_i, \mathbf{x}^{(b)})$, and returns ct to $\mathcal{A}$;

An adversary $\mathcal{A}$ is said to be *admissible* iff the following hold with probability 1:

- $\mathcal{A}$ always makes all **KGen** queries ahead of any **Enc** query; moreover, all **KGen** queries tagged with the same $(\mathcal{U}_K, t)$ pair must be made all at once;
- if $\mathcal{A}$ ever makes an **Enc** query with the tuple $(i, \mathbf{x}_i^{(0)}, \mathbf{x}_i^{(1)})$ for some corrupt $i \in \mathcal{K}$, it must be that $\mathbf{x}_i^{(0)} = \mathbf{x}_i^{(1)}$;
- if $\mathcal{A}$ ever makes an **KGen** query with the tuple $(i, \mathbf{y}_i^{(0)}, \mathbf{y}_i^{(1)}, \mathcal{U}_K, t)$ for some corrupt $i \in \mathcal{K}$, it must be that $\mathbf{y}_i^{(0)} = \mathbf{y}_i^{(1)}$;
- there are no tuples $\{i, \mathbf{y}_i^{(0)}, \mathbf{y}_i^{(1)}, \mathcal{U}_i, t_i\}_{i \in \mathcal{U}_K}$, and $\{i, \mathbf{x}_i^{(0)}, \mathbf{x}_i^{(1)}\}_{i \in \mathcal{U}_K}$, such that

– for all $i \in \mathcal{U}_K$, the adversary $\mathcal{A}$ has made a **KGen** query of the form $(i, \mathbf{y}_i^{(0)}, \mathbf{y}_i^{(1)}, \mathcal{U}_i, t_i)$,

– for all $i \in \mathcal{U}_K$, the adversary $\mathcal{A}$ has made an **Enc** query of the form $(i, \mathbf{x}_i^{(0)}, \mathbf{x}_i^{(1)})$,

– however, $f(\{i, (\mathbf{y}_i^{(0)}, \mathcal{U}_K, t)\}_{i \in \mathcal{U}_K}, \{i, \mathbf{x}_i^{(0)}\}_{i \in \mathcal{U}_K}) \neq f(\{i, (\mathbf{y}_i^{(1)}, \mathcal{U}_K, t)\}_{i \in \mathcal{U}_K}, \{i, \mathbf{x}_i^{(1)}\}_{i \in \mathcal{U}_K})$.

Definition 2 (Selective function-hiding IND-security of aMIIPE)**.** We say that an aMIIPE scheme satisfies selective, function-hiding IND-security iff for any non-uniform probabilistic polynomial-time (PPT) admissible adversary $\mathcal{A}$, its views in $\mathsf{aMIIPE\text{-}Expt}^0(1^\lambda)$ and $\mathsf{aMIIPE\text{-}Expt}^1(1^\lambda)$ are computationally indistinguishable.

4

Preliminaries

4.1 Bilinear Groups

Throughout, we use λ to denote the security parameter. Boldface letters such as $\mathbf{x}$ denote vectors, and normal-font letters such as x to denote scalars. Given two vectors $\mathbf{x} \in \mathbb{Z}_q^\ell$ and $\mathbf{y} \in \mathbb{Z}_q^\ell$ each of dimension ℓ, we use $\langle \mathbf{x}, \mathbf{y} \rangle \in \mathbb{Z}_q$ to denote their inner-product modulo q.

Notation for group elements. Given a cyclic group $\mathbb{G}$ of prime order q, and a generator $g \in \mathbb{G}$, we use $[\![x]\!]$ to denote $g^x \in \mathbb{G}$ where $x \in \mathbb{Z}_q$. Given a vector $\mathbf{x} := (x_1, x_2, \ldots, x_\ell) \in \mathbb{Z}_q^\ell$, the notation $[\![\mathbf{x}]\!]$ denotes the vector of group elements $(g^{x_1}, g^{x_2}, \ldots, g^{x_\ell})$.

We will employ a bilinear group $(\mathbb{G}, \mathbb{G}_T)$ of prime order q with a pairing operator $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, and a random generator $g \in \mathbb{G}$. In this case, the notation $[\![x]\!]$ means g^x, and the notation $[\![x]\!]_T$ means $e(g, g)^x$. The notations for vectors are similarly defined.

Implicit notation for group operations. If a party knows $[\![x]\!] \in \mathbb{G}$ and $y \in \mathbb{Z}_q$, it is able to efficiently compute $[\![xy]\!] := [\![x]\!]^y$. Therefore, without risk of ambiguity, often when we write "compute $[\![xy]\!]$" in an algorithm description, we mean compute the group exponentiation $[\![x]\!]^y$ or $[\![y]\!]^x$. The same rule also extends to vectors as well as bilinear groups.

4.2 The Decisional Linear Assumption

The Decisional Linear assumption. We say that the Decisional Linear assumption holds for the group generator $\mathcal{G}$, iff the following two experiments are computationally indistinguishable:

1. Sample $\mathsf{pp} := (q, \mathbb{G}, g) \xleftarrow{\$} \mathcal{G}(1^\kappa)$ where $\mathbb{G}$ is a cyclic group of order q with a random generator $g = [\![1]\!]$. Sample random $\beta, \gamma, u, v \xleftarrow{\$} \mathbb{Z}_q$. Output the tuple $(\mathsf{pp}, [\![1]\!], [\![\beta]\!], [\![\gamma]\!], [\![u]\!], [\![\beta v]\!], [\![\gamma(u+v)]\!])$.
2. Sample $\mathsf{pp} := (q, \mathbb{G}, g) \xleftarrow{\$} \mathcal{G}(1^\kappa)$ where $\mathbb{G}$ is a cyclic group of order q with a random generator $g = [\![1]\!]$. Sample random $\beta, \gamma, u, v, z \xleftarrow{\$} \mathbb{Z}_q$. Output the tuple $(\mathsf{pp}, [\![1]\!], [\![\beta]\!], [\![\gamma]\!], [\![u]\!], [\![\beta v]\!], [\![z]\!])$.

Without risk of ambiguity, we often say that the Decisional Linear assumption holds for the group $\mathbb{G}$ where $\mathbb{G}$ is the group sampled by the group generator $\mathcal{G}$.

The Vector Decisional Linear assumption. For convenience, the operational version of the Decisional Linear assumption we use is in fact a vectorized version, which is implied by the aforementioned standard Decisional Linear assumption through a standard hybrid argument. The Vector Decisional Linear assumption [SW21] posits that the following two distributions are computationally indistinguishable:

1. Sample $\mathsf{pp} := (q, \mathbb{G}, g) \xleftarrow{\$} \mathcal{G}(1^\kappa)$ where $\mathbb{G}$ is a cyclic group of order q with a random generator $g = [\![1]\!]$. Sample random $\beta, \gamma \xleftarrow{\$} \mathbb{Z}_q$, and random $\mathbf{u}, \mathbf{v} \xleftarrow{\$} \mathbb{Z}_q^n$. Output the tuple $(\mathsf{pp}, [\![1]\!], [\![\beta]\!], [\![\gamma]\!], [\![\mathbf{u}]\!], [\![\beta\mathbf{v}]\!], [\![\gamma(\mathbf{u}+\mathbf{v})]\!])$.
2. Sample $\mathsf{pp} := (q, \mathbb{G}, g) \xleftarrow{\$} \mathcal{G}(1^\kappa)$ where $\mathbb{G}$ is a cyclic group of order q with a random generator $g = [\![1]\!]$. Sample random $\beta, \gamma \xleftarrow{\$} \mathbb{Z}_q$, and random $\mathbf{u}, \mathbf{v}, \mathbf{z} \xleftarrow{\$} \mathbb{Z}_q^n$. Output the tuple $(\mathsf{pp}, [\![1]\!], [\![\beta]\!], [\![\gamma]\!], [\![\mathbf{u}]\!], [\![\beta\mathbf{v}]\!], [\![\mathbf{z}]\!])$.

Fact 4.2.1 ([SW21])**.** *Assume that the Decisional Linear assumption holds in $\mathbb{G}$, then the above Vector Decisional Linear assumption holds in $\mathbb{G}$ as well.*

4.3 The Decisional Bilinear Diffie-Hellman Assumption

*The Decisional Bilinear Diffie-Hellman Assumption (*DBDH*).* We say that the Decisional Bilinear Diffie-Hellman Assumption holds in group $\mathbb{G}$, which is part of a bilinear group $(\mathbb{G}, \mathbb{G}_T)$, if the following two experiments are computationally indistinguishable:

1. Sample $g = [\![1]\!] \xleftarrow{\$} \mathbb{G}$ where $\mathbb{G}$ is a cyclic group of order q. Sample random $a, b, c \xleftarrow{\$} \mathbb{Z}_q$. Output the tuple $([\![1]\!], [\![a]\!], [\![b]\!], [\![c]\!], [\![abc]\!]_T)$.
2. Sample $g = [\![1]\!] \xleftarrow{\$} \mathbb{G}$ where $\mathbb{G}$ is a cyclic group of order q. Sample random $a, b, c, z \xleftarrow{\$} \mathbb{Z}_q$. Output the tuple $([\![1]\!], [\![a]\!], [\![b]\!], [\![c]\!], [\![z]\!]_T)$.

The Q-fold Decisional Bilinear Diffie-Hellman Assumption. We say that the Q-fold Decisional Bilinear Diffie-Hellman Assumption holds in group $\mathbb{G}$, which is part of a bilinear group $(\mathbb{G}, \mathbb{G}_T)$, if the following two experiments are computationally indistinguishable:

1. Sample $g = [\![1]\!] \xleftarrow{\$} \mathbb{G}$ where $\mathbb{G}$ is a cyclic group of order q. Sample random $a, b, c_1, ..., c_Q \xleftarrow{\$} \mathbb{Z}_q$. Output the tuple $([\![1]\!], [\![a]\!], [\![b]\!], [\![c]\!], \{[\![abc_i]\!]_T\}_{i \in Q})$.
2. Sample $g = [\![1]\!] \xleftarrow{\$} \mathbb{G}$ where $\mathbb{G}$ is a cyclic group of order q. Sample random $a, b, c, z_1, ..., z_Q \xleftarrow{\$} \mathbb{Z}_q$. Output the tuple $([\![1]\!], [\![a]\!], [\![b]\!], [\![c]\!], \{[\![z_i]\!]_T\}_{i \in Q})$.

4.4 Function-Hiding (Single-Input) Inner Product Encryption

We will need a single-input inner-product encryption scheme — henceforth we call this building block Inner Production Encryption (IPE). IPE can be viewed as a special case of multi-client inner product encryption when $n = 1$. However, we will need our underlying IPE to satisfy a few nice properties, including the fact that **Enc** and **KGen** should still work when taking in the group encoding of the plaintext or key vector; moreover, we want that the scheme computes the "inner-product in the exponent".

Formally, the special IPE scheme we need consists of the following possibly randomized algorithms:

- $\mathsf{pp} \leftarrow \mathbf{Gen}(1^\lambda)$: takes in a security parameter λ and samples public parameters pp. We will assume that pp contains the description of a bilinear group $(\mathbb{G}, \mathbb{G}_T)$ of prime order q, a random generator $g \in \mathbb{G}$, and the description of the pairing operator $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.
- $\mathsf{imsk} \leftarrow \mathbf{Setup}(\mathsf{pp}, m)$: takes in the public parameters pp and the dimension m of the plaintext vector, outputs a secret key imsk.
- $\mathsf{sk}_{\mathbf{y}} \leftarrow \mathbf{KGen}(\mathsf{imsk}, [\![\mathbf{y}]\!])$: takes in the secret key imsk, and a vector of group elements $[\![\mathbf{y}]\!] \in \mathbb{G}^m$ which represents the group encoding of the vector $\mathbf{y} \in \mathbb{Z}_q^m$, outputs a functional (secret) key $\mathsf{sk}_{\mathbf{y}}$.
- $\mathsf{ct} \leftarrow \mathbf{Enc}(\mathsf{imsk}, [\![\mathbf{x}]\!])$: takes in the secret key imsk, a plaintext vector $[\![\mathbf{x}]\!] \in \mathbb{G}^m$ represented in group encoding, and outputs a ciphertext ct.
- $[\![v]\!]_T \leftarrow \mathbf{Dec}(\mathsf{sk}_{\mathbf{y}}, \mathsf{ct})$: takes in the functional key $\mathsf{sk}_{\mathbf{y}}$ and a ciphertext ct, and outputs a decrypted outcome $[\![v]\!]_T$.

Correctness. Correctness requires that for any $\lambda, m \in \mathbb{N}, \mathbf{x}, \mathbf{y} \in \mathbb{Z}_q^m$, the following holds with probability 1: let $\mathsf{pp} \leftarrow \mathbf{Gen}(1^\lambda)$, $\mathsf{imsk} \leftarrow \mathbf{Setup}(\mathsf{pp}, m)$, $\mathsf{sk}_{\mathbf{y}} \leftarrow \mathbf{KGen}(\mathsf{imsk}, [\![\mathbf{y}]\!])$, $\mathsf{ct} \leftarrow \mathbf{Enc}(\mathsf{imsk}, [\![\mathbf{x}]\!])$, $[\![v]\!]_T \leftarrow \mathbf{Dec}(\mathsf{sk}_{\mathbf{y}}, \mathsf{ct})$, then, it must be that $v := \langle \mathbf{x}, \mathbf{y} \rangle$.

Function-hiding security. Consider the following experiment $\mathsf{IPE\text{-}Expt}^b(1^\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:

Experiment $\mathsf{IPE\text{-}Expt}^b(1^\lambda)$:

- **Setup.** The challenger $\mathcal{C}$ runs $\mathsf{pp} \leftarrow \mathbf{Gen}(1^\lambda)$, and $\mathsf{imsk} \leftarrow \mathbf{Setup}(\mathsf{pp}, m)$, and gives pp to $\mathcal{A}$.
- **Query.** $\mathcal{A}$ makes the following types of queries to $\mathcal{C}$:
  - **KGen** queries: the adversary $\mathcal{A}$ submits $(\mathbf{y}^{(0)}, \mathbf{y}^{(1)})$; the challenger $\mathcal{C}$ computes $\mathsf{sk}_{\mathbf{y}^{(b)}} \leftarrow \mathbf{KGen}(\mathsf{msk}, [\![\mathbf{y}^{(b)}]\!])$ and returns to $\mathcal{A}$ the resulting $\mathsf{sk}_{\mathbf{y}^{(b)}}$.
  - **Enc** queries: the adversary $\mathcal{A}$ submits $(\mathbf{x}^{(0)}, \mathbf{x}^{(1)})$; the challenger $\mathcal{C}$ computes $\mathsf{ct} \leftarrow \mathbf{Enc}(\mathsf{mpk}, [\![\mathbf{x}^{(b)}]\!])$, and returns ct to $\mathcal{A}$.

An adversary $\mathcal{A}$ is said to be *admissible* iff the following holds with probability 1:

- all **KGen** queries must be made before any **Enc** query; and
- for any $(\mathbf{x}^{(0)}, \mathbf{x}^{(1)})$ tuple submitted in an **Enc** query, for any $(\mathbf{y}^{(0)}, \mathbf{y}^{(1)})$ tuple submitted in a **KGen** query, it must be that $\langle \mathbf{x}^{(0)}, \mathbf{y}^{(0)} \rangle = \langle \mathbf{x}^{(1)}, \mathbf{y}^{(1)} \rangle$.

Definition 3 (Function-hiding IND-security of IPE)**.** We say that the IPE scheme satisfies selective, function-hiding IND-security, iff for any non-uniform probabilistic polynomial-time (PPT) admissible adversary, its views in $\mathsf{IPE\text{-}Expt}^0$ and $\mathsf{IPE\text{-}Expt}^1$ are computationally indistinguishable.

Prior works [Wee16, ACF+18, SW21] have shown how to construct a function-hiding IPE scheme from the standard Decisional Linear assumption in bilinear groups. The idea is to first construct an IPE scheme without function privacy from Decisional Linear [Wee16, ACF+18, SW21] and then apply a function privacy upgrade [Lin17, Wee16, ACF+18, SW21] to obtain function hiding. The resulting constructions indeed satisfy the aforementioned nice properties that we need.

4.5 Correlated Pseudorandom Function

A correlated pseudorandom function family consists of the following randomized algorithms:

- $(K_1, K_2, \ldots, K_n) \leftarrow \mathbf{Gen}(1^\lambda, n, q)$: takes a security parameter 1^λ and the number of users n, some prime q, and outputs the user secret key K_i for each $i \in [n]$.
- $v \leftarrow \mathbf{Eval}(K_i, x)$: given a user secret key K_i and an input $x \in \{0,1\}^\lambda$, output an evaluation result $v \in \mathbb{Z}_q$.

Correctness. For correctness, we require that for any $\lambda \in \mathbb{N}$, any $(K_1, \ldots, K_n)$ in the support of $\mathbf{Gen}(1^\lambda)$, any input $x \in \{0,1\}^\lambda$, the following holds:

$$\sum_{i \in [n]} \mathsf{CPRF}.\mathbf{Eval}(K_i, x) = 0 \mod q$$

Correlated pseudorandomness. We require that for any non-uniform PPT adversary $\mathcal{A}$ who is allowed corrupt $f \leqslant n-2$ users and obtain their user secret keys, for any subset U of at most $n-f-1$ honest users, for any input x, the evaluations $\{\mathsf{CPRF}.\mathbf{Eval}(K_i, x)\}_{i \in U}$ are computationally indistinguishable from random values, as long as the adversary has not made a query on the input x.

More formally, correlated pseudorandomness is defined as below. Consider a game denoted $\mathsf{CPRF\text{-}Expt}^b(1^\lambda, n, q)$ between $\mathcal{A}$ and a challenger $\mathcal{C}$, parametrized by a bit $b \in \{0,1\}$.

- **Setup.** $\mathcal{A}$ submits a set of corrupt nodes $\mathcal{K} \subset [n]$ of size at most $n-2$. Henceforth, let $\mathcal{H} := [n] \backslash \mathcal{K}$. Now, $\mathcal{C}$ runs the honest $(K_1, \ldots, K_n) := \mathsf{CPRF}.\mathbf{Gen}(1^\lambda, n, q)$ algorithm, and gives $\{K_i\}_{i \in \mathcal{K}}$ to $\mathcal{A}$.
- **Queries.** $\mathcal{A}$ can adaptively make queries: for each query, $\mathcal{A}$ submits an input x. If $b = 0$, the challenger $\mathcal{C}$ chooses random $\{v_i\}_{i \in \mathcal{H}} \xleftarrow{\$} \mathbb{Z}_q^{|\mathcal{H}|}$ subject to the condition that $\sum_{i \in \mathcal{H}} v_i + \sum_{j \in \mathcal{K}} \mathsf{CPRF}.\mathbf{Eval}(K_j, x) = 0$, and returns $\{v_i\}_{i \in \mathcal{H}}$ to $\mathcal{A}$. Else if $b = 1$, the challenger gives $\{\mathsf{CPRF}.\mathbf{Eval}(K_i, x)\}_{i \in \mathcal{H}}$ to $\mathcal{A}$.

We say that a correlated pseudorandom function family CPRF satisfies correlated pseudorandomness, iff for any n and q, any non-uniform PPT adversary $\mathcal{A}$'s views in $\mathsf{CPRF\text{-}Expt}^0(1^\lambda, n, q)$ and $\mathsf{CPRF\text{-}Expt}^1(1^\lambda, n, q)$ are computationally indistinguishable.

Construction. Several prior works [BIK+17, ABG19, SW21] showed how to construct a correlated pseudorandom function from a standard pseudorandom function (PRF). Without loss of generality, we may assume that PRF's output range is $[0, q-1]$. During the setup phase denoted by $\mathbf{Gen}$, sample random PRF keys $k_{i,j}$ for all $i < j$, and let $k_{j,i} = k_{i,j}$. Party i's secret key K_i is defined to be the set $\{k_{i,j}\}_{j \in [n], j \neq i}$. The evaluation function $\mathbf{Eval}(K_i, x)$ is defined as follows:

$$\mathbf{Eval}(K_i, x) = \sum_{j \in [n], j \neq i} (-1)^{j<i} \cdot \mathsf{PRF}(k_{i,j}, x) \mod q$$

Prior works [BIK+17, ABG19, SW21] proved that the above CPRF construction satisfies correctness and correlated pseudorandomness, as long as the underlying PRF is secure.

4.6 Decentralized Secure Summation (in the Exponent)

A decentralized secure summation (in the exponent) (DSum) scheme, first formulated by Chotard et al. [CDSG+20]. consists of the following possibly randomized algorithms:

- $\mathsf{dpp} \leftarrow \mathbf{Gen}(1^\lambda, \mathbb{G})$: takes in a security parameter λ, the description of a cyclic group $\mathbb{G}$[1] of prime order, and samples and outputs public parameters dpp.
- $(\mathsf{dpk}_i, \mathsf{dsk}_i) \leftarrow \mathbf{USetup}(\mathsf{dpp})$: each client i runs the **USetup** algorithm, which takes in the public parameters dpp, and outputs a public and secret key pair denoted dpk_i and dsk_i, respectively.
- $\mathsf{ct}_i \leftarrow \mathbf{Enc}(\mathsf{dsk}_i, ([\![x]\!], \mathcal{U}, t), \{\mathsf{dpk}_j\}_{j\in\mathcal{U}})$: given the secret key dsk_i, an element $[\![x]\!] \in \mathbb{G}$, a label t, a set of clients $\mathcal{U} \subseteq [n]$ and their respective public keys $\{\mathsf{dpk}_j\}_{j\in\mathcal{U}}$, outputs a ciphertext ct_i; we may assume that ct_i is always tagged with t and $\mathcal{U}$.
- $[\![v]\!] \leftarrow \mathbf{Dec}(\mathsf{dpp}, \{\mathsf{ct}_i\}_{i\in\mathcal{U}})$: given the public parameters dpp, and a set of ciphertexts $\{\mathsf{ct}_i\}_{i\in\mathcal{U}}$ tagged with the labels t_i and sets $\mathcal{U}_i$, output a decrypted value $[\![v]\!] = f(\{i, ([\![x]\!]_i, \mathcal{U}_i, t_i)\}_{i\in\mathcal{U}})$.

The DSum function f is defined as follows:

$$f(\{i, ([\![x]\!]_i, \mathcal{U}_i, t_i)\}_{i\in\mathcal{U}}) = \begin{cases} [\![\sum_{i\in\mathcal{U}} x_i]\!] & \text{if the condition (*) below is satisfied} \\ \perp & \text{otherwise} \end{cases}$$

Condition (*) is defined as: there exists label t such that $\forall i \in \mathcal{U} : t_i = t$, and moreover, $\mathcal{U}_i = \mathcal{U}$.

In other words, fix some $\mathcal{U}$, t: given a collection of ciphertexts from users in $\mathcal{U}$ all tagged with $(\mathcal{U}, t)$, then an evaluator can compute the product $[\![\sum_{i\in\mathcal{U}} x_i]\!]$.

[1] Group $\mathbb{G}$ here is same as $\mathbb{G}$ of the bilinear groups $(\mathbb{G}, \mathbb{G}_T)$ used in the aMIIPE construction later.

Correctness. For correctness, we require that decryption correctly outputs the product (i.e., summation in the exponent) of the encrypted values, as long as the **Dec** algorithm is given a collection of ciphertexts from every client in the specified set $\mathcal{U}$, and moreover, all ciphertexts must be encrypted to the specified t and $\mathcal{U}$. More formally, the following holds with probability 1:

$$[\![v]\!] = f\left(\{i, ([\![x]\!]_i, \mathcal{U}_i, t_i)\}_{i\in\mathcal{U}}\right) : \begin{array}{l} \mathsf{pp} \leftarrow \mathbf{Gen}(1^\lambda, \mathbb{G}) \\ \forall i \in \mathcal{U} : (\mathsf{pk}_i, \mathsf{msk}_i) \leftarrow \mathbf{USetup}(\mathsf{pp}) \\ \forall i \in \mathcal{U} : \mathsf{ct}_i \leftarrow \mathbf{Enc}(\mathsf{dsk}_i, ([\![x]\!], \mathcal{U}, t), \{\mathsf{dpk}_j\}_{j\in\mathcal{U}}) \\ [\![v]\!] = \mathbf{Dec}(\mathsf{dpp}, \{\mathsf{ct}_i\}_{i\in\mathcal{U}}) \end{array}$$

Security. We now define the security requirement for DSum. Roughly speaking, we want that if all ciphertexts from everyone in $\mathcal{U}$ has been collected and all ciphertexts are encrypted to the same $(\mathcal{U}, t)$ pair, then, the decrypter can learn only the product of the messages encrypted. If not all ciphertexts from everyone in $\mathcal{U}$ have been collected, the decrypter learns nothing. More formally, consider the following security experiment.

Experiment $\mathsf{DSum\text{-}Expt}^b(1^\lambda)$:

- **Setup.** $\mathcal{A}(1^\lambda)$ outputs a set of parties to corrupt $\mathcal{K} \subset [n]$. The challenger $\mathcal{C}$ runs $\mathsf{dpp} \leftarrow \mathbf{Gen}(1^\lambda, \mathbb{G})$, and for $i \in [n]$, it runs $(\mathsf{dpk}_i, \mathsf{dsk}_i) \leftarrow \mathbf{USetup}(\mathsf{dpp})$. It returns $\{\mathsf{dpk}_i\}_{i\in[n]}$ as well as $\{\mathsf{dsk}_i\}_{i\in\mathcal{K}}$ to $\mathcal{A}$.
- **Query.** The adversary can make **Enc** queries of the following form: for each **Enc** query,
  - the adversary submits a set $\mathcal{U} \subseteq [n]$, a label t, a client $i \in \mathcal{U}$, and a pair of messages $(x^{(0)}, x^{(1)})$,
  - the challenger $\mathcal{C}$ calls $\mathsf{ct} \leftarrow \mathbf{Enc}(\mathsf{dsk}_i, (x^{(b)}, \mathcal{U}, t), \{\mathsf{dpk}_j\}_{j\in\mathcal{U}})$, and returns ct to $\mathcal{A}$.

In the above experiment, if for some $(\mathcal{U}, t)$ pair, the adversary $\mathcal{A}$ has submitted an encryption query for every $i \in \mathcal{U}\backslash\mathcal{K}$ pertaining to the pair $(\mathcal{U}, t)$, then we say that $(\mathcal{U}, t)$ is *complete*. Else, the pair $(\mathcal{U}, t)$ is said to be *incomplete*. We say that the adversary $\mathcal{A}$ is *admissible* iff the following conditions hold:

- For every **Enc** query submitted for some corrupt $i \in \mathcal{K}$, let $(x^{(0)}, x^{(1)})$ be the two plaintext messages submitted, then, it must be that $x^{(0)} = x^{(1)}$.

- For every *complete* pair $(\mathcal{U}, t)$, let $(x_i^{(0)}, x_i^{(1)})$ be the pair of messages submitted in an **Enc** query for $i \in \mathcal{U}$ and pertaining to the pair $(\mathcal{U}, t)$, then, it must be that $\prod_{i \in \mathcal{U} \setminus \mathcal{K}} x_i^{(0)} = \prod_{i \in \mathcal{U} \setminus \mathcal{K}} x_i^{(1)}$.

We say that a DSum scheme is IND-secure, iff for every non-uniform PPT admissible adversary, its views in $\mathsf{DSum\text{-}Expt}^0(1^\lambda)$ and $\mathsf{DSum\text{-}Expt}^1(1^\lambda)$ are computationally indistinguishable. We say that a DSum scheme is *selective*-IND-secure, if in the above game, the adversary sends all the encryption queries at once.

5

Function-Hiding Multi-Client Inner-Product Encryption

In this section, we give our detailed construction of function-hiding multi-client inner-product encryption scheme and its formal proof.

5.1 Detailed Construction

Let $\mathsf{IPE} := (\mathbf{Gen}, \mathbf{Setup}, \mathbf{KGen}, \mathbf{Enc}, \mathbf{Dec})$ denote a function-hiding inner-product encryption scheme, and let $\mathsf{CPRF} := (\mathbf{Gen}, \mathbf{Eval})$ denote a correlated pseudorandom function.

Function-hiding, multi-client inner-product encryption

- $\mathbf{Gen}(1^\lambda)$: let $\mathsf{pp} \leftarrow \mathsf{IPE}.\mathbf{Gen}(1^\lambda)$, and output pp.
- $\mathbf{Setup}(\mathsf{pp}, m, n)$:
  - let $(K_1, \ldots, K_n) := \mathsf{CPRF}.\mathbf{Gen}(1^\lambda, n, q)$;
  - for $i \in [n]$: let $\mathsf{imsk}_i \leftarrow \mathsf{IPE}.\mathbf{Setup}(\mathsf{pp}, 2m+3)$, and $a_i \xleftarrow{\$} \mathbb{Z}_q$;
  - output
$$\mathsf{mpk} := \mathsf{pp}, \quad \mathsf{msk} := \{\mathsf{imsk}_i, a_i\}_{i\in[n]}, \text{and}$$
$$\{\mathsf{ek}_i := (\mathsf{imsk}_i, K_i, a_i)\}_{i\in[n]}$$
- $\mathbf{KGen}(\mathsf{mpk}, \mathsf{msk}, \mathbf{y})$:
  - sample $\rho \xleftarrow{\$} \mathbb{Z}_q$;
  - let $\widetilde{\mathbf{y}}_i = (\mathbf{y}_i, 0^m, \rho, -\rho a_i, 0)$;
  - let $\mathsf{isk}_i \leftarrow \mathsf{IPE}.\mathbf{KGen}(\mathsf{imsk}_i, [\![\widetilde{\mathbf{y}}]\!]_i)$, and output $\mathsf{sk}_\mathbf{y} := \{\mathsf{isk}_i\}_{i\in[n]}$.

- $\mathbf{Enc}(\mathsf{mpk}, \mathsf{ek}_i, \mathbf{x}_i, t)$:
  - sample $\mu_{i,t} \xleftarrow{\$} \mathbb{Z}_q$ if $\mu_{i,t}$ has not been sampled before;
  - let $\widetilde{\mathbf{x}}_i = (\mathbf{x}_i, 0^m, \mathsf{CPRF.Eval}(K_i, t) + a_i\mu_{i,t}, \mu_{i,t}, 0)$;
  - call $\mathsf{ct} \leftarrow \mathsf{IPE.Enc}(\mathsf{imsk}_i, [\![\widetilde{\mathbf{x}}_i]\!])$, and output ct.
- $\mathbf{Dec}(\mathsf{mpk}, \mathsf{sk}_{\mathbf{y}}, \{\mathsf{ct}_{i,t}\}_{i\in[n]})$: let $[\![v]\!]_T := \prod_{i\in[n]} \mathsf{IPE.Dec}(\mathsf{isk}_i, \mathsf{ct}_i)$, and output $v := \log([\![v]\!]_T)$.

Asymptotical efficiency. We can instantiate the function-hiding IPE using the scheme described in earlier works [Wee16, ACF+18, SW21], based on the Decisional Linear assumption. For the underlying IPE scheme, the ciphertext contains $O(m)$ group elements where m is the length of the vector being encrypted. Similarly, each functional key has only $O(m)$ group elements too. The public parameters contain only the group description.

In our MCIPE construction, to encrypt a length-m vector, each client's ciphertext has only $O(m)$ group elements. A functional key for a length $(n \cdot m)$-vector has size $O(n \cdot m)$ group elements. Each client's secret key has size $O(n)$ where the big-O hides terms related to the security parameter. The public parameters contain only the group description.

Theorem 5.1.1 (Restatement of Theorem 1.1.1)**.** *Suppose that the Decisional Linear assumption holds in* $\mathbb{G}$*,* IPE *satisfies selective, function-hiding IND-security (see Definition 3), and moreover,* CPRF *satisfies correlated pseudorandomness. Then, the above* MCIPE *scheme satisfies selective function-hiding IND-security.*

Proof. The proof is presented next in Section 5.2 □

5.2 Proof of Theorem 5.1.1

We consider a sequence of hybrid experiments.

Experiment $\mathsf{MCIPE\text{-}Expt}^1$. This is the real-world experiment, parametrized by $b = 1$. In the experiment $\mathsf{MCIPE\text{-}Expt}^1$, the challenger $\mathcal{C}$ answers **Enc** and **KGen** queries using the following vectors:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, 0^m, \mathsf{CPRF.Eval}(K_i, t) + a_i\mu_{i,t}, \mu_{i,t}, 0\right), \quad \widetilde{\mathbf{y}}_i = \left(\mathbf{y}_i^{(1)}, 0^m, \rho, -\rho a_i, 0\right)$$

where ρ is freshly chosen for every **KGen** query.

Experiment Hyb_0. Same as $\mathsf{MCIPE\text{-}Expt}^b$ except that for any honest $i \in \mathcal{H}$, the challenger $\mathcal{C}$ answers **Enc** queries using

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, \mathsf{CPRF.Eval}(K_i, t) + a_i\mu_{i,t}, \mu_{i,t}, 0\right)$$

Since this modification preserves the inner products $\langle \widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}_i \rangle$ for any pair of encryption and key vectors queried, and for any $i \in \mathcal{H}$, Hyb_0 is computationally indistinguishable from $\mathsf{MCIPE\text{-}Expt}^1$ due to the function-hiding IND-security of the IPE scheme.

Experiment Hyb_ℓ. We next define a sequence of hybrid experiments Hyb_ℓ where $\ell \in [Q_{\text{kgen}}]$ where Q_{kgen} denotes an upper bound the number of **KGen** queries made by $\mathcal{A}$. In Hyb_ℓ, for the first ℓ **KGen** queries, the challenger $\mathcal{C}$ uses $\widetilde{\mathbf{y}}_i = \left(0^m, \mathbf{y}_i^{(0)}, \rho, -\rho a_i, 0\right)$ for any honest $i \in \mathcal{H}$, and uses $\widetilde{\mathbf{y}}_i = \left(\mathbf{y}_i^{(1)}, 0^m, \rho, -\rho a_i, 0\right)$ for any corrupt $i \in \mathcal{K}$. For the remaining $Q_{\text{kgen}} - \ell$ number of **KGen** queries, $\mathcal{C}$ uses $\widetilde{\mathbf{y}}_i = \left(\mathbf{y}_i^{(1)}, 0^m, \rho, -\rho a_i, 0\right)$ for all $i \in [n]$.

In Lemma 5.2.1, we prove that $\mathsf{Hyb}_{\ell-1}$ is computationally indistinguishable from Hyb_ℓ for $\ell \in [Q_{\text{kgen}}]$.

Experiment Hyb^*. The challenger $\mathcal{C}$ answers **Enc** and **KGen** queries using the following vectors for any honest $i \in \mathcal{H}$:

$$\widetilde{\mathbf{x}}_i = \left(0^m, \mathbf{x}_i^{(0)}, \mathsf{CPRF.Eval}(K_i, t) + a_i\mu_{i,t}, \mu_{i,t}, 0\right), \quad \widetilde{\mathbf{y}}_i = \left(0^m, \mathbf{y}_i^{(0)}, \rho, -\rho a_i, 0\right)$$

For corrupt $i \in \mathcal{K}$, the challenger $\mathcal{C}$ still uses:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, 0^m, \mathsf{CPRF.Eval}(K_i, t) + a_i\mu_{i,t}, \mu_{i,t}, 0\right), \quad \widetilde{\mathbf{y}}_i = \left(\mathbf{y}_i^{(1)}, 0^m, \rho, -\rho a_i, 0\right)$$

Observe that $\mathsf{Hyb}_{Q_{\text{kgen}}}$ and Hyb^* are almost identical except that the first m coordinates in $\widetilde{\mathbf{x}}_i$ are replaced with 0^m for $i \in \mathcal{H}$. Since this modification preserves the inner products $\langle \widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}_i \rangle$

for any pair of encryption and key vectors queried, and for any $i \in \mathcal{H}$, Hyb^* is computationally indistinguishable from $\mathsf{Hyb}_{Q_{\mathrm{kgen}}}$ due to the function-hiding IND-security of the IPE scheme.

Finally, observe that Hyb^* is computationally indistinguishable from $\mathsf{MCIPE\text{-}Expt}^0$ since for honest $i \in \mathcal{H}$, the inner-product $\langle \widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}_i \rangle$ is preserved for any pair of encryption and key vectors queried; and for corrupt $i \in \mathcal{K}$, recall that our admissibility stipulates that $\mathbf{x}_i^{(0)} = \mathbf{x}_i^{(1)}$ and $\mathbf{y}_i^{(0)} = \mathbf{y}_i^{(1)}$, and thus it makes no difference whether $\mathbf{x}_i^{(0)}, \mathbf{y}_i^{(0)}$ is used or whether $\mathbf{x}_i^{(1)}, \mathbf{y}_i^{(1)}$ is used by $\mathcal{C}$.

Therefore, to complete the proof of Theorem 5.1.1, it suffices to prove the following lemma, which shows the computational indistinguishability of $\mathsf{Hyb}_{\ell-1}$ and Hyb_ℓ.

Lemma 5.2.1. *Suppose that the Decisional Linear assumption holds in* $\mathbb{G}$, IPE *satisfies function-hiding IND-security, and moreover,* CPRF *satisfies correlated pseudorandomness. Then,* $\mathsf{Hyb}_{\ell-1}$ *is computationally indistinguishable from* Hyb_ℓ *for any* $\ell \in [Q_{\mathrm{kgen}}]$.

Proof. We consider a sequence of hybrid experiments.

Experiment $\mathsf{H}_{\ell-1,1}$. In experiment $\mathsf{H}_{\ell-1,1}$, for any honest $i \in \mathcal{H}$, the challenger $\mathcal{C}$ uses the following vectors to answer the **Enc** and **KGen** queries where $\rho^* \xleftarrow{\$} \mathbb{Z}_q$, and we use $\mathbf{y}^{*(0)}$, $\mathbf{y}^{*(1)}$ to denote the key vectors submitted during the ℓ-th **KGen** query:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, \mathsf{CPRF}.\mathbf{Eval}(K_i, t) + a_i \mu_{i,t}, \mu_{i,t}, \mathsf{CPRF}.\mathbf{Eval}(K_i, t) \cdot \rho^* + \langle \mathbf{x}_i^{(1)}, \mathbf{y}_i^{*(1)} \rangle\right),$$

$$\widetilde{\mathbf{y}}_i = \begin{cases} \left(0^m, \mathbf{y}_i^{(0)}, \rho, -\rho a_i, 0\right) & \text{first } \ell - 1 \textbf{ KGen} \text{ queries} \\ (0^m, 0^m, 0, 0, 1) & \ell\text{-th } \textbf{KGen} \text{ query} \\ \left(\mathbf{y}_i^{(1)}, 0^m, \rho, -\rho a_i, 0\right) & \text{remaining } Q_{\mathrm{kgen}} - \ell \textbf{ KGen} \text{ queries} \end{cases}$$

In the above, ρ is freshly chosen for every **KGen** query, and ρ^* corresponds to the randomness chosen for the challenge **KGen** query, i.e., the ℓ-th **KGen** query.

Observe that $\mathsf{H}_{\ell-1,1}$ is almost identical to $\mathsf{Hyb}_{\ell-1}$ except for the above modifications highlighted in blue. Since these modification preserves the inner products $\langle \widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}_i \rangle$ for any pair of

encryption and key vectors queried, and for any $i \in \mathcal{H}$, $\mathsf{H}_{\ell-1,1}$ and $\mathsf{Hyb}_{\ell-1}$ are computationally indistinguishable due to the function-hiding IND-security of the IPE scheme.

Experiment $\mathsf{H}_{\ell-1,2}$. Almost identical to $\mathsf{H}_{\ell-1,1}$, except that for each t label that appears first in an **Enc** query, the challenger $\mathcal{C}$ chooses $\{R_{i,t}\}_{i\in\mathcal{H}}$ at random from $\mathbb{Z}_q$ subject to $\sum_{i\in\mathcal{H}} R_{i,t} = -\sum_{i\in\mathcal{K}} \mathsf{CPRF}.\mathbf{Eval}(K_i, t)$. For honest $i \in \mathcal{H}$, the challenger $\mathcal{C}$ uses the following vector to answer **Enc** queries:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, R_{i,t} + a_i\mu_{i,t}, \mu_{i,t}, R_{i,t}\cdot\rho^* + \langle \mathbf{x}_i^{(1)}, \mathbf{y}_i^{*(1)}\rangle\right),$$

Experiment $\mathsf{H}_{\ell-1,2}$ is computationally indistinguishable from $\mathsf{H}_{\ell-1,1}$ due to the correlated pseudorandomness of CPRF.

Observe that in this hybrid, the challenger needs to know challenge key vector $\mathbf{y}^{*(1)}$ when answering **Enc** queries, and this is why our proof technique works only for selective security, where the $\mathcal{A}$ must make all **KGen** queries ahead of any **Enc** query.

Experiment $\mathsf{H}_{\ell-1,3}$. Almost identical to $\mathsf{H}_{\ell-1,2}$, except that the challenger $\mathcal{C}$ chooses random $\{T_{i,t}\}_{i\in\mathcal{H}}$ subject to $\sum_{i\in\mathcal{H}} T_{i,t} = -\rho^* \cdot \sum_{i\in\mathcal{K}} \mathsf{CPRF}(K_i, t)$, and uses the following vector in any **Enc** query for an honest $i \in \mathcal{H}$:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, R_{i,t} + a_i\mu_{i,t}, \mu_{i,t},\ T_{i,t} + \langle \mathbf{x}_i^{(1)}, \mathbf{y}_i^{*(1)}\rangle\right)$$

Claim 5.2.2. *Suppose that the Decisional Linear assumption holds in* $\mathbb{G}$. *Then,* $\mathsf{H}_{\ell-1,3}$ *is computationally indistinguishable from* $\mathsf{H}_{\ell-1,2}$.

Proof. We will consider a sequence of hybrid experiments over the set of honest clients. Henceforth let d be the number of honest clients, and let $\mathcal{H} := \{i_1, i_2, \ldots, i_d\} \subseteq [n]$ denote the set of honest clients. We define the hybrid G_j as follows where $j \in [d-1] \cup \{0\}$:

- If i is among the first j honest clients, then the challenger $\mathcal{C}$ chooses $\widetilde{T}_{i,t}$ at random;
- If i is not among the first j honest clients and $i \neq i_d$, then, the challenger $\mathcal{C}$ chooses $\widetilde{T}_{i,t} = R_{i,t}\cdot\rho^*$;

- For the last honest client $i = i_d$, the challenger $\mathcal{C}$ chooses $\widetilde{T}_{i,t}$ such that $\sum_{i\in\mathcal{H}} \widetilde{T}_{i,t} = -\sum_{i\in\mathcal{K}} \mathsf{CPRF.Eval}(K_i, t)$.

The challenger uses the following vector when answering **Enc** queries for any honest $i \in \mathcal{H}$:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, R_{i,t} + a_i\mu_{i,t}, \mu_{i,t},\ \widetilde{T}_{i,t} + \langle \mathbf{x}_i^{(1)}, \mathbf{y}_i^{*(1)} \rangle\right) \tag{5.1}$$

Observe that G_0 is the same as $\mathsf{H}_{\ell-1,2}$, and G_{d-1} is the same as $\mathsf{H}_{\ell-1,3}$. Therefore, to prove Claim 5.2.2, it suffices to prove that any two adjacent hybrids G_j and G_{j+1} are computationally indistinguishable for $\ell \in \{0, 1, \ldots, d-2\}$. Below, we prove that if the Decisional Linear assumption holds in $\mathbb{G}$, then indeed G_j and G_{j+1} are computationally indistinguishable for $\ell \in \{0, 1, \ldots, d-2\}$.

Suppose there is an efficient adversary $\mathcal{A}$ that can distinguish G_j and G_{j+1} with non-negligible probability, we show how to construct an efficient reduction $\mathcal{B}$ that can break the Decisional Linear assumption. Let Q_{enc} denote the maximum number of labels t submitted during **Enc** queries. $\mathcal{B}$ obtains an instance $([\![1]\!], [\![\beta]\!], [\![\gamma]\!], [\![\mathbf{u}]\!], [\![\beta\mathbf{v}]\!], [\![\mathbf{z}]\!])$ from a Vector Decisional Linear challenger (see Section 4.2), where $\mathbf{u}, \mathbf{v}, \mathbf{z} \in \mathbb{Z}_q^{Q_{\text{enc}}}$ and $\beta, \gamma \in \mathbb{Z}_q$. $\mathcal{B}$'s task is to distinguish whether $[\![\mathbf{z}]\!] = [\![\gamma(\mathbf{u}+\mathbf{v})]\!]$ or random. $\mathcal{B}$ will now interact with $\mathcal{A}$ and embed this Decisional Linear instance in its answers.

Let $i^* = i_{j+1} \in \mathcal{H}$ be the index of the $(j+1)$-th honest client.

- **Setup**. $\mathcal{B}$ chooses $\xi \in \mathbb{Z}_q$ at random, and implicitly sets $a_{i^*} = \beta^{-1}$ and $a_{i_d} = \xi \cdot \beta^{-1}$, without actually computing them. $\mathcal{B}$ chooses all other terms in the **Setup** algorithm honestly, and gives mpk and $\{\mathsf{ek}_i\}_{i\in\mathcal{K}}$ to $\mathcal{A}$.

- **KGen queries.**

  1. For the first $\ell - 1$ **KGen** queries:

     – for any honest $i \in \mathcal{H}$, $i \neq i^*$ and $i \neq i_d$, $\mathcal{B}$ knows all the terms necessary to compute isk_i.

- for $i = i^*$, the reduction $\mathcal{B}$ does not know a_{i^*}, but it can replace the terms $[\![\rho, -\rho a_{i^*}]\!]$ with $[\![\beta\rho', -\rho']\!]$ instead where $\rho' \xleftarrow{\$} \mathbb{Z}_q$. It can compute $[\![\beta\rho]\!]$ because it knows $[\![\beta]\!]$ and ρ'. $\mathcal{B}$ can now continue computing $\mathsf{isk}_{i^*} \leftarrow \mathsf{IPE.KGen}(\mathsf{imsk}_i, [\![0^m, \mathbf{y}_i^{(0)}, \beta\rho', -\rho', 0]\!])$ normally.
- for $i = i_d$, $\mathcal{B}$ can compute isk_{i_d} in a similar fashion as above, even if it does not know $a_{i_d} = \xi \cdot \beta^{-1}$.
- for any corrupt $i \in \mathcal{K}$, $\mathcal{B}$ computes isk_i using the original honest algorithm, since it knows all the necessary terms.

2. For any **KGen** query after the first ℓ queries, the reduction $\mathcal{B}$ can compute functional key just like for the first $\ell - 1$ queries.
3. For the ℓ-th **KGen** query, $\mathcal{B}$ wants to embed the γ term from the Decisional Linear challenge into the ρ term for this specific functional key. Recall that $\mathcal{B}$ knows only $[\![\gamma]\!]$ but not γ itself.
   - For any corrupt $i \in \mathcal{K}$, observe that $\mathcal{B}$ can compute their respective key component isk_i knowing only $[\![\gamma]\!]$ but not γ itself.
   - For any honest $i \in \mathcal{H}$, $\mathcal{B}$ computes $\mathsf{isk}_i \leftarrow \mathsf{IPE.KGen}(\mathsf{imsk}_i, [\![0^m, 0^m, 0, 0, 1]\!])$.

- **Enc queries.** The adversary $\mathcal{A}$ submits $(i, t, \mathbf{x}_{i,t}^{(0)}, \mathbf{x}_{i,t}^{(1)})$. If $i \in \mathcal{K}$, $\mathcal{B}$ can compute the ciphertext normally since it knows all the necessary terms. Below we focuse on the case when $i \in \mathcal{H}$. The first time the label t appears in an **Enc** query for some honest $i \in \mathcal{H}$, the reduction $\mathcal{B}$ picks $\{\widetilde{T}_{i,t}\}_{i \in \mathcal{H}}$ as follows, where u_t, v_t, and z_t denote the t-th component of the vector $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{z}$ from the Decisional Linear challenge[1].

  (a) If $i \in \mathcal{H}$, $i \neq i^*$, and $i \neq i_d$: $\mathcal{B}$ chooses $\widetilde{T}_{i,t}$ at random if i is among the first j honest clients, else it implicitly lets $\widetilde{T}_{i,t} := R_{i,t} \cdot \gamma$.

[1] For convenience, we may imagine that the labels t have been renamed to be the integers $\{1, 2, \ldots, Q_{\mathrm{enc}}\}$.

(b) If $i = i^*$: $\mathcal{B}$ implicitly chooses

$$R_{i^*,t} + a_{i^*}\mu_{i^*,t} = u_t, \quad \mu_{i^*,t} = -\beta v_t, \quad \widetilde{T}_{i^*,t} = z_t$$

(c) If $i = i_d$: $\mathcal{B}$ samples $\phi \xleftarrow{\$} \mathbb{Z}_q$, and implicitly chooses

$$\mu_{i_d,t} = -\mu_{i^*,t} \cdot \xi^{-1} + a_{i^*}^{-1} \cdot \phi, \; R_{i_d,t} = -\left(\sum_{i \in \mathcal{H}, i \neq i_d} R_{i,t} + \sum_{i \in \mathcal{K}} \mathsf{CPRF.Eval}(K_i, t) \right),$$

$$\widetilde{T}_{i_d,t} = -\left(\sum_{i \in \mathcal{K}} \mathsf{CPRF.Eval}(K_i, t) + \sum_{i \in \mathcal{H}, i \neq i^*, i \neq i_d} \widetilde{T}_i + z_t \right)$$

For case (a), computing the ciphertext (see Equation 5.1) is straightforward. For case (b), it is also easy to see that given $\mathcal{B}$'s knowledge of $[\![u_t]\!]$, $[\![\beta v_t]\!]$, and $[\![z_t]\!]$, one can compute the ciphertext in a straightforward way. For case (c), observe the following. Let

$$\nu = -\left(\sum_{i \in \mathcal{H}, i \neq i_d, i \neq i^*} R_{i,t} + \sum_{i \in \mathcal{K}} \mathsf{CPRF.Eval}(K_i, t) \right);$$

and thus $R_{i_d,t} = \nu - R_{i^*,t}$.

$$\begin{aligned}
[\![R_{i_d,t} + a_{i_d}\mu_{i_d,t}]\!] &= [\![\nu - R_{i^*,t} + \xi a_{i^*} \cdot (-\mu_{i^*,t} \cdot \xi^{-1} + a_{i^*}^{-1} \cdot \phi)]\!] \\
&= [\![\nu - R_{i^*,t} - a_{i^*}\mu_{i^*,t} + \xi \cdot \phi]\!] \\
&= [\![\nu - u_t + \xi \cdot \phi]\!]
\end{aligned}$$

Further, $[\![\mu_{i_d,t}]\!] = [\![\beta v_t \cdot \xi^{-1} + \beta \cdot \phi]\!]$. Therefore, both $[\![R_{i_d,t} + a_{i_d}\mu_{i_d,t}]\!]$ and $[\![\mu_{i_d,t}]\!]$ can be computed knowing ν, $[\![u_t]\!]$, ξ, ϕ, $[\![\beta v_t]\!]$, and $[\![\beta]\!]$.

Observe that

$$R_{i^*,t} \cdot \gamma = (u_t - a_{i^*}\mu_{i^*,t})\gamma = (u_t + \beta^{-1} \cdot \beta v_t)\gamma = (u_t + v_t)\gamma$$

Therefore, in the Decisional Linear Challenge $([\![1]\!], [\![\beta]\!], [\![\gamma]\!], [\![\mathbf{u}]\!], [\![\beta\mathbf{v}]\!], [\![\mathbf{z}]\!])$ obtained by $\mathcal{B}$, if $\mathbf{z} = \gamma(\mathbf{u} + \mathbf{v})$, then $\mathcal{A}$'s view is identically distributed as in G_j. Else $\mathcal{A}$'s view is identically distributed as in G_{j+1}. □

We now continue with the proof of Lemma 5.2.1.

Experiment $\mathsf{H}'_{\ell-1,3}$. Almost identical to $\mathsf{H}_{\ell-1,3}$, except that the challenger $\mathcal{C}$ uses the following vector in any **Enc** query for an honest $i \in \mathcal{H}$:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, R_{i,t} + a_i\mu_{i,t}, \mu_{i,t},\ T_{i,t} + \langle \mathbf{x}_i^{(0)}, \mathbf{y}_i^{*(0)} \rangle\right)$$

where the terms $\{T_{i,t}\}_{i\in\mathcal{H}}$ are chosen at random subject to $\sum_{i\in\mathcal{H}} T_{i,t} = -\rho^* \cdot \sum_{i\in\mathcal{K}} \mathsf{CPRF}(K_i, t)$.

As long as $\mathcal{A}$ respects the admissibility rule defined in Section 3.1, $\mathsf{H}_{\ell-1,3}$ and $\mathsf{H}'_{\ell-1,3}$ are identically distributed.

Experiment $\mathsf{H}'_{\ell-1,2}$. Almost identical to $\mathsf{H}_{\ell-1,2}$, except that the challenger $\mathcal{C}$ chooses uses the following vector to answer **Enc** queries:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, R_{i,t} + a_i\mu_{i,t}, \mu_{i,t}, R_{i,t} \cdot \rho^* + \langle \mathbf{x}_i^{(0)}, \mathbf{y}_i^{*(0)} \rangle\right),$$

Experiment $\mathsf{H}'_{\ell-1,1}$. Almost identical to $\mathsf{H}_{\ell-1,1}$, except that the challenger $\mathcal{C}$ chooses uses the following vector to answer **Enc** queries:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, \mathsf{CPRF}.\mathbf{Eval}(K_i, t) + a_i\mu_{i,t}, \mu_{i,t}, \mathsf{CPRF}.\mathbf{Eval}(K_i, t) \cdot \rho^* + \langle \mathbf{x}_i^{(0)}, \mathbf{y}_i^{*(0)} \rangle\right),$$

Using a symmetric argument as before, we can prove the computational indistinguishability between $\mathsf{H}'_{\ell-1,3}$ and $\mathsf{H}'_{\ell-1,2}$, and between $\mathsf{H}'_{\ell-1,2}$ and $\mathsf{H}'_{\ell-1,1}$.

Finally, $\mathsf{H}'_{\ell-1,1}$ and Hyb_ℓ are computationally indistinguishable due to the function-hiding IND-security of the IPE scheme, since the inner-product $\langle \widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}_i \rangle$ is preserved for any pair of encryption and key vectors queried and for $i \in \mathcal{H}$. This concludes the proof of Lemma 5.2.1.

□

6

Function-Hiding Ad Hoc Multi-Input Inner-Product Encryption

In this section, we give our detailed construction of function-hiding ad hoc multi-input inner-product encryption scheme and its formal proof.

6.1 Construction

Let $\mathsf{IPE} := (\mathbf{Gen}, \mathbf{USetup}, \mathbf{KGen}, \mathbf{Enc}, \mathbf{Dec})$ denote a function-hiding inner-product encryption scheme and let $\mathsf{DSum} := (\mathbf{Gen}, \mathbf{USetup}, \mathbf{Enc}, \mathbf{Dec})$ denote a decentralized secure summation scheme.

Function-hiding, ad hoc multi-input inner-product encryption

- $\mathbf{Gen}(1^\lambda, m)$:
  - let $\mathsf{ipp} \leftarrow \mathsf{IPE}.\mathbf{Gen}(1^\lambda)$
  - let $\mathsf{dpp} \leftarrow \mathsf{DSum}.\mathbf{Gen}(1^\lambda, \mathbb{G})$, where $\mathbb{G} \in \mathsf{ipp}$
  - output $\mathsf{pp} = (\mathsf{ipp}, \mathsf{dpp}, m)$.
- $\mathbf{USetup}(\mathsf{pp})$:
  - let $\mathsf{imsk}_i \leftarrow \mathsf{IPE}.\mathbf{Setup}(\mathsf{ipp}, 2m+2)$,
  - let $(\mathsf{dpk}_i, \mathsf{dsk}_i) \leftarrow \mathsf{DSum}.\mathbf{USetup}(\mathsf{dpp})$
  - output $\mathsf{pk}_i := (\mathsf{pp}, \mathsf{dpk}_i), \mathsf{msk}_i := (\mathsf{imsk}_i, \mathsf{dsk}_i)$

- $\mathbf{KGen}(\mathsf{msk}_i, (\mathbf{y}_i, \mathcal{U}_i, t_i), \{\mathsf{pk}_j\}_{j\in\mathcal{U}_i})$:
  - sample $r_i \xleftarrow{\$} \mathbb{Z}_q$, where q is the prime order of group $\mathbb{G}$
  - let $\widetilde{\mathbf{y}}_i = (\mathbf{y}_i, 0^m, r_i, 0)$;
  - let $\mathsf{isk}_i \leftarrow \mathsf{IPE}.\mathbf{KGen}(\mathsf{imsk}_i, [\![\widetilde{\mathbf{y}}_i]\!])$
  - let $d_i = r_i$ and $\mathsf{dct}_i \leftarrow \mathsf{DSum}.\mathbf{Enc}(\mathsf{dsk}_i, ([\![d_i]\!], \mathcal{U}_i, t_i), \{\mathsf{dpk}_i\}_{i\in\mathcal{U}_i})$.
  - output $\mathsf{sk}_i := \{\mathsf{isk}_i, \mathsf{dct}_i\}$.
- $\mathbf{Enc}(\mathsf{pk}_i, \mathsf{msk}_i, \mathbf{x}_i)$:
  - let $\widetilde{\mathbf{x}}_i = (\mathbf{x}_i, 0^m, 1, 0)$;
  - output $\mathsf{ct}_i \leftarrow \mathsf{IPE}.\mathbf{Enc}(\mathsf{imsk}_i, [\![\widetilde{\mathbf{x}}_i]\!])$.
- $\mathbf{Dec}(\{\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ct}_i\}_{i\in\mathcal{U}_K})$:
  - compute $[\![v]\!]_T := \prod_{i\in\mathcal{U}_K} \mathsf{IPE}.\mathbf{Dec}(\mathsf{isk}_i, \mathsf{ct}_i)$
  - compute $[\![z]\!] = [\![\sum_{i\in\mathcal{U}_K} r_i]\!] \leftarrow \mathsf{DSum}.\mathbf{Dec}(\mathsf{dpp}, \{\mathsf{dct}_i\}_{i\in\mathcal{U}_K})$
  - output $v := \log([\![v]\!]_T / [\![z]\!]_T)$

Note. For our aMIIPE scheme to be efficient, we need a new, efficient DSum scheme without random oracles. The earlier work by Chotard et al. [CDSG+20] showed that DSum can be constructed from an "all-or-nothing encryption" scheme. They then suggested two all-or-nothing encryption schemes: 1) one without random oracles, but the per-user ciphertext size is proportional to the number of users n; and 2) one with random oracles where the ciphertext length are independent of n. To get a DSum scheme whose per-user ciphertext size is independent of n, we will need an underlying all-or-nothing encryption scheme with corresponding efficiency. We give such a construction without random oracles in Appendix A.

Correctness. Observe that -

$$\begin{aligned}
[\![v]\!]_T &= [\![\Sigma_{i\in\mathcal{U}_K}\langle \mathbf{x}_i||0^m||1||0, \mathbf{y}_i||0^m||r_i||0\rangle]\!]_T \\
&= [\![\Sigma_{i\in\mathcal{U}_K}\langle \mathbf{x}_i, \mathbf{y}_i\rangle + \Sigma_{i\in\mathcal{U}_K} r_i]\!]_T \\
&= [\![\Sigma_{i\in\mathcal{U}_K}\langle \mathbf{x}_i, \mathbf{y}_i\rangle + z]\!]_T \\
&= [\![\Sigma_{i\in\mathcal{U}_K}\langle \mathbf{x}_i, \mathbf{y}_i\rangle]\!]_T \times [\![z]\!]_T
\end{aligned}$$

Hence, it follows that $v = \log([\![v]\!]_T / [\![z]\!]_T) = \sum_{i\in\mathcal{U}_K}\langle \mathbf{x}_i, \mathbf{y}_i\rangle$

Theorem 6.1.1 (Restatement of Theorem 1.1.2)**.** *Suppose that* IPE *satisfies selective, function-hiding IND-security and* DSum *satisfies selective-IND-security, then, the above* aMIIPE *scheme satisfies selective function-hiding IND-security.*

Proof. The proof is presented next in Section 6.2 □

6.2 Proof of Theorem 6.1.1

We consider a sequence of hybrid experiments. Denote by $\mathcal{H} = [n] \backslash \mathcal{K}$, the set of honest parties.

Experiment $\mathsf{aMIIPE\text{-}Expt}^1$. This is the real-world experiment, parameterized by $b = 1$. In the experiment $\mathsf{aMIIPE\text{-}Expt}^1$, the challenger $\mathcal{C}$ answers **Enc** and **KGen** queries using the following vectors/scalars:

$$\widetilde{\mathbf{x}}_i = (\mathbf{x}_i^{(1)}, 0^m, 1, 0), \quad \widetilde{\mathbf{y}}_i = (\mathbf{y}_i^{(1)}, 0^m, r_i, 0), \quad d_i = r_i$$

where r_i is freshly chosen for every **KGen** query and $i \in [n]$.

Experiment Hyb^*. Same as $\mathsf{aMIIPE\text{-}Expt}^1$ except that for any honest $i \in \mathcal{H}$, $d_{i_1} = \sum_{i \in \mathcal{H}} r_i$ and rest all of $d_{i_2}, ..., d_{i_{|\mathcal{H}|}} = 0$.

$\mathsf{aMIIPE\text{-}Expt}^1$ and Hyb^* are computationally indistinguishable by the selective-IND-security of the underlying DSum scheme as the decentralized sum function output is the same in both the hybrids for all sets of **KGen** queries.

Experiment Hyb_0. Same as Hyb_0 except that $\forall i \in \mathcal{H}, \widetilde{\mathbf{x}}_i = (\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, 1, 0)$.

Hyb^* and Hyb_0 are computationally indistinguishable by the function-hiding IND-security of the underlying IPE scheme since the inner products are the same across both the hybrids for any pair of encryption and key vectors queried and for $i \in \mathcal{H}$.

Experiment Hyb_ℓ. We next define a sequence of hybrid experiments. Hyb_ℓ for $\ell \in [Q_{\text{kgen}}]$, where Q_{kgen} denotes an upper bound the number of **KGen** queries made by $\mathcal{A}$. In Hyb_ℓ, for the first ℓ **KGen** queries, the challenger $\mathcal{C}$ uses $\widetilde{\mathbf{y}}_i = \left(0^m, \mathbf{y}_i^{(0)}, r_i, 0\right)$ for any honest $i \in \mathcal{H}$, and uses $\widetilde{\mathbf{y}}_i = \left(\mathbf{y}_i^{(1)}, 0^m, r_i, 0\right)$ for any corrupt $i \in \mathcal{K}$. For the remaining $Q_{\text{kgen}} - \ell$ number of **KGen** queries, $\mathcal{C}$ uses $\widetilde{\mathbf{y}}_i = \left(\mathbf{y}_i^{(1)}, 0^m, r_i, 0\right)$ for all $i \in [n]$.

We show in Lemma 6.2.1 that for all $\ell \in [Q_{\text{kgen}}]$, hybrids $\mathsf{Hyb}_{\ell-1}$ and Hyb_ℓ are computationally indistinguishable.

Experiment $\mathsf{Hyb}^{\#}$. The challenger $\mathcal{C}$ answers **Enc** and **KGen** queries using the following vectors/scalars for any honest $i \in \mathcal{H}$:

$$\widetilde{\mathbf{x}}_i = \left(0^m, \mathbf{x}_i^{(0)}, 1, 0\right), \quad \widetilde{\mathbf{y}}_i = \left(0^m, \mathbf{y}_i^{(0)}, r_i, 0\right), \quad d_i = \begin{cases} \sum_{j \in \mathcal{H}} r_j & \text{if } i = i_1 \in \mathcal{H} \\ 0 & \text{if } i \in \mathcal{H} \backslash \{i_1\} \end{cases}$$

For corrupt $i \in \mathcal{K}$, the challenger $\mathcal{C}$ still uses:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, 0^m, 1, 0\right), \quad \widetilde{\mathbf{y}}_i = \left(\mathbf{y}_i^{(1)}, 0^m, r_i, 0\right), \quad d_i = r_i$$

Observe that $\mathsf{Hyb}_{Q_{\text{kgen}}}$ and $\mathsf{Hyb}^{\#}$ are almost identical except that the first m coordinates in $\widetilde{\mathbf{x}}_i$ are replaced with 0^m for $i \in \mathcal{H}$. Since this modification preserves the inner products $\langle \widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}_i \rangle$ for any pair of encryption and key vectors queried, and for any $i \in \mathcal{H}$, $\mathsf{Hyb}^{\#}$ is computationally indistinguishable from $\mathsf{Hyb}_{Q_{\text{kgen}}}$ due to the function-hiding IND-security of the IPE scheme.

Experiment Hyb^{**}. Same as $\mathsf{Hyb}^{\#}$ except that for all $i \in \mathcal{H}$, d_i are restored back to r_i.

Observe that the decentralized sum function output is preserved across experiments $\mathsf{Hyb}^{\#}$ and Hyb^{**}. Hence, assuming the selective-IND-security of DSum scheme, the two experiments are computationally indistinguishable.

Experiment $\mathsf{aMIIPE\text{-}Expt}^0$. $\mathsf{aMIIPE\text{-}Expt}^0$ differs from Hyb^{**} only in the slot in which $\mathbf{x}_i^{(0)}$ and $\mathbf{y}_i^{(0)}$ are put in $\widetilde{\mathbf{x}}$ and $\widetilde{\mathbf{y}}$ respectively for all honest parties. Since for honest $i \in \mathcal{H}$, the inner-product $\langle \widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}_i \rangle$ is preserved for any pair of encryption and key vectors queried; and for corrupt $i \in \mathcal{K}$, recall that our admissibility stipulates that $\mathbf{x}_i^{(0)} = \mathbf{x}_i^{(1)}$ and $\mathbf{y}_i^{(0)} = \mathbf{y}_i^{(1)}$, and thus it makes no difference whether $\mathbf{x}_i^{(0)}, \mathbf{y}_i^{(0)}$ is used or whether $\mathbf{x}_i^{(1)}, \mathbf{y}_i^{(1)}$ is used by $\mathcal{C}$, therefore Hyb^{**} is computationally indistinguishable from $\mathsf{aMIIPE\text{-}Expt}^0$.

Therefore, to complete the proof of Theorem 6.1.1, it suffices to prove the following lemma, which shows the computational indistinguishability of $\mathsf{Hyb}_{\ell-1}$ and Hyb_ℓ.

Lemma 6.2.1. *Suppose that* IPE *satsifies function-hiding IND-security. Then,* $\mathsf{Hyb}_{\ell-1}$ *is computationally indistinguishable from* Hyb_ℓ *for any* $\ell \in [Q_{\mathrm{kgen}}]$.

Proof. Note that going from $\mathsf{Hyb}_{\ell-1}$ to Hyb_ℓ, the only difference is in how $\widetilde{\mathbf{y}}_i$ are chosen for all $i \in \mathcal{H}$ for answering ℓ-th **KGen** query. Hence, we consider a sequence of hybrid experiments to make this transition.

Experiment $\mathsf{H}_{\ell-1,1}$. In experiment $\mathsf{H}_{\ell-1,1}$, for any honest $i \in \mathcal{H}$, the challenger $\mathcal{C}$ uses the following vectors/scalars to answer the **Enc** and **KGen** queries where $r_i^* \leftarrow \mathbb{Z}_q^k$ denotes the randomness chosen during ℓ-th **KGen** query and we use $\mathbf{y}^{*(0)}$, $\mathbf{y}^{*(1)}$ to denote the key vectors submitted during the ℓ-th **KGen** query:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, 1, r_i^* + \langle \mathbf{x}_i^{(1)}, \mathbf{y}_i^{*(1)} \rangle\right),$$

$$\widetilde{\mathbf{y}}_i = \begin{cases} \left(0^m, \mathbf{y}_i^{(0)}, r_i, 0\right) & \text{first } \ell - 1 \textbf{ KGen} \text{ queries} \\ (0^m, 0^m, 0, 1) & \ell\text{-th } \textbf{KGen} \text{ query} \\ \left(\mathbf{y}_i^{(1)}, 0^m, r_i, 0\right) & \text{remaining } Q_{\mathrm{kgen}} - \ell \textbf{ KGen} \text{ queries} \end{cases},$$

$$d_{i_1} = \begin{cases} \sum_{j \in \mathcal{H}} r_j^* & \ell\text{-th } \textbf{KGen} \text{ query} \\ \sum_{j \in \mathcal{H}} r_j & \text{otherwise} \end{cases}, d_{i_2} = 0, ..., d_{i_{\mathcal{H}}} = 0$$

In the above, r_i are freshly chosen for every **KGen** query, and r_i^* corresponds to the randomness chosen for the challenge **KGen** query, i.e., the ℓ-th **KGen** query.

Observe that $\mathsf{H}_{\ell-1,1}$ is almost identical to $\mathsf{Hyb}_{\ell-1}$ except for the above modifications highlighted in blue. Since these modification preserves the inner products $\langle \widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}_i \rangle$ for any pair of encryption and key vectors queried, and for any $i \in \mathcal{H}$, $\mathsf{H}_{\ell-1,1}$ and $\mathsf{Hyb}_{\ell-1}$ are computationally indistinguishable due to the function-hiding IND-security of the IPE scheme.

Experiment $\mathsf{H}'_{\ell-1,1}$. Almost identical to $\mathsf{H}_{\ell-1,1}$ except that the challenger $\mathcal{C}$ chooses the following vectors to answer **Enc** queries for an honest $i \in \mathcal{H}$:

$$\widetilde{\mathbf{x}}_i = \left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(0)}, 1, r_i^* + \langle \mathbf{x}_i^{(0)}, \mathbf{y}_i^{*(0)} \rangle \right),$$

In the above, r_i^* corresponds to the randomness chosen for the challenge **KGen** query, i.e., the ℓ-th **KGen** query.

As long as $\mathcal{A}$ respects the admissibility rule defined in Section 3.2, $\mathsf{H}_{\ell-1,1}$ and $\mathsf{H}'_{\ell-1,1}$ are identically distributed.

Finally, $\mathsf{H}'_{\ell-1,1}$ and Hyb_{ℓ} are computationally indistinguishable due to the function-hiding IND-security of the IPE scheme, since the inner-product $\langle \widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}_i \rangle$ is preserved for any pair of encryption and key vectors queried and for $i \in \mathcal{H}$. This concludes the proof of Lemma 6.2.1.

□

Bibliography

[ABG19] Michel Abdalla, Fabrice Benhamouda, and Romain Gay. From single-input to multi-client inner-product functional encryption. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 552–582. Springer, 2019. [Accessed 2021-12-06].

[ABM+20] Michel Abdalla, Florian Bourse, Hugo Marival, David Pointcheval, Azam Soleimanian, and Hendrik Waldner. Multi-client inner-product functional encryption in the random-oracle model. In *International Conference on Security and Cryptography for Networks*, pages 525–545. Springer, 2020. [Accessed 2021-12-06].

[ACF+18] Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 597–627. Springer, 2018. [Accessed 2021-12-06].

[ACF+20] Shweta Agrawal, Michael Clear, Ophir Frieder, Sanjam Garg, Adam O'Neill, and Justin Thaler. Ad hoc multi-input functional encryption. In *ITCS*, 2020. [Accessed 2021-12-06].

[AGRW17] Michel Abdalla, Romain Gay, Mariana Raykova, and Hoeteck Wee. Multi-input inner-product functional encryption from pairings. In *Advances in Cryptology – EUROCRYPT 2017*, pages 601–626, 2017. [Accessed 2021-12-06].

[AGT21a] Shweta Agrawal, Rishab Goyal, and Junichi Tomida. Multi-input quadratic functional encryption from pairings. In *Annual International Cryptology Conference*, pages 208–238. Springer, 2021. [Accessed 2021-12-06].

[AGT21b] Shweta Agrawal, Rishab Goyal, and Junichi Tomida. Multi-party functional encryption. In *Theory of Cryptography Conference*, pages 224–255. Springer, 2021. [Accessed 2021-12-06].

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 2004. [Accessed 2021-12-06].

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001. [Accessed 2021-12-06].

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1175–1191, New York, NY, USA, 2017. [Accessed 2021-12-06].

[CDG+18] Jeremy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In *Advances in Cryptology – ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 703–732. Springer, 2018. [Accessed 2021-12-06].

[CDSG+20] Jeremy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Dynamic decentralized functional encryption. In *Annual International Cryptology Conference*, pages 747–775. Springer, 2020. [Accessed 2021-12-06].

[Gen06] Craig Gentry. Practical identity-based encryption without random oracles. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 445–464. Springer, 2006. [Accessed 2021-12-06].

[GGG+14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Eurocrypt*, volume 8441, pages 578–602, 2014. [Accessed 2021-12-06].

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2013. [Accessed 2021-12-06].

[GGJS13] Shafi Goldwasser, Vipul Goyal, Abhishek Jain, and Amit Sahai. Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/727, 2013. https://eprint.iacr.org/2013/727, [Accessed 2021-12-06].

[GKL+13] S. Dov Gordon, Jonathan Katz, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/774, 2013. https://eprint.iacr.org/2013/774, [Accessed 2021-12-06].

[Lin17] Huijia Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 prgs. In *Advances in Cryptology - CRYPTO, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 599–629. Springer, 2017. [Accessed 2021-12-06].

[LT19] Benoit Libert and Radu Titiu. Multi-client functional encryption for linear functions in the standard model from LWE. In *Advances in Cryptology – ASIACRYPT 2019*, volume 11923, pages 520–551. Springer, 2019. [Accessed 2021-12-06].

[SCR+11] Elaine Shi, T-H. Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Network and Distributed System Security Symposium (NDSS)*, 2011. [Accessed 2021-12-06].

[SW21] Elaine Shi and Ke Wu. Non-interactive anonymous router. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 489–520. Springer, 2021. [Accessed 2021-12-06].

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 114–127. Springer, 2005. [Accessed 2021-12-06].

[Wee16] Hoeteck Wee. New techniques for attribute-hiding in prime-order bilinear groups. Manuscript, 2016. [Accessed 2021-12-06].

Appendix A

Efficient All-Or-Nothing Encryption without Random Oracles

A.1 Definition: All or Nothing Encryption

All-or-nothing encryption (AoNE) was defined by Chotard et al. [CDSG+20]. Intuitively, we want to guarantee that if one collects all ciphertexts from all users in a set $\mathcal{U}$, encrypted to the same $(\mathcal{U}, t)$ pair, then everyone's plaintext message can be decrypted. However, if one does not have the collection of all ciphertexts from all users in $\mathcal{U}$, then, nothing is leaked about the plaintexts.

Formally, an all-or-nothing encryption scheme, parametrized by a plaintext length m which is a polynomially bounded function of the security parameter λ, consists of the following possibly randomized algorithms:

- $\mathsf{app} \leftarrow \mathbf{Gen}(1^\lambda, n, m)$: takes a security parameter 1^λ, the space size of user identities n, the length of the messages m, and outputs the public parameters app;
- $(\mathsf{aPK}_i, \mathsf{aSK}_i) \leftarrow \mathbf{USetup}(\mathsf{app})$: takes in the public parameters app, and outputs a public key and secret key pair for a user, denoted aPK_i and aSK_i, respectively;
- $\mathsf{ct}_i \leftarrow \mathbf{Enc}(\mathsf{app}, \mathsf{aSK}_i, \mathsf{msg} = (\mathbf{x}, \mathcal{U}, t), \{\mathsf{aPK}_j\}_{j\in\mathcal{U}})$: takes in the public parameters app a user's secret key aSK_i, and a message $\mathsf{msg} = (\mathbf{x}, \mathcal{U}, t) \in \mathcal{M}$ which consists of a private component $\mathbf{x}$, a set of users $\mathcal{U}$ and a label t, and a set of public keys $\{\mathsf{aPK}_j\}_{j\in\mathcal{U}}$ for users

in $\mathcal{U}$; and outputs a ciphertext ct_i.

- $v \leftarrow \mathbf{Dec}(\mathsf{app}, \{\mathsf{ct}_i\}_{i\in\mathcal{U}})$: takes in public parameters app, and a set of ciphertexts $\{\mathsf{ct}_i\}_{i\in\mathcal{U}}$, and outputs a decrypted value $v = \{\mathbf{x}_i\}_{i\in\mathcal{U}}$; if decryption fails, output $v = \perp$.

Correctness. Correctness is defined in the most natural way: for any $\lambda \in \mathbb{N}$, any $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \{0,1\}^m$, and any $t \in \{0,1\}^*$,

$$\Pr\left[\begin{array}{l} \mathsf{app} \leftarrow \mathbf{Gen}(1^\lambda, n, m) \\ \forall i \in [n] : (\mathsf{aPK}_i, \mathsf{aSK}_i) \leftarrow \mathbf{USetup}(\mathsf{app}) \\ \forall i \in \mathcal{U} : \mathsf{ct}_i \leftarrow \mathbf{Enc}(\mathsf{app}, \mathsf{aSK}_i, \mathsf{msg} = (\mathbf{x}_i, \mathcal{U}, t), \{\mathsf{aPK}_j\}_{j\in\mathcal{U}}) \\ \{\mathbf{x}' m_j\}_{j\in\mathcal{U}} \leftarrow \mathbf{Dec}(\mathsf{app}, \{\mathsf{ct}_j\}_{j\in\mathcal{U}}) \end{array} : \forall i \in [n] : \mathbf{x}'_i = \mathbf{x}_i \right] = 1$$

Definition 4 (IND-security of AoNE)**.** We say that an all-or-nothing encryption scheme is IND-secure, if for any non-uniform PPT *admissible* adversary $\mathcal{A}$, the following two experiments $\mathsf{AoNExpt}^0$ and $\mathsf{AoNExpt}^1$ are computationally indistinguishable, where $\mathsf{AoNExpt}^b$ for $b \in \{0,1\}$ is defined as follows:

- **Setup.** The challenger $\mathcal{C}$ runs $\mathsf{app}, \leftarrow \mathbf{Setup}(1^\lambda)$, $\forall i \in [n] : (\mathsf{aPK}_i, \mathsf{aSK}_i) \leftarrow \mathbf{USetup}(\mathsf{app})$ gives $\mathsf{app}, \{\mathsf{aPK}_i\}_{i\in[n]}$ to the adversary $\mathcal{A}$, and receives the corrupted set $\mathcal{K} \subset [n]$ from the adversary $\mathcal{A}$. The challenger $\mathcal{C}$ now gives the corrupted keys $\{\mathsf{aSK}_i\}_{i\in\mathcal{K}}$ to $\mathcal{A}$.
- **Query.** The adversary $\mathcal{A}$ can adaptively submit encryption queries of the following form $(i, \mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathcal{U}, t)$, and the challenger computes $\mathsf{ct}_i \leftarrow \mathbf{Enc}(\mathsf{app}, \mathsf{aSK}_i, \mathsf{msg} = (\mathbf{x}^{(b)}, \mathcal{U}, t), \{\mathsf{aPK}_j\}_{j\in\mathcal{U}})$ and returns ct_i to $\mathcal{A}$.

We say that the adversary $\mathcal{A}$ is *admissible* iff the following conditions hold with probability 1:

- For any encryption query $(i, \mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathcal{U}, t)$ pertaining to a corrupted user i, it must be that $\mathbf{x}^{(0)} = \mathbf{x}^{(1)}$.
- For any label t for which the adversary $\mathcal{A}$ has submitted an encryption query for every honest user, it must be that for every encryption query of the form $(i, \mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathcal{U}, t)$ pertaining to this label t, $\mathbf{x}^{(0)} = \mathbf{x}^{(1)}$.

Definition 5 (Selective-IND-security of AoNE)**.** We say that an all-or-nothing encryption scheme is selective-IND-secure if in definition 4, the adversary makes all the encryption queries at once instead of adaptively.

A.2 Construction

The efficient construction of AoNE by Chotard et al. [CDSG+20] crucially relied on random oracles. Our construction follows their blueprint of using IBE as a building block but gets rid of random oracles. The root cause of needing random oracles in their construction was that they used the Boneh-Franklin IBE [BF01]. Subsequent constructions of IBE [BB04, Wat05, Gen06] have successfully managed to get rid of random oracles. The construction by Gentry [Gen06] is from non-standard assumptions. Among the other two from standard assumptions, the construction by Boneh and Boyen [BB04] has shorter public parameters but only achieves selective-ID IND-security, whereas the construction by Waters [Wat05] has larger public parameters and is fully secure. For our purposes, selective IND-secure AoNE is sufficient as our aMIIPE construction achieves selective function-hiding IND-security. To construct selective IND-secure IBE, selective-ID IND-secure IBE is sufficient. Hence, we will use Boneh and Boyen's [BB04] IBE construction to construct our AoNE scheme.

Let $\mathbb{G}$ be a bilinear group of prime order q and let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map. Deonte a generator of group $\mathbb{G}$ by g and let $\mathcal{PG}$ denote an algorithm which chooses all these values. For all set of users $\mathcal{U} \subseteq [n]$ and a label space $\mathcal{T}$, let $\mathcal{U} \times \mathcal{T} \subseteq \mathbb{Z}_q$. Denote by F a publicly computable function $F : \mathbb{Z}_q \to \mathbb{G}$. For this construction, F is defined as $F_i(ID) = k(g^{\alpha_i})^{ID}$ for i^{th}-user and $F_{\mathcal{U}}(ID) = k(\prod_{j \in \mathcal{U}} g^{\alpha_j})^{ID}$ for a set of users $\mathcal{U}$, where k is a random element of the group $\mathbb{G}$ generated as part of the public params, g^{α_i} is part of the public key of the i^{th}-user. Lastly, denote by SKE a symmetric key encryption scheme which is one-time secure.

Efficient all-or-nothing encryption without random oracles

- **Gen**$(1^\lambda, n, m)$:

- let $(G, G_T, q, e, g) \leftarrow \mathcal{PG}(1^\lambda)$
- let $h, k \xleftarrow{\$} G$
- output $\mathsf{app} = (G, G_T, q, e, g, h, k)$.

- **USetup**(app):
  - Sample $\alpha_i \xleftarrow{\$} \mathbb{Z}_q$
  - output $\mathsf{aPK}_i = g^{\alpha_i}, \mathsf{aSK}_i = h^{\alpha_i}$
- **Enc**($\mathsf{app}, \mathsf{aSK}_i, m_i = (\mathbf{x}_i, \mathcal{U}, t), \{\mathsf{aPK}_j\}_{j \in \mathcal{U}}$):
  - Sample $\rho_i \xleftarrow{\$} \mathbb{Z}_q$ and compute $A_1^i = e(\prod_{j \in \mathcal{U}} g^{\alpha_j}, h)^{\rho_i}, A_2^i = g^{\rho_i}, A_3^i = F_{\mathcal{U}}(\mathcal{U}||t)^{\rho_i}$
  - Sample $r_i \xleftarrow{\$} \mathbb{Z}_q$ and compute $B_1^i = h^{\alpha_i} \cdot F_{\mathcal{U}}(\mathcal{U}||t)^{r_i}, B_2^i = g^{r_i}$
  - Compute the symmetric key as $K_{i,\mathcal{U},t} = A_1^i$ and encrypt x_i using it as $\overline{ct_i} \leftarrow \mathsf{SKE}.\mathbf{Enc}(K_{i,\mathcal{U},t}, x_i)$
  - Compute a share of the decryption key as $S_{i,\mathcal{U},t} = (A_2^i, A_3^i, B_1^i, B_2^i)$
  - output $\mathsf{ct}_i = (\overline{ct_i}, S_{i,\mathcal{U},t}, \mathcal{U}, t)$
- **Dec**($\mathsf{app}, \{\mathsf{ct}_i\}_{i \in \mathcal{U}}$):
  - Parse the ciphertexts for all $i \in \mathcal{U}$ as $\mathsf{ct}_i = (\overline{ct_i}, S_{i,\mathcal{U},t}, \mathcal{U}, t)$, where $S_{i,\mathcal{U},t} = (A_2^i, A_3^i, B_1^i, B_2^i)$
  - Compute $B_1 = \prod_{j \in \mathcal{U}} B_1^j$ and $B_2 = \prod_{j \in \mathcal{U}} B_2^j$
  - $\forall i \in \mathcal{U}$, recover $A_1^i = \frac{e(A_2^i, B_1)}{e(A_3^i, B_2)}$ and then recover $x_i \leftarrow \mathsf{SKE}.\mathbf{Dec}(K_{i,\mathcal{U},t} = A_1^i, \overline{ct_i})$
  - output $\{x_i\}_{i \in \mathcal{U}}$

A.3 Correctness

If the decryption algorithm can recover A_1^i's correctly, then, by the correctness of the symmetric key encryption scheme, our scheme's output must be correct. Therefore, we just need to show the A_1^i's are recoverred correctly. If all the users correctly run the algorithms **Gen**,

USetup and **Enc**, then, observe that -

$$\begin{aligned}\frac{e(A_2^i, B_1)}{e(A_3^i, B_2)} &= \frac{e(g^{\rho_i}, h^{\Sigma\alpha_i} \cdot F_{\mathcal{U}}(\mathcal{U}||t)^{\Sigma r_i})}{e(F_{\mathcal{U}}(\mathcal{U}||t)^{\rho_i}, g^{\Sigma r_i})} \\ &= \frac{e(g^{\rho_i}, h^{\Sigma\alpha_i}) \cdot e(g^{\rho_i}, F_{\mathcal{U}}(\mathcal{U}||t)^{\Sigma r_i})}{e(F_{\mathcal{U}}(\mathcal{U}||t)^{\rho_i}, g^{\Sigma r_i})} \\ &= e(g^{\rho_i}, h^{\Sigma\alpha_i}) \\ &= e(g^{\Sigma\alpha_i}, h)^{\rho_i} \\ &= A_1^i\end{aligned}$$

A.4 Security

Theorem A.4.1. *Suppose that q_r-fold DBDH assumption holds true for some fixed q_r, then, the above* AoNE *scheme satisfies sel-IND-security.*

Proof. Let q_e denote the number of unique IDs $(\mathcal{U}, t)$ for which the adversary sends atleast one encryption query and let q_r deonte the maximum number of encryption queries for any such unique ID. Then, we prove security via a hybrid argument. For $i \in \{1, ..., q_e\}$, the hybri experiments are as follows -

Experiment $\mathsf{Hyb}_{i,0}$. The challenger plays the game as defined in Section A.1 except the response to encryption queries for $(m_0^j = (x_0, \mathcal{U}^j, t^j), m_1^j = (x_1, \mathcal{U}^j, t^j))$ changes as follows - for all $j \geqslant i$, the challenger encrypts m_0^j and for all $j < i$, the challenger encrypts m_1^j.

Experiment $\mathsf{Hyb}_{i,1}$. This experiment is similiar to $\mathsf{Hyb}_{i,0}$, except that in all the encryption queries for the i^{th} unique ID $(\mathcal{U}^i, t^i)$, the challenger uses an ephemeral random value K to compute $\overline{ct_i} \leftarrow \mathsf{SKE.Enc}(K, x_0)$ instead of using $K_{i,\mathcal{U}^i,t^i}$. Here, x_0, x_1 are the part of messages m_0^i, m_1^i respectively sent by the adversary. Note that the adversary can send multiple encryptions queries for the same unique ID $(\mathcal{U}^i, t^i)$ and the keys K are ephemeral in the sense that they are sampled freshly and randomly for each such encryption query.

Experiment $\mathsf{Hyb}_{i,2}$. This experiment is similiar to $\mathsf{Hyb}_{i,1}$, except that in all the encryption queries for the i^{th} unique ID $(\mathcal{U}^i, t^i)$, the challenger encrypts x_1 instead of x_0. That is, it computes $\overline{ct_i} \leftarrow \mathsf{SKE}.\mathbf{Enc}(K, x_1)$ where x_0, x_1 were the part of messages m_0^i, m_1^i respectively sent by the adversary.

Experiment $\mathsf{Hyb}_{i,3}$. This experiment is similar to $\mathsf{Hyb}_{i,2}$, except that in all the encryption queries for the i^{th} unique ID $(\mathcal{U}^i, t^i)$, the challenger restores back to using K as in the original construction. That is, it encrypts $\overline{ct_i} \leftarrow \mathsf{SKE}.\mathbf{Enc}(K_{i,\mathcal{U}^i,t^i}, x_1)$.

The sequence of experiments we follow is $\mathsf{Hyb}_{1,0}, ..., \mathsf{Hyb}_{1,3}, \mathsf{Hyb}_{2,0}, ...\mathsf{Hyb}_{2,3}, \mathsf{Hyb}_{3,0}, ..., \mathsf{Hyb}_{q_e,3}$. Note that in experiment $\mathsf{Hyb}_{1,0}$, the challenger always encrypts m_0^j and hence this is same as $\mathsf{AoNExpt}^0$. In experiment $\mathsf{Hyb}_{q_e,1}$, the challenger always encrypts m_1^j and hence this is same as $\mathsf{AoNExpt}^1$. Further, note that $\mathsf{Hyb}_{i,3} = \mathsf{Hyb}_{i+1,0}$ for all $i \in [q_e - 1]$. Therefore, to prove that $\mathsf{Hyb}_{1,0}$ and $\mathsf{Hyb}_{q_e,3}$ are computationally indistinguishable, we need to show that for all $i \in [q_e]$: $\mathsf{Hyb}_{i,0} \approx_c \mathsf{Hyb}_{i,1}$, $\mathsf{Hyb}_{i,1} \approx_c \mathsf{Hyb}_{i,2}$, $\mathsf{Hyb}_{i,2} \approx_c \mathsf{Hyb}_{i,3}$.

Observe that the one-time security of the symmetric key encryption directly implies that $\mathsf{Hyb}_{i,1} \approx_c \mathsf{Hyb}_{i,2}$. Further, the other two experiment transitions involve switching between honestly generated symmetric key $K_{i,\mathcal{U}^i,t^i}$ and randomly sampled symmetric key K. Hence, we just need to show that $\mathsf{Hyb}_{i,0} \approx_c \mathsf{Hyb}_{i,1}$ and by similar argument it would follow that $\mathsf{Hyb}_{i,2} \approx_c \mathsf{Hyb}_{i,3}$.

$\mathsf{Hyb}_{i,0} \approx_c \mathsf{Hyb}_{i,1}$. We prove this by contradiction. We start with an adversary $\mathcal{A}$ which can distinguish between $\mathsf{Hyb}_{i,0}$ and $\mathsf{Hyb}_{i,1}$ with noticeable advantage and construct an adversary $\mathcal{B}$ which breaks the DBDH assumption with noticeable advantage. The reduction is as follows. Note that the two hybrids differ in how the symmetric keys are generated while responding to encryption queries for the i^{th} unique ID $(\mathcal{U}^i, t^i)$.

Reduction. The DBDH challenger generates the pairing groups $(G, G_T, q, e, g) \leftarrow \mathcal{PG}(1^\lambda)$ and the challenge tuple $(g, g_1 = g^a, g_2 = g^b, \{g_{3,i} = g^{c_i}, T_i\}_{i\in[q_r]})$ where $a, b, c_1, ..., c_{q_r} \xleftarrow{\$} \mathbb{Z}_q$,

$\beta \xleftarrow{\$} \{0,1\}$ and if $\beta = 0$, then, $T_i = e(g,g)^{abc_i}$, else $T_i \xleftarrow{\$} G_T$. It sends all this to $\mathcal{B}$ which needs to guess β correctly to win the game. Also, $\mathcal{A}$ sends a set of corrupt parties $\mathcal{K} \subset [n]$ to $\mathcal{B}$ and all the encryption queries of the form $(m_0^j = (x_0, \mathcal{U}, t), m_1^j = (x_1, \mathcal{U}, t))$ to $\mathcal{B}$. $\mathcal{B}$ needs to first send back everyone's public keys and corrupt parties' master secret keys back to $\mathcal{A}$, and then also send back replies to encryption queries.

$\mathcal{B}$ will now embed this q_r-fold DBDH challenge carefully in the game against $\mathcal{A}$. g_1 will be embedded as part of the public key for some honest party z as $\mathsf{pk}_z = g^{\alpha_z} := g_1$ (we postpone the discussion of this choice of honest party to after the reduction as the reduction would be insighful for it), g_2 will be part of the public params as $h = g_2$, $g_{3,i}$'s will serve as randomness in encryption and T_i's will be used for selecting the symmetric keys for the honest party chosen above.

$\mathcal{B}$ starts by setting $ID^* = (\mathcal{U}^i || t^i)$. It then samples $\delta \xleftarrow{\$} \mathbb{Z}_q$, sets $k = g^\delta / g_1^{ID^*}, h = g_2$ and $pp = (G, G_T, q, e, g, h, k)$. Further, $\forall j \in [n] \backslash \{z\}$: $\mathcal{B}$ samples $\alpha_j \xleftarrow{\$} \mathbb{Z}_q$ and sets $\mathsf{pk}_j = g^{\alpha_j}, msk_j = h^{\alpha_j}$. For the honest party z, it sets $\mathsf{pk}_z = g^{\alpha_z} := g_1$. According to this, msk_z is supposed to be g^{ab} which $\mathcal{B}$ doesn't know, else it will be able to trivially break the DBDH challenged by computing $e(g^{ab}, g^c)$. It sends the public keys of all users $\{\mathsf{pk}_j\}_{j \in [n]}$ and the master secret keys of corrupt users $\{msk_j\}_{j \in \mathcal{K}}$ to $\mathcal{A}$.

In AoNE, there are no **KGen** queries, so we need to only show how does $\mathcal{B}$ simulate $\mathcal{A}$'s **Enc** queries. For an **Enc** query of the form $(m_0^{j,s} = (x_0^s, \mathcal{U}^j, t^j), m_1^{j,s} = (x_1^s, \mathcal{U}^j, t^j))$, where $(\mathcal{U}^j, t^j)$ is the j^{th} unique ID, and it is the s^{th} query for this ID, $\mathcal{B}$ simulates the response as follows.

- Choose $K^s_{j,\mathcal{U}^j,t^j} = A_1^{j,s} = T_s \cdot e(g_{3,s}, \prod_{\mathsf{pk}_j \in \mathcal{U} \backslash \{z\}} h^{\alpha_j})$, $A_2^{j,s} = g_{3,s}$, $A_3^{j,s} = g_{3,s}^\delta$
- Compute $\overline{ct_{j,s}} \leftarrow \mathsf{SKE}.\mathbf{Enc}(K^s_{j,\mathcal{U}^j,t^j}, x_0^s)$
- Sample $r_{j,s} \xleftarrow{\$} \mathbb{Z}_q$ and if it is the z^{th} party, then, compute $B_1^{j,s} = g_2^{-\delta/(ID - ID^*)} \cdot F_{\mathcal{U}}(ID)^{r_{j,s}}$, and $B_2^{j,s} = g_2^{-1/(ID-ID^*)} \cdot g^{r_{j,s}}$, else, for any other party w, compute $B_1^{j,s} = h^{\alpha_w} \cdot F_{\mathcal{U}}(ID)^{r_{j,s}}$, and $B_2^{j,s} = g^{r_{j,s}}$. Here, $ID = (\mathcal{U}^j || t^j)$

- Compute the share of the decryption key $S^s_{j,\mathcal{U}^j,t^j} = (A_2^{j,s}, A_3^{j,s}, B_1^{j,s}, B_2^{j,s})$
- Send back the output $\mathsf{ct}_{j,s} = (\overline{ct_{j,s}}, S^s_{j,\mathcal{U}^j,t^j}, \mathcal{U}^j, t^j)$ to $\mathcal{A}$

After sending back all the responses, $\mathcal{B}$ receives a guess $\beta' \in \{0,1\}$ from $\mathcal{A}$ indicating that $\mathcal{A}$ thinks that the distribution belonged to $\mathsf{Hyb}_{i,\beta'}$. $\mathcal{B}$ forwards this guess to the q_r-fold DBDH challenger. $\mathcal{B}$ wins the game if $\beta = \beta'$.

Choice of honest party z. Observe that in the above simulation, $\mathcal{B}$ can't simulate values $B_1^{j,s}, B_2^{j,s}$ for the honest party z when $ID = ID^*$. To get around this hurdle, we use an observation made by [CDSG+20] in their AoNE security proof that there must exist a honest party which is not queried on $ID = ID^*$. This is because, for $\mathcal{A}$ to distinguish between $\mathsf{Hyb}_{i,0}$ and $\mathsf{Hyb}_{i,1}$, $\mathcal{A}$ must make an encryption query with $x_0^s \neq x_1^s$ with noticeable probability, and conditioned on this event, $\mathcal{A}$ retains noticeable advantage. Moreover, if it were the case that $\mathcal{A}$ queries every party in U^i, then, the last admissibility condition in the security game gets violated as $x_0^s \neq x_1^s$ for some party, resulting in the $\mathcal{B}$'s guess to be set to a random value, thus making $\mathcal{A}$'s efforts useless. Thus, it is safe to assume that there exists $\mathsf{pk}_z \in \mathcal{U}^j$ which in not queried at all for ID^*. Thus, $\mathcal{B}$ randomly guesses an honest party $z \xleftarrow{\$} [n]$ which is not queried for ID^*. If $\mathcal{B}$'s guess turns out to be wrong, it aborts the protocol with $\mathcal{A}$ and sends a random guess to the DBDH challenger. This results in a polynomial loss of security in the reduction.

Reduction argument. We argue that when $\beta = 0$, the response to encryption queries have the same distribution as in $\mathsf{Hyb}_{i,0}$. This is because, our challenge embedding results in $e(g^{\alpha_z}, h)^{\rho_i} = e(g^a, g^b)^{c_s} = e(g,g)^{abc_s}$ which is exactly what T_s is when $\beta = 0$. When $\beta = 1$, the distribution is same as in $\mathsf{Hyb}_{i,1}$ as $T_s \xleftarrow{\$} \mathbb{G}$ makes the symmetric key look uniformly random. Hence, if $\mathcal{A}$ can distinguish $\mathsf{Hyb}_{i,0}, \mathsf{Hyb}_{i,1}$ with noticeable probability, then, $\mathcal{B}$, can break the DBDH assumption with noticeable probabiity.

□

Appendix B

MCIPE: Removing the All-or-Nothing Admissibility Rule

Recall that in our definition of MCIPE earlier in Section 3.1, the second admissibility rule is of an "all or nothing" nature: it requires that if the adversary queries one ciphertext for a label t, then it must query all honest clients' ciphertexts for the same label. This all-or-nothing admissibility rule can be lifted if we require that the adversary be "strongly selective", i.e., it is required to submit all **Enc** and **KGen** queries in one shot upfront in the security game. The transformation was described by Chotard et al. [CDSG+20] but their scheme is not function-hiding. For completeness, we explicitly describe this transformation, and show that it works for function-hiding MCIPE too.

We will use all-or-nothing encryption as a blackbox. In MCIPE, as the set of users is not dynamic, hence, we will use the following simplified notation of AoNE.

- $\mathsf{app}, \mathsf{aSK}_1, \ldots, \mathsf{aSK}_n \leftarrow \mathbf{Setup}(1^\lambda)$: the **Setup** algorithm takes in a security parameter 1^λ and outputs n client encryption keys denoted $\mathsf{aSK}_1, \ldots, \mathsf{aSK}_n$, respectively, and the public parameters denoted app.
- $\mathsf{ct} \leftarrow \mathbf{Enc}(\mathsf{app}, \mathsf{aSK}_i, x, t)$: takes in the public parameters app, a client secret key aSK_i, a plaintext message $x \in \{0,1\}^\ell$, and a label $t \in \{0,1\}^*$, outputs a ciphertext ct.

- $x_1, \ldots, x_n \leftarrow \mathbf{Dec}(\mathsf{app}, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$: given n ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ gathered from all clients and encrypted to the same label, output the decrypted messages $x_1, \ldots, x_n$.

B.1 Removing the All-or-Nothing Admissibility Rule

For simplicity, henceforth, we say that an MCIPE scheme is *strongly selective, function-hiding AoN-IND-secure* iff it satisfies satisfies our function-hiding IND-security notion defined earlier in Definition 1 of Section 3.1, and moreover, the adversary must now submit all **KGen** and **Enc** queries in one shot. We say that an MCIPE scheme is *strongly selective, function-hiding IND-secure* if it satisfies the same security notion as above except that the adversary is no longer required to respect the second admissibility rule.

We now show how to upgrade an MCIPE scheme that is strongly selective, function-hiding AoN-IND-secure (henceforth denoted MCIPE) to one that is strongly selective, function-hiding IND-secure (henceforth denoted MCIPE^*). The idea is quite simple: simply wrap the ciphertexts in another layer of all-or-nothing encryption. Henceforth, let $\mathsf{AoNE} := (\mathbf{Setup}, \mathbf{Enc}, \mathbf{Dec})$ denote an all-or-nothing encryption scheme.

- $\mathsf{MCIPE}^*.\mathbf{Gen}(1^\lambda)$: call $\mathsf{pp} \leftarrow \mathsf{MCIPE}.\mathbf{Gen}(1^\lambda)$ and output $\mathsf{pp}^* = \mathsf{pp}$.
- $\mathsf{MCIPE}^*.\mathbf{Setup}(\mathsf{pp}^*, m, n)$:
  1. call $(\mathsf{mpk}, \mathsf{msk}, \{\mathsf{ek}_i\}_{i\in[n]}) \leftarrow \mathsf{MCIPE}.\mathbf{Setup}(\mathsf{pp}, m, n)$,
  2. call $\mathsf{app}, \mathsf{aPK}_1, \ldots, \mathsf{aPK}_n \leftarrow \mathsf{AoNE}.\mathbf{Setup}(1^\lambda)$.
  3. Output $\mathsf{mpk}^* := (\mathsf{mpk}, \mathsf{app})$, $\mathsf{msk}^* = \mathsf{msk}$, and for $i \in [n]$, output $\mathsf{ek}_i^* := (\mathsf{ek}_i, \mathsf{aPK}_i)$.
- $\mathsf{MCIPE}^*.\mathbf{KGen}(\mathsf{mpk}^*, \mathsf{msk}^*, \mathbf{y})$: call $\mathsf{sk}_\mathbf{y}^* \leftarrow \mathsf{MCIPE}.\mathbf{KGen}(\mathsf{mpk}, \mathsf{msk}, \mathbf{y})$ and output $\mathsf{sk}_\mathbf{y}^* = \mathsf{sk}_\mathbf{y}$.
- $\mathsf{MCIPE}^*.\mathbf{Enc}(\mathsf{mpk}^*, \mathsf{ek}_i^*, \mathbf{x}_i, t)$:
  1. call $\mathsf{ct} \leftarrow \mathsf{MCIPE}.\mathbf{Enc}(\mathsf{mpk}, \mathsf{ek}_i, \mathbf{x}_i, t)$; and
  2. output $\mathsf{ct}^* := \mathsf{AoNE}.\mathbf{Enc}(\mathsf{aPK}_i, \mathsf{ct}, t)$.
- $\mathsf{MCIPE}^*.\mathbf{Dec}(\mathsf{mpk}^*, \mathsf{sk}_\mathbf{y}^*, \{\mathsf{ct}_{i,n}^*\}_{i\in[n]})$:

1. call $\mathsf{ct}_1, \ldots, \mathsf{ct}_n \leftarrow \mathsf{AoNE}.\mathbf{Dec}(\mathsf{app}, \mathsf{ct}_1^*, \ldots, \mathsf{ct}_n^*)$, and
2. output $\mathsf{MCIPE}.\mathbf{Dec}(\mathsf{mpk}, \mathsf{sk}_{\mathbf{y}}, \{\mathsf{ct}_{i,n}\}_{i\in[n]})$.

Theorem B.1.1. *Suppose that the underlying* MCIPE *scheme is strongly selective, function-hiding AoN-IND-secure and* AoNE *is a secure all-or-nothing encryption scheme, then the resulting* MCIPE^* *scheme is strongly selective, function-hiding IND-secure.*

Proof. We can consider a sequence of hybrid games.

AoNExp^0. This is the real security game for MCIPE^* where the challenger uses the bit $b = 0$, and was defined earlier.

Hyb^0. In Hyb^0, the challenger checks which labels t are "complete" w.r.t encryption queries. A label t is said to be complete if the adversary has submitted at least one encryption query pertaining to t for every honest client; otherwise, it is said to be incomplete. Now, for any incomplete label t, whenever the challenger is about to call $\mathsf{ct}^* \leftarrow \mathsf{AoNE}.\mathbf{Enc}(\mathsf{aPK}_i, \mathsf{ct}, t)$ for some honest client i, the challenger instead calls $\mathsf{ct}^* \leftarrow \mathsf{AoNE}.\mathbf{Enc}(\mathsf{aPK}_i, \mathbf{0}, t)$. Through a straightforward reduction, one can show that Hyb^0 is computationally indistinguishable from AoNExp^0 due to the security of AoNE.

Hyb^1. Same as Hyb^0 except that the challenger uses the bit $b = 1$ now. Through a straightforward reduction, one can show that Hyb^1 is computationally indistinguishable from Hyb^0 due to the selective, function-hiding AoN-IND-security of the underlying MCIPE scheme.

AoNExp^1. This is the real security game for MCIPE^* where the challenger uses the bit $b = 1$. Hyb^1 is computationally indistinguishable from AoNExp^1 for the same reason why Hyb^0 is computationally indistinguishable from AoNExp^0.

□