Verifiable Decentralized Multi-Client Functional Encryption for Inner Product

Dinh Duy Nguyen[1][0009−0002−7892−9146], Duong Hieu Phan[1][0000−0003−1136−4064], and David Pointcheval[2][0000−0002−6668−683X]

[1] LTCI, Telecom Paris, Institut Polytechnique de Paris, France
dinh.nguyen@telecom-paris.fr, hieu.phan@telecom-paris.fr
[2] DIENS, École normale supérieure, CNRS, Inria, PSL University, Paris, France
david.pointcheval@ens.fr

Abstract. Joint computation on encrypted data is becoming increasingly crucial with the rise of cloud computing. In recent years, the development of multi-client functional encryption (MCFE) has made it possible to perform joint computation on private inputs, without any interaction. Well-settled solutions for linear functions have become efficient and secure, but there is still a shortcoming: if one user inputs incorrect data, the output of the function might become meaningless for all other users (while still useful for the malicious user). To address this issue, the concept of verifiable functional encryption was introduced by Badrinarayanan *et al.* at Asiacrypt '16 (BGJS). However, their solution was impractical because of strong statistical requirements. More recently, Bell *et al.* introduced a related concept for secure aggregation, with their ACORN solution, but it requires multiple rounds of interactions between users. In this paper,

– we first propose a computational definition of verifiability for MCFE. Our notion covers the computational version of BGJS and extends it to handle any valid inputs defined by predicates. The BGJS notion corresponds to the particular case of a fixed predicate in our setting;
– we then introduce a new technique called *Combine-then-Descend*, which relies on the class group. It allows us to construct One-time Decentralized Sum (ODSUM) on verifiable private inputs. ODSUM is the building block for our final protocol of a verifiable decentralized MCFE for inner-product, where the inputs are within a range. Our approach notably enables the efficient identification of malicious users, thereby addressing an unsolved problem in ACORN.

Keywords: Verifiability · Decentralized · Functional Encryption · Inner Product

1 Introduction

Multi-Client Functional Encryption. Functional Encryption (FE) [BSW11] is a paradigm designed to overcome the *all-or-nothing* limitation of traditional encryption, allowing the sender to control access to their encrypted data in a more fine-grained manner through *functional decryption keys*. This paradigm enables the preservation of user's privacy in cloud computing services, where clouds can learn nothing beyond the delegated function evaluated on user's private data. FE with a single user appears to be quite restrictive in practice, as the number of useful functions may be small. In this case, the Public Key Encryption (PKE) can be transformed into FE by encrypting the evaluations of various functions using specific keys. However, this approach is not feasible for multi-user settings, even if a fixed function only is considered. To address this, Multi-Input Functional Encryption (MIFE) and Multi-Client Functional Encryption (MCFE) were thus introduced [GGG$^+$14, GKL$^+$13], allowing multiple clients to encrypt their individual data independently and contribute encrypted inputs to a joint function, with the help of possibly a trusted authority who runs the setup procedure and generates functional decryption keys. Among the classes of functions for MIFE/MCFE, the inner product is an expressive class that allows computing weighted averages and sums over encrypted data, making it especially useful for statistical analysis.

Chotard *et al.* [CDG$^+$18] first introduced the notion of decentralized MCFE (DMCFE) in which there is no requirement for a trusted authority, and each client can have a complete control over their encrypted individual data and over the generation of functional decryption keys. The authors also provided a DMCFE scheme for inner product that is secure in random oracle model and in which all clients only need to run an MPC protocol once during the setup. As follow-up works, new constructions of DMCFE for inner product that improve the security model such as by allowing incomplete ciphertext queries [ABKW19], by removing the random oracle [ABG19, LT19], or by allowing dynamic join of new users [CDSG+20] have been introduced. In particular, the MPC protocol in [CDG+18] was removed by a decentralized sum protocol in [CDSG+20], making the DMCFE for inner product completely non-interactive and eliminating the need for pairings in the groups. In this paper, we focus on the decentralized MCFE.

Importance of Verifiability in MCFE. Historically, the security of an encryption scheme has focused on the confidentiality of the message being encrypted. The (multi-client) functional encryption is not an exception, with its indistinguishability security ensuring that given two encrypted values and decryption keys for functions that evaluate the same at these two values, then it is computationally hard to distinguish between the ciphertexts of these two values. However, Badrinarayanan et al. [BGJS16] showed that the security of computation for an honest-but-curious receiver is necessary: a malicious sender could provide a false ciphertext and false functional decryption keys, so that the value encrypted within the ciphertext can vary when computed with these different functions through an honest decryption process. An analogous notion for the receiver in the multi-input setting is also provided.

In this work, we address a practical concern when using (decentralized) multi-client FE for inner product in real-world applications. The DMCFE for inner-product protocol can be run by thousands of senders, but they may not be all honest. If we assume that a small percentage of them are malicious, trying to bias the function evaluations by sending random data, or even fake data, and contributing dishonest functional key shares. To minimize the impact of these malicious clients, we propose a verification scheme for ciphertexts and one for functional decryption key shares, so that once all are valid, the decryption result is guaranteed to not be significantly biased. Furthermore, our concrete DMCFE scheme allows practically-efficient identification of malicious senders. Beyond the inner products, we define a verifiable DMCFE, which consequently provides input validation for the receiver as in [BGL+22]. Compared to their scheme, our verifiable MCFE scheme works on a larger class of functions than the sum, and does not require interaction between senders and receiver during the verification process.

Verifiable DMCFE for Inner Product. Verifiability for DMCFE in the general case is very difficult, because a small modification of the input can cause a significant difference in the output (e.g. inverse functions). We can formalize the validity condition as a predicate, depending on each application. However, for linear functions with small coefficients and small inputs (which are the most useful in practice, like average functions for example), a change in the input does not result in a major change in the output, unless there is a significant modification to an input. When the number of users is large enough, the inputs are bounded (which are often considered in Inner-Product Functional Encryption) then if an input is changed but still remains within a reasonable range, the output function will be quite close to the exact value. Additionally, most of the IP-DMCFE schemes need a final discrete logarithm computation to get the result, which requires it to be small enough, and so the inputs should also be in a reasonable range. For these reasons, we target DMCFE for inner product, and verifiability checks that the inputs stay within a specific range. Such a range verification will be our predicate in the general framework (for both the encrypted inputs and the functions in the keys).

A Real-Life Example. We consider *Aggregating Household Energy Consumption* as a practical motivation. For optimization purpose, an energy supplier may want to aggregate the units of energy (kilowatt-hours or kWh) consumed by its customers during some specific periods of the day. However, the energy consumption of each customer is a private information, as it may include, for example, the time they get up in the morning, leave their house, return home and which electronic devices they use. Still, they may be willing to help the supplier with their data to improve its service. To protect user's privacy, the customers are recommended to use a decentralized multi-client functional encryption to send their data in an encrypted form. However, nothing guarantees that the electricity supplier receives a correct aggregate of the metered energy consumption or at least an approximation of this value. In fact, some customers may provide malformed ciphertexts and malformed functional key shares to bias the joint-input function. Therefore, if we can enforce each client to correctly encrypt a value in some valid range and to generate a correct functional key share, the noise from the input made by a small number of malicious clients can be mitigated when the aggregate value is computed among a large number of clients.

The fact that this scenario has not been captured in prior work of (decentralized) MCFE is historically reasonable: in single-input FE, there is only one encryptor. When this encryptor wants to bias the result computed by the functional decryptor, he can encrypt an invalid input, such as values out of the use domain or singular points of the function. Since this FE is single-input, a receiver can trivially identify the invalidity of the input by looking at the result of the function. Therefore, the standard security notion of single-input FE only considers the confidentiality of the individual input, which is later inherited by DMCFE. On the other hand, a receiver in DMCFE, can only learn the joint function evaluated on the joint inputs, then it seems not trivial to efficiently identify invalid individual inputs of the malicious clients out of the valid ones. We stress that using functional encryption schemes for modular inner product over $\mathbb{Z}_p$ where p can be any prime [ALS16] to reduce the inner-product value space would not solve this problem. An adversary can always inject an arbitrary value to make the computation over $\mathbb{Z}_p$ become uniformly random over the space. Therefore, guaranteeing that each encrypted input is within a specified range will cause overhead costs but plays an important step in tackling this issue.

Our contributions for verifiable DMCFE can be listed as:

- **Concept**: We introduce the definition of verifiable DMCFE with the ability to identify malicious senders. The verifiability guarantees that the decryption process, given as input a vector of ciphertexts and functional key shares that passed public verification schemes, always outputs the delegated functions evaluated on a vector of inputs satisfying specific predicates. If any verification fails, verifiability guarantees that malicious senders will be identified.
- **Technique**: We develop a technique called *Combine-then-Descend*. This technique enables senders to combine their verifiable private inputs in exponents in a decentralized manner. Subsequently, the final result is descended to obtain the sum in scalars, which can be used within a pairing-based protocol. Private inputs are put in exponent to facilitate efficient verification using Σ-protocols. We exploit the particular setting of class group in which the final result falls in a subgroup where the discrete logarithm problem is easy. Then, we construct the One-time Decentralized Sum (ODSUM) scheme in class groups, which serves as the building block for subsequent constructions.
- **Construction**: We present a concrete construction of range-verifiable DMCFE for inner product. We show a technique of extending from one-time security to multiple-time security for the ODSUM scheme that preserves the efficiency of the proof of correct encoding. The resulting DMCFE scheme then has verifiability with overhead costs depending only on the range proof for the ciphertext. Notably, our approach efficiently addresses the problem of identifying an unbounded number of malicious senders, which remained unsolved in secure aggregation protocols like ACORN.

1.1 Technical Overview

Combine-then-Descend Technique. We construct a new decentralized sum scheme (DSUM) in a DDH group that has an easy DL subgroup (class group, [CL15]). Our DSUM scheme will not compute the pair-wise shared masks for private input by using a pseudo-random function (PRF) as in [CDSG+20]. The reason is that using a general non-interactive zero-knowledge argument (NIZK) to prove the correct computation of a PRF on input an exchanged key can be very expensive. Instead, each private input will be encoded as a power of the generator f of the easy DL subgroup and masked directly by pair-wise exchanged keys in the bigger group as follows

$$C_i = f^{x_i} \cdot \left(\prod_{i<j} T_j \cdot \prod_{i>j} T_j^{-1} \right)^{t_i} .$$

Here, C_i is the ciphertext of a sender $\mathcal{S}_i$ that encrypts x_i under a secret key t_i and each $(T_j)^{t_i}$ is a Diffie-Hellman exchanged key with a public key T_j of another sender $\mathcal{S}_j$. Given public parameters $(f, (T_j)_{j \neq i})$, then proving that C_i is encrypted correctly with the witness (x_i, t_i) can be done efficiently by using a Σ-protocol in an unknown-order group [GPS06, CCL+20]. After verifying that all ciphertexts are valid, a receiver can combine them into $f^{\sum_i x_i} = \prod_i C_i$, then efficiently descend the sum $\sum_i x_i$ from the power of f. Unlike in the standard DDH group, there is no restriction on the size of the sum to be descended when this DSUM scheme is instantiated in a class group. Therefore, with only a constant overhead for proving time and proof size, this DSUM scheme allows senders to jointly compute the sum of private random shares of other cryptographic protocols and to efficiently identify the senders who gave malformed ciphertexts. However, to decentralize an MCFE scheme for inner product as in [CDG+18], this DSUM scheme is not yet enough since for each setup of pair-wise key exchanges, the scheme only supports one-time encryption (ODSUM). We then show an extension from one-time secure DSUM to multiple-time secure DSUM by leveraging the encryption with labels of inner-product MCFE scheme in [CDG+18] itself so that the correct encryption of the resulting DSUM scheme can still be efficiently proved and verified by a Σ-protocol.

Range-Verifiable DMCFE *for Inner Product.* To mitigate the effect of malicious inputs on the inner-product evaluation, each sender is restricted to encrypt values within a data range, which is relatively small compared to the possible range of the plaintexts. We design a decentralized MCFE scheme where anybody can verify the correctness of each encryption and each functional key share. Our work will not focus on the proof schemes, but on the design of encryption scheme such that the relations for proofs of correct generation are simplified.

We use the following building blocks: the MCFE scheme for inner product from [CDG+18], a Σ-protocol, a range proof on Pedersen commitments, and the ODSUM scheme that we presented above.

To recall, a ciphertext in the MCFE scheme [CDG+18] is computed in the form of a Pedersen commitment with message x_i and an opening $\boldsymbol{s}_{\mathsf{MCFE},i}$, namely $[c_i] = [\boldsymbol{u}_\ell^\top]\cdot\boldsymbol{s}_{\mathsf{MCFE},i}+[x_i]$ where $[\boldsymbol{u}_\ell] \in \mathbb{G}^2$ is the output of a random oracle taking a label ℓ as input, and $\boldsymbol{s}_{\mathsf{MCFE},i}$ is a private encryption key that is chosen uniformly from $\mathbb{Z}_p^2$, and $x_i \in \mathbb{Z}_p$ is the value to encrypt. For an inner-product function $\boldsymbol{y}$, the functional decryption key is computed as $\mathsf{dk}_{\boldsymbol{y}} = (\mathsf{dk} := \sum_i \boldsymbol{s}_{\mathsf{MCFE},i} \cdot y_i, \boldsymbol{y})$. This scheme can be transformed into a decentralized MCFE by letting each sender use DSUM to encrypt his share of functional key $\boldsymbol{s}_{\mathsf{MCFE},i} \cdot y_i$, so that $\mathsf{dk}_{\boldsymbol{y}}$ will be revealed as the sum of all senders' shares.

For the ciphertext verification (and also for the key share verification), a commitment of private key $\boldsymbol{s}_{\mathsf{MCFE},i}$ needs to be produced as $([\boldsymbol{u}_{\ell_{\mathsf{MCFE},b}}^\top] \cdot \boldsymbol{s}_{\mathsf{MCFE},i})_{b\in[2]}$ where $[\boldsymbol{u}_{\ell_{\mathsf{MCFE},b}}] \in \mathbb{G}^2$ is the output of a random oracle taking an initialization label $\ell_{\mathsf{MCFE},b}$ as input, and published since the key generation process. The relation for a proof of correct encryption now states that a ciphertext is correct if it encrypts a value within a data range under the committed private key. A proof scheme for this relation is a combination of a Pedersen-commitment range proof and a Σ-protocol in the standard DDH group.

For the key share verification, on one hand we want a DSUM scheme that supports multi-label encryption as in [CDSG+20], that is, only ciphertexts generated under the same label can be combined to decrypt the sum of encrypted inputs. If an adversary mixes and matches ciphertexts of different labels, he receives nothing. On the other hand, the relation for a proof of correct encryption has to be simple so that it can be proved by a Σ-protocol. An MCFE for inner product (so for the sum) has the former property, while an ODSUM has the latter. Therefore, we leverage both these schemes to achieve a label-supporting LDSUM with efficient proofs of correctness: in the key generation process, each sender publishes an ODSUM encryption of his private MCFE key $\boldsymbol{s}_{\mathsf{LDSUM},i} \in \mathbb{Z}_p$ as his public key, then each input x_i is encrypted by the MCFE scheme under a label ℓ and a private key $\boldsymbol{s}_{\mathsf{LDSUM},i}$. To decrypt the sum $\sum_i x_i$, a receiver first collects all senders' public keys to reveal $\mathsf{dk}_{\mathbf{1}} = \sum_i \boldsymbol{s}_{\mathsf{LDSUM},i} \in \mathbb{Z}_p$, which is exactly the MCFE functional decryption key for vector $\mathbf{1} = (1, ..., 1)$ (the sum). Using $\mathsf{dk}_{\mathbf{1}}$, he can continue to decrypt the sum of x_i that is encrypted by the MCFE scheme. An important point is that the order of the easy-DL subgroup in the class group can be instantiated to be equal to the prime order p of the standard DDH group. Therefore, both the encryption-key space of LDSUM and the plaintext space of ODSUM are $\mathbb{Z}_p$.

A final point to note is that when using the LDSUM scheme to encrypt MCFE key shares $\boldsymbol{s}_{\mathsf{MCFE},i} \cdot y_i$, the functional key dk in $\mathsf{dk}_{\boldsymbol{y}} = (\mathsf{dk} := \sum_i \boldsymbol{s}_{\mathsf{MCFE},i} \cdot y_i, \boldsymbol{y})$ may not be revealed as a scalar. The reason is that LDSUM is technically a particular instantiation of the MCFE scheme for inner product in [CDG+18], which can only decrypt when the inner product is small enough by computing a discrete logarithm. Therefore, we will use a pairing group to solve this issue.

1.2 Related Work and Comparisons

Formalization. Our definition of verifiable decentralized MCFE is a generalized computational version of the verifiability for MIFE in [BGJS16]. The first additional point is that the verifiability in our definition implies the validation of encrypted inputs with respect to a class of predicates. Moreover, the decentralized multi-client setting is a more general context: there is no central functional key authority, and each sender generates ciphertexts and functional key shares independently. While this setting is not considered in [BGJS16], our verifiability guarantees that any malicious sender, who gave malformed ciphertexts and functional key shares to make a global public verification with those of other honest senders fail, will be identified. On the other hand, if we restrict the functionality to be the sum of encrypted inputs, then we can obtain an analogous input validation for secure aggregation (ACORN protocols) as in [BGL+22]. Our protocol and ACORN protocols [BGL+22] have been independently developed using completely different approaches, which we consider below.

Solutions for Efficient Malicious Sender Identification For all protocols that allow multiple senders to compute a joint function on their private inputs, Malicious Sender Identification is a desirable feature, but it is not obvious to obtain within a practical efficiency. An example is that both our DMCFE scheme and the ACORN protocols in [BGL+22] need a decentralized sum to allow senders to generate ciphertexts that encrypt the decryption key shares of the bigger protocol in a decentralized manner. We both had the same problem in achieving the input validation: it could be very costly to use a general NIZK to prove and verify the correct encryption of the initial underlying decentralized sum.

To overcome this issue, the authors in [BGL+22] proposed two protocols: ACORN-detect and ACORN-robust. The first allows validating the aggregated (decryption) key, which is combined from all key shares of senders. Each key share is committed by a Pedersen commitment, and the combined key is compared with the aggregation of committed key shares thanks to the homomorphic property of the commitment. Besides requiring an interactive Σ-protocol between the server and each sender, a major drawback is that now a sender can send a malicious key share to make the combined key broken without being identified. The second protocol ACORN-robust can identify malicious senders and remove their inputs based on the help of neighbour honest senders, but allows at most $\frac{1}{3}$ number of senders to be malicious and at least 6 rounds of interaction between each sender and the server (more rounds of interaction may happen with a decreasing probability). In our verifiable inner-product DMCFE scheme, we gain the efficiency by constructing and then adapting a new decentralized sum that is efficient to verify. The result covers all and even better advantages of the two previous ACORN protocols: our DMCFE scheme has malicious sender identification, requires no round of interaction between each sender and a receiver, allows an unbounded number of malicious senders, and eventually allows a larger class of functionality (inner product over sum). Notably, in our verifiable DMCFE scheme, the constant time (group exponentiations) and the constant size (group elements) for proving each key share can even be more efficient than those for each ciphertext, which are dominated by a range proof as in ACORN.

2 Preliminaries

2.1 Groups and Assumptions

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

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

Definition 1 (Decisional Diffie-Hellman Assumption). *The Decisional Diffie-Hellman Assumption states that, in a prime-order group $\mathcal{G} \overset{\$}{\leftarrow} \mathsf{GGen}(1^\lambda)$, no PPT adversary can distinguish between the two following distributions with non-negligible advantage:*

$$\{([a], [r], [ar]) | a, r \overset{\$}{\leftarrow} \mathbb{Z}_p\} \text{ and } \{([a], [r], [s]) | a, r, s \overset{\$}{\leftarrow} \mathbb{Z}_p\}.$$

Equivalently, this assumption states it is hard to distinguish, knowing $[a]$, a random element from the span of $[\boldsymbol{a}]$ for $\boldsymbol{a} = (1, a)$, from a random element in $\mathbb{G}^2$: $[\boldsymbol{a}] \cdot r = [\boldsymbol{a}r] = ([r], [ar]) \approx ([r], [s])$.

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

Definition 2 (Symmetric eXternal Diffie-Hellman Assumption). *The Symmetric eXternal Diffie-Hellman (SXDH) Assumption states that, in a pairing group $\mathcal{PG} \xleftarrow{\$} \mathsf{PGen}(1^\lambda)$, the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Class Group. We recall the notion of a DDH group with an easy DL subgroup (first introduced in [CL15]), which can be instantiated from class groups of imaginary quadratic fields and also recall the corresponding computational assumptions.

Definition 3 (Generator for a DDH group with an easy DL subgroup [CCL+19,CCL+20]). *Let GenClassGroup be a pair of algorithms* $(\mathsf{Gen}, \mathsf{Solve})$. *The* Gen *algorithm is a group generator which takes as inputs a security parameter λ and a prime p and outputs a tuple $(p, \tilde{s}, \hat{g}, f, \hat{g}_p, \hat{G}, F, \hat{G}^p)$. The set $(\hat{G}, \cdot)$ is a cyclic group of odd order ps where s is an integer, p is a μ-bit prime, and* $\gcd(p, s) = 1$. *The algorithm* Gen *only outputs an upper bound $\tilde{s}$ of s. The set $\hat{G}^p = \{x^p, x \in \hat{G}\}$ is the subgroup of order s of $\hat{G}$, and F is the subgroup of order p of $\hat{G}$, so that $\hat{G} = F \times \hat{G}^p$. The algorithm* Gen *outputs f, $\hat{g}_p$ and $\hat{g} = f.\hat{g}_p$ which are respective generators of $F, \hat{G}^p$ and $\hat{G}$. Moreover, the* DL *problem is easy in F, which means that the* Solve *algorithm is a deterministic polynomial time algorithm that solves the discrete logarithm problem in F.*

An important feature of the $\mathsf{GenClassGroup}$ is that we can choose the same prime order as in the standard DDH (including pairing groups) for the easy DL subgroup. A concrete instantiation of such a group can be found in [CCL+19].

Let g_p be a random power of $\hat{g}_p$, G^p be a subgroup generated by g_p, and G be a subgroup generated by $g := g_p f$. The following assumption is called *Hard subgroup membership* assumption, which states that it is hard to distinguish random elements of G^p in G.

Definition 4 (HSM assumption [CCL+20]). *Let GenClassGroup* $= (\mathsf{Gen}, \mathsf{Solve})$ *be a generator for DDH groups with an easy DL subgroup. Let $(\tilde{s}, f, \hat{g}_p, \hat{G}, F)$ be an output of* Gen, *g_p be a random power of $\hat{g}_p$, and $g := g_p f$. We denote by $\mathcal{D}$ (resp. $\mathcal{D}_p$) a distribution over the integers s.t. the distribution $\{g^x, x \hookleftarrow \mathcal{D}\}$ (resp. $\{\hat{g}_p^x, x \hookleftarrow \mathcal{D}_p\}$) is at distance less than $2^{-\lambda}$ from the uniform distribution in $\langle g \rangle$ (resp. in $\langle \hat{g}_p \rangle$) . Let $\mathcal{A}$ be an adversary for the HSM problem, its advantage is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}(\lambda) := \left| \Pr \begin{bmatrix} (\tilde{s}, f, \hat{g}_p, F, \hat{G}^p) \leftarrow \mathsf{Gen}(1^\lambda, p), t \leftarrow \mathcal{D}_p, g_p = \hat{g}_p^t, & \\ x \hookleftarrow \mathcal{D}, x' \hookleftarrow \mathcal{D}_p, b \xleftarrow{\$} \{0,1\}, & \\ Z_0 \leftarrow g^x, Z_1 \leftarrow g_p^{x'}, & : b = b' \\ b' \leftarrow \mathcal{A}(p, \tilde{s}, f, \hat{g}_p, g_p, F, \hat{G}^p, Z_b, \mathsf{Solve}(\cdot)) & \end{bmatrix} - \frac{1}{2} \right|$$

The HSM problem is said to be hard in G if for all probabilistic polynomial time attacker $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}(\lambda)$ *is negligible.*

From [CLT18,CCL+19], one can set $S := 2^{\lambda-2} \cdot \tilde{s}$, and instantiate $\mathcal{D}_p$ as the uniform distribution on $\{0, ..., S\}$ and $\mathcal{D}$ as the uniform distribution on $\{0, ..., pS\}$. We also put the *Low order assumption* and the *Strong root assumption* in Appendix A, which is used to prove the soundness of our Σ-protocol over the class group.

2.2 Non-interactive Zero-knowledge Proofs

Zero-knowledge Proofs. Let $\mathcal{R}$ be a polynomial-time decidable relation. We call w a witness for a statement u if $\mathcal{R}(u;w)=1$. A language L associated with $\mathcal{R}$ is defined as $L=\{u|\exists w:\mathcal{R}(u;w)=1\}$. A zero-knowledge proof for L consists of a pair of algorithms $(\mathcal{P},\mathcal{V})$ where $\mathcal{P}$ convinces $\mathcal{V}$ that a common input $u\in L$ without revealing information about a witness w. If $u\notin L$, $\mathcal{P}$ has a negligible chance of convincing $\mathcal{V}$ to accept that $u\in L$. In a zero-knowledge proof of knowledge, $\mathcal{P}$ additionally proves that it owns a witness w as input such that $\mathcal{R}(u;w)=1$. In this work, we focus on the non-interactive proofs where $\mathcal{P}$ sends only one message π to $\mathcal{V}$. On the input π, some public parameters and its own inputs, $\mathcal{V}$ decides to accept or not. A formal definition, from [AGM18, BFM88, FLS90], is given below.

Definition 5 (Non-interactive Zero-knowledge Argument). *A NIZK argument for a language L defined by an NP relation $\mathcal{R}$ consists of a triple of PPT algorithms* $(\mathsf{SetUp},\mathsf{Prove},\mathsf{Verify})$*:*

– $\mathsf{SetUp}(\lambda)$*: Takes as input a security parameter λ, and outputs a common reference string (CRS) σ. The CRS is implicit input to other algorithms;*
– $\mathsf{Prove}(u,w)$*: Takes as input a statement u and a witness w, and outputs an argument π.*
– $\mathsf{Verify}(u,\pi)$*: Takes as input a statement u and an argument π, outputs either 1 accepting the argument or 0 rejecting it.*

Sometimes in this paper we will call π a proof. The algorithms satisfy the following properties.

1. *Completeness. For all u,w such that $\mathcal{R}(u;w)=1$,*
$$\Pr\left[\begin{matrix}\sigma\leftarrow\mathsf{SetUp}(\lambda),\\ \pi\leftarrow\mathsf{Prove}(u,w)\end{matrix}:\mathsf{Verify}(u,\pi)=1\right]=1.$$
2. *Computational Soundness. For all PPT adversaries $\mathcal{A}$, there is a negligible function $\mu(\lambda)$ such that*
$$\Pr\left[\begin{matrix}\sigma\leftarrow\mathsf{SetUp}(\lambda),\\ (u,\pi)\leftarrow\mathcal{A}(\sigma)\end{matrix}:\mathsf{Verify}(u,\pi)=1\wedge u\notin L\right]\leq\mu(\lambda).$$
3. *Zero-Knowledge. There exists a PPT simulator $(\mathcal{S}_1,\mathcal{S}_2)$ such that for all PPT adversaries $(\mathcal{A}_1,\mathcal{A}_2)$, there is a negligible function $\mu(\lambda)$ such that*
$$\left|\Pr\left[\begin{matrix}\sigma\leftarrow\mathsf{SetUp}(\lambda),\\ (u,w,\mathsf{st})\leftarrow\mathcal{A}_1(\sigma)\\ \pi\leftarrow\mathsf{Prove}(u,w)\end{matrix}:\begin{matrix}\mathcal{A}_2(\sigma,\pi,\mathsf{st})=1\\ \wedge\,\mathcal{R}(u;w)=1\end{matrix}\right]-\Pr\left[\begin{matrix}(\sigma,\tau)\leftarrow\mathcal{S}_1(\lambda),\\ (u,w,\mathsf{st})\leftarrow\mathcal{A}_1(\sigma)\\ \pi\leftarrow\mathcal{S}_2(\sigma,u,\tau)\end{matrix}:\begin{matrix}\mathcal{A}_2(\sigma,\pi,\mathsf{st})=1\\ \wedge\,\mathcal{R}(u;w)=1\end{matrix}\right]\right|\leq\mu(\lambda).$$
where τ is a trapdoor for σ and st *is an internal state.*

We defer the definitions and notations for non-interactive zero-knowledge arguments of knowledge and range proof to Appendix A.

2.3 Decentralized Sum

The decentralized sum (DSUM) [CDSG+20] is a primitive that allows several parties of a group to commit to values, so that only the sum of their values can be revealed when all parties of the group have sent the shares. Another important feature of DSUM is that there is no trusted party: each party totally controls the generation of its secret key. The definition of this primitive was first introduced as a particular case of the general Dynamic Decentralized Functional Encryption in [CDSG+20]. For the use in this work, we focus on a more relaxed security: given all senders' shares, a receiver cannot learn any information about individual inputs beyond their sum.

Definition 6 (Decentralized Sum). *A decentralized sum over an Abelian group $\mathcal{M}$ and a set of n senders consists of four algorithms:*

- $\mathsf{SetUp}(\lambda)$*: Takes as input the security parameter λ, and outputs the public parameters* pp*. The public parameters are implicit arguments to all the other algorithms;*
- $\mathsf{KeyGen}()$*: This is a protocol between the senders $(\mathcal{S}_i)_{i\in[n]}$ that eventually each generates its own secret key* sk_i*. The protocol also outputs a public key* pk*, which can be an implicit argument;*
- $\mathsf{Encrypt}(\mathsf{sk}_i, m_i)$*: Takes as input a secret key sk_i and a message m_i. Parses $m_i = (x_i, \ell)$ where $x_i \in \mathcal{M}$ and ℓ can be considered as an encryption label. Outputs the ciphertext $\mathsf{ct}_{\ell,i}$;*
- $\mathsf{Decrypt}(\epsilon, (\mathsf{ct}_{\ell,i})_{i\in[n]})$*: Takes as input an empty key ϵ (no private decryption key is required) and an n-vector ciphertext $(\mathsf{ct}_{\ell,i})_{i\in[n]}$ under the same label ℓ. Returns $\sum_{i\in[n]} x_i \in \mathcal{M}$ or $\perp$.*

Correctness. Given $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$, $((\mathsf{sk}_i)_{i\in[n]}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$, and $\mathsf{ct}_{i,\ell} \leftarrow \mathsf{Encrypt}(\mathsf{sk}_i, m_i)$ where $m_i = (x_i, \ell)$ for all $i \in [n]$, then the probability that $\mathsf{Decrypt}(\epsilon, (\mathsf{ct}_{\ell,i})_{i\in[n]}) = \sum_{i\in[n]} x_i$ is equal to 1.

Definition 7 (`IND`-Security Game for DSUM). *Let us consider a DSUM scheme over a message space $\mathcal{M}$ and a set of n senders. No adversary $\mathcal{A}$ should be able to win the following security game with a non-negligible probability against a challenger $\mathcal{C}$:*

- *Initialization: the challenger $\mathcal{C}$ runs the setup algorithm $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$ and the key generation $((\mathsf{sk}_i)_{i\in[n]}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$ and chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It sends $(\mathsf{pp}, \mathsf{pk})$ to the adversary $\mathcal{A}$.*
- *Encryption queries $\mathsf{QEncrypt}(i, x^0, x^1, \ell)$: $\mathcal{A}$ has unlimited and adaptive access to a Left-or-Right encryption oracle, and receives the ciphertext $\mathsf{ct}_{\ell,i}$ generated by $\mathsf{Encrypt}(\mathsf{sk}_i, (x_i, \ell))$. Any further query for the same pair (ℓ, i) will later be ignored.*
- *Corruption queries $\mathsf{QCorrupt}(i)$: $\mathcal{A}$ can make an unlimited number of adaptive corruption queries on input index i, to get the secret key sk_i of any sender i of its choice.*
- *Finalize: $\mathcal{A}$ provides its guess b' on the bit b, and this procedure outputs the result β of the security game, according to the analysis given below.*

The output β of the game depends on some conditions, where $\mathcal{CS}$ is the set of corrupted senders (the set of indexes i input to $\mathsf{QCorrupt}$ during the whole game), and $\mathcal{HS}$ is the set of honest (non-corrupted) senders. We set the output to $\beta \leftarrow b'$, unless one of the three cases below is true, in which case we set $\beta \xleftarrow{\$} \{0,1\}$:

1. *some $\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$-query has been asked for an index $i \in \mathcal{CS}$ with $x_i^0 \neq x_i^1$;*
2. *for some label ℓ, an encryption-query $\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$ has been asked for some $i \in \mathcal{HS}$, but encryption-queries $\mathsf{QEncrypt}(j, x_j^0, x_j^1, \ell)$ have not all been asked for all $j \in \mathcal{HS}$;*
3. *for some label ℓ, there exists a pair of vectors $(\boldsymbol{x}^0 = (x_i^0)_i, \boldsymbol{x}^1 = (x_i^1)_i)$ such that $\sum_i x_i^0 \neq \sum_i x_i^1$, when*
   - *$x_i^0 = x_i^1$, for all $i \in \mathcal{CS}$;*
   - *$\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$-queries have been asked for all $i \in \mathcal{HS}$.*

We say this DSUM is `IND`-secure if for any adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{DSUM}}(\mathcal{A}) = |P[\beta = 1|b = 1] - P[\beta = 1|b = 0]|$$

is negligible.

Weaker Notions. For some weaker variants of indistinguishability, some queries can only be sent before the initialization phase:

- Selective Security ($\mathtt{sel} - \mathtt{IND}$): the encryption queries ($\mathsf{QEncrypt}$) are sent before the initialization;
- Static Security ($\mathtt{sta} - \mathtt{IND}$): the corruption queries ($\mathsf{QCorrupt}$) are sent before the initialization.

3 ODSUM from Combine-then-Descend Technique

3.1 Motivation

In a high level overview, we want to develop a concrete DSUM scheme such that no adversary playing on behalf of some senders can make the decryption with other honest senders' encryptions fail, for example by behaving maliciously in the KeyGen process or sending maliciously generated encryptions.

A straight approach is letting each sender use a general NIZK to prove the correctness of his encryptions and send the proofs with them. However, this approach may bring a heavy computational overhead for the encryption time. For example, given a simplified instantiation (without the All-or-Nothing Encapsulation layer) of DSUM scheme from the construction in [CDSG+20], to encrypt a message $x_i \in \mathbb{Z}_p$ under a label $\ell \in \{0,1\}^*$, a sender $\mathcal{S}_i$ computes a ciphertext

$$c_{\ell,i} = x_i + \sum_{i<j} \mathsf{PRF}((T_j)^{t_i}, \ell) - \sum_{i>j} \mathsf{PRF}((T_j)^{t_i}, \ell) \in \mathbb{Z}_p.$$

where PRF is a pseudorandom function, and each $(T_j)^{t_i}$ is a pair-wise exchanged key in a DDH group $\mathbb{G}$ with a public key T_j and a secret key t_i. The complex part to be implemented for NIZK is the computation of the pseudorandom function on input a pair-wise exchanged key and an encryption label.

Our expected DSUM scheme will remove the use of PRF in the above manner. Instead, the input is encoded as a power of a (sub)-group generator f and is directly masked by group multiplications with the pair-wise exchanged keys. This helps greatly simplify the relation of correct encryption so that each ciphertext can be proved and verified by using a Σ-protocol only. As the computational cost and communication cost of such a Σ-protocol are constant, then the overhead is asymptotically optimal. After *combining* all valid encryptions by multiplying them together, the pair-wise masks vanish, and we want the receiver to efficiently *descend* the sum from the exponent of f. We refer to this technique as the *combine-then-descend* technique. The DDH groups with an easy DL subgroup [CL15], which are instantiated in class groups of imaginary quadratic fields, is an extremely suitable environment to construct our DSUM scheme with those desired properties. Moreover, unlike the composite modulus for plaintext in Paillier encryption, we can choose a prime for the order of f before creating a class group, which makes the sum computed by the DSUM scheme automatically compatible with other applications in pairing groups.

3.2 Class Group-Based One-time Decentralized Sum (ODSUM)

We construct a DSUM scheme from the combine-then-descend technique:

- $\mathsf{SetUp}(\lambda)$: It generates a DDH group with an easy DL subgroup $(\tilde{s}, f, \hat{g}_p, \hat{G}, F) \leftarrow \mathsf{GenClassGroup}(1^\lambda, p)$. It samples a $t \hookleftarrow \mathcal{D}_p$ and sets $g_p = \hat{g}_p^t$[3]. The public parameters are $\mathsf{pp} = (\tilde{s}, f, \hat{g}_p, g_p, \hat{G}, F, p)$, which is an implicit input to other algorithms.
- $\mathsf{KeyGen}()$: Each sender generates a secret key $\mathsf{sk}_i = t_i \hookleftarrow \mathcal{D}_p$ and publishes $T_i = g_p^{t_i}$. The public key is defined as $\mathsf{pk} = (T_i)_{i\in[n]}$.
- $\mathsf{Encrypt}(x_i, \mathsf{pk}, \mathsf{sk}_i)$: The encryption is supposed to be done one time for one message in the protocol, so there is no label. It generates a ciphertext

$$C_i = f^{x_i} \cdot \left(\prod_{i<j} T_j \cdot \prod_{i>j} T_j^{-1} \right)^{t_i}.$$

- $\mathsf{Decrypt}(\epsilon, (C_i)_{i\in[n]})$: No decryption key is required (empty key ϵ). It computes $M = \prod_{i\in[n]} C_i$ and outputs $\alpha \leftarrow \mathsf{Solve}(M) \in \mathbb{Z}_p$ or $\perp$.

[3] This step can be done in a decentralized manner, with up to $n_{\mathsf{SetUp}} - 1$ malicious parties out of n_{SetUp} as in the interactive setup for the CL scheme in [CCL+20].

Correctness. Given $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$, $((\mathsf{sk}_i)_i, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$, and the ciphertexts $C_i \leftarrow \mathsf{Encrypt}(x_i, \mathsf{pk}, \mathsf{sk}_i)$ for $i \in [n]$, we have

$$\begin{aligned} M = \prod_{i\in[n]} C_i &= \prod_{i\in[n]} \left(f^{x_i} \cdot \left(\prod_{i<j} T_j \cdot \prod_{i>j} T_j^{-1} \right)^{t_i} \right) \\ &= f^{\sum_{i\in[n]} x_i} \cdot \prod_{i\in[n]} g_p^{\sum_{i<j} t_j t_i - \sum_{i>j} t_j t_i} = f^{\sum_{i\in[n]} x_i}, \end{aligned}$$

and $\mathsf{Solve}(M) = \sum_{i\in[n]} x_i$.

An important advantage of this scheme is that a proof of correct encryption can be generated by using a Σ-protocol in an unknown-order group [GPS06], which has a soundness based on the Strong Root Assumption and the γ-Low Order Assumption in class groups [CCL+20]. Moreover, thanks to the easy DL subgroup generated by f, there is no restriction on the size of the sum to be aggregated. If the above scheme is in a standard DDH group, a range proof for each encrypted input is required and thus the computational overhead and the proof size can not be constant anymore.

On the other hand, we call the above scheme as one-time decentralized sum (ODSUM), as it only supports one-time secure encryption. Therefore, each sender is supposed to encrypt a message once only. Without this restriction, an adversary can mix and match between (possibly) multiple ciphertexts of the same sender with other senders' ciphertexts in decryption to extract information related to the mixed-and-matched encrypted inputs. Later, we will provide a technique to extend from one-time security to multiple-time security and show that the extended DSUM scheme is applicable to be used in verifiable decentralized MCFE for inner product.

One-time Security Model. The DSUM scheme described in Section 3.2 is one-time secure, therefore the security model is as defined in Definition 7, except for the encryption oracle:

– Encryption queries $\mathsf{QEncrypt}(i, x^0, x^1)$: $\mathcal{A}$ has unlimited and adaptive access to a Left-or-Right encryption oracle, and receives the ciphertext ct_i generated by $\mathsf{Encrypt}(\mathsf{sk}_i, x_i)$ (no label ℓ). Any further query for the same sender i will later be ignored.

Theorem 1. *The One-time Decentralized Sum scheme described in Section 3.2 is* IND*-secure under the HSM assumption, as in the one-time security model above. More precisely, we have*

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{ODSUM}}(t, q_E) \le 2n(n-1)^2 \cdot (\mathsf{Adv}_{\mathsf{HSM}}(t) + 2^{-2\lambda})$$

where

– $\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{ODSUM}}(t, q_E)$ *is the best advantage of any PPT adversary running in time t with q_E encryption queries against the* IND*-security game of the* ODSUM *scheme;*
– $\mathsf{Adv}_{\mathsf{HSM}}(t)$ *is best advantage of any PPT adversary running in time t to distinguish a HSM instance.*
– $q_E \le n$ *according to the security model of* ODSUM.

Proof. We may note that to have a non-negligible advantage of winning the game, the adversary has to let at least two clients be non-corrupted (honest) such that each of them has two different messages (x^0, x^1) for encryption queries. We call such clients as explicitly honest clients. Indeed,

– if there is no explicitly honest client: unless the game output β is randomized by the finalizing condition 1 in Definition 7, the adversary has only access to the $\mathsf{QEncrypt}$ for any client index i of the same message $x_i^b = x_i^0 = x_i^1$, which implies that the adversary has no information about b;
– if there is only one explicitly honest client: we denote by (i, x_i^0, x_i^1) with $x_i^0 \ne x_i^1$ be the only query of two different messages to $\mathsf{QEncrypt}$, then the game output β is randomized by the finalizing condition 3 in Definition 7.

From above, we consider all PPT adversaries that let at least two clients be explicitly honest. We proceed by using a hybrid argument. Let $\mathcal{A}$ be a PPT adversary running in time t. For any game $\mathbf{G}$, we write $\mathsf{Adv}_{\mathbf{G}}$ the advantage of $\mathcal{A}$ in the game $\mathbf{G}$. Note that $\mathbf{G}_0$ is the security game defined in the one-time security model, whereas $\mathsf{Adv}_{\mathbf{G}_1} = 0$, since the adversary's view in $\mathbf{G}_1$ does not depend on the random bit $b \xleftarrow{\$} \{0,1\}$.

Game $\mathbf{G}_0^*$: this game is as $\mathbf{G}_0$, except the challenger guesses the number of explicitly honest clients. The challenger samples $\kappa \xleftarrow{\$} [2,n]$. If eventually the number of explicitly honest clients is not κ, the game output is $\beta \xleftarrow{\$} \{0,1\}$. We have that $\mathsf{Adv}_{\mathbf{G}_0^*} = \frac{\mathsf{Adv}_{\mathbf{G}_0}}{n-1}$. For all $t \in [2,\kappa]$, we define the following games.

Game $\mathbf{G}_{0,t}^*$: this game is as $\mathbf{G}_0^*$, except that for the first explicitly honest client $\mathsf{id}_1 \in [n]$, $\mathsf{QEncrypt}(\mathsf{id}_1, x_{\mathsf{id}_1}^0, x_{\mathsf{id}_1}^1)$ uses

$$C_{\mathsf{id}_1} = f^{x_{\mathsf{id}_1}^b \boxed{- \sum_{j \in \{2,t\}} u_j}} \cdot \prod_{\mathsf{id}_1 < j} (T_j)^{t_{\mathsf{id}_1}} \cdot \prod_{\mathsf{id}_1 > j} (T_j)^{-t_{\mathsf{id}_1}},$$

where $u_j \xleftarrow{\$} \mathbb{Z}_p$ for all $j \in [t]$. For the ρ'th explicitly honest client id_ρ with $1 < \rho \le t$, $\mathsf{QEncrypt}(\mathsf{id}_\rho, x_{\mathsf{id}_\rho}^0, x_{\mathsf{id}_\rho}^1)$ uses

$$C_{\mathsf{id}_\rho} = f^{x_{\mathsf{id}_\rho}^b \boxed{+ u_\rho}} \cdot \prod_{\mathsf{id}_\rho < j} (T_j)^{t_{\mathsf{id}_\rho}} \cdot \prod_{\mathsf{id}_\rho > j} (T_j)^{-t_{\mathsf{id}_\rho}}$$

The changes from $\mathbf{G}_0^*$ are highlighted in gray. From $\mathbf{G}_0^*$ to $\mathbf{G}_{0,2}^*$, we construct a sub-transition with a similar strategy as in the IND-security proof for the CL encryption scheme [CLT18]:

- **Game $\mathbf{G}_{sub}^0$**: this game is as $\mathbf{G}_0^*$, except that the challenger guesses the second explicitly honest client, denoted by id_2. If the guess is incorrect, the challenger aborts and returns a random bit. This incurs a security loss of n.
- **Game $\mathbf{G}_{sub}^1$**: this game is as $\mathbf{G}_{sub}^0$ except that the challenger creates secret keys $(t_i)_{i \in [n]}$ from a distribution $\mathcal{D}$ instead of $\mathcal{D}_p$, so that $(t_i)_{i \in [n]}$ are close to be uniform over the order of G (the subgroup generated by $g_p f$). For the pairwise-shared mask $K_{\mathsf{id}_2,i} := (T_{\mathsf{id}_2})^{t_i} = (T_i)^{t_{\mathsf{id}_2}}$ with $i \neq \mathsf{id}_2$ that appears in the encryption queries for id_2 and i respectively, the challenger will now compute $K_{\mathsf{id}_2,i} = (T_{\mathsf{id}_2})^{t_i}$. These two modifications does not change the adversary's view, so the simulation remains perfect.
- **Game $\mathbf{G}_{sub}^2$**: by guessing as in $\mathbf{G}_{sub}^0$, the challenger creates $T_{\mathsf{id}_2} = f^u g_p^{t_{\mathsf{id}_2}}$ with $u \xleftarrow{\$} \mathbb{Z}_p$ and $t_{\mathsf{id}_2} \hookleftarrow \mathcal{D}_p$. In other words, T_{id_2} is close to be uniform over G. It computes $K_{\mathsf{id}_2,i}$ with $i \neq \mathsf{id}_2$ as in the previous game, so we have

$$K_{\mathsf{id}_2,\mathsf{id}_1} = (T_{\mathsf{id}_2})^{t_{\mathsf{id}_1}} = f^{u t_{\mathsf{id}_1}} g_p^{t_{\mathsf{id}_2} t_{\mathsf{id}_1}}.$$

  The gap between $\mathbf{G}_{sub}^0$ and $\mathbf{G}_{sub}^2$ is

$$\left| \mathsf{Adv}_{\mathbf{G}_{sub}^0} - \mathsf{Adv}_{\mathbf{G}_{sub}^2} \right| \le \mathsf{Adv}_{\mathsf{HSM}}(t).$$

  As p is a 2λ-bit prime, the probability that $u = 0 \mod p$ is a negligible $2^{-2\lambda}$. On the other hand, t_{id_1} is close to be uniform over the order $n_G = p s_p$ of G with $\gcd(p, s_p) = 1$. Therefore the value $(t_{\mathsf{id}_1} \mod p)$ appearing in the exponent of f and the value $(t_{\mathsf{id}_1} \mod s_p)$ appearing in the exponent of g_p are independent (more details in Lemma 1, [CCL+19]). Unless $u = 0 \mod p$, the value $(u t_{\mathsf{id}_1} \mod p)$ is then uniformly random over modulus p, even when an unbounded adversary can extract $(t_{\mathsf{id}_1} \mod s_p)$ from T_{id_1} and $(K_{i,\mathsf{id}_1})_{i \neq \mathsf{id}_1}$.
- **Game $\mathbf{G}_{sub}^3$**: this game is as $\mathbf{G}_{sub}^2$, except that $K_{\mathsf{id}_2,\mathsf{id}_1}$ is computed as

$$K_{\mathsf{id}_2,\mathsf{id}_1} = f^{\mu_2 + u t_{\mathsf{id}_1}} g_p^{t_{\mathsf{id}_2} t_{\mathsf{id}_1}} = f^{\mu_2} (T_{\mathsf{id}_2})^{t_{\mathsf{id}_1}}$$

  where $\mu_2 \xleftarrow{\$} \mathbb{Z}_p$. Unless $u = 0 \mod p$, the distributions $\{u t_{\mathsf{id}_1} \mod p : t_{\mathsf{id}_1} \xleftarrow{\$} \mathbb{Z}_p\}$ in $\mathbf{G}_{sub}^2$ and $\{\mu_2 + u t_{\mathsf{id}_1} : \mu_2 \xleftarrow{\$} \mathbb{Z}_p, t_{\mathsf{id}_1} \xleftarrow{\$} \mathbb{Z}_p\}$ in $\mathbf{G}_{sub}^3$ are the same, so we have

$$\left| \mathsf{Adv}_{\mathbf{G}_{sub}^2} - \mathsf{Adv}_{\mathbf{G}_{sub}^3} \right| \le 2^{-2\lambda}.$$

By switching $T_{\mathsf{id}_2} = f^u g_p^{t_{\mathsf{id}_2}}$ back to $T_{\mathsf{id}_2} = g_p^{t_{\mathsf{id}_2}}$ and lifting the requirement that the challenger has to guess id_2, we obtain the game $\mathbf{G}_{0,2}^*$. Formally, we have

$$\left| \mathsf{Adv}_{\mathbf{G}_0^*} - \mathsf{Adv}_{\mathbf{G}_{0,2}^*} \right| \le 2n \cdot (\mathsf{Adv}_{\mathsf{HSM}}(t) + 2^{-2\lambda}).$$

The transition from $\mathbf{G}^*_{0,t-1}$ to $\mathbf{G}^*_{0,t}$ for $t \in [3,\kappa]$ is similar: the challenger has to guesses the t-th explicitly honest client. If the guess is unsuccessful, the challenger aborts and returns a random bit. As before, this incurs a security loss of n. We then use a similar game-based sub-transition as above. Eventually, we obtain

$$\left|\mathsf{Adv}_{\mathbf{G}^*_0} - \mathsf{Adv}_{\mathbf{G}^*_{0,\kappa}}\right| \leq 2(n-1)n \cdot (\mathsf{Adv}_{\mathsf{HSM}}(t) + 2^{-2\lambda}).$$

Game $\mathbf{G}^*_{1,\kappa}$: For all $i \in [n]$, we put $\Delta_i = x_i^0 - x_i^b$. This game is as $\mathbf{G}^*_{0,\kappa}$ except that u_t is replaced by $u_t + \Delta_{\mathsf{id}_t}$ for all $t \in [2,\kappa]$. As u_t is sampled uniformly, then the transition from $\mathbf{G}^*_{0,t}$ to $\mathbf{G}^*_{1,t}$ remains perfect.
On the other hand, by the condition (b) in the security game, we know that $\sum_{t\in[\kappa]} x^0_{\mathsf{id}_t} = \sum_{t\in[\kappa]} x^1_{\mathsf{id}_t}$, this implies that $x^0_{\mathsf{id}_1} = x^b_{\mathsf{id}_1} - \sum_{j\in\{2,t\}} \Delta_{\mathsf{id}_t}$. Therefore, we have

$$f^{x^b_{\mathsf{id}_1} - \sum_{j\in[2,\kappa]}(u_j + \Delta_{\mathsf{id}_t})} = f^{x^0_{\mathsf{id}_1} - \sum_{j\in[2,\kappa]} u_j}$$

and

$$f^{x^b_{\mathsf{id}_\rho} + (u_\rho + \Delta_{\mathsf{id}_\rho})} = f^{x^0_{\mathsf{id}_\rho} + u_\rho}$$

for $1 < \rho \leq \kappa$. In other words, this game is as $\mathbf{G}^*_{0,\kappa}$ except that $x^b_{\mathsf{id}_\rho}$ is replaced by $x^0_{\mathsf{id}_\rho}$ for all $\rho \in [\kappa]$. We transition gradually from $\mathbf{G}^*_{1,\kappa}$ to $\mathbf{G}^*_1$ as a switch back from $\mathbf{G}^*_{0,\kappa}$ to $\mathbf{G}^*_0$ when $x^b_{\mathsf{id}_\rho}$ is replaced by $x^0_{\mathsf{id}_\rho}$ for all $\rho \in [\kappa]$. Hence, all the answers to encryption queries in $\mathbf{G}^*_1$ are encryptions of x^0, which is independent of b. In the game $\mathbf{G}_1$, we finally lift the requirement that the challenger has to guess the number of explicitly honest clients κ.
In conclusion, we have

$$\mathsf{Adv}_{\mathbf{G}_0} \leq 2n(n-1)^2 \cdot (\mathsf{Adv}_{\mathsf{HSM}}(t) + 2^{-2\lambda}).$$

4 Verifiable Decentralized MCFE

We denote by $\mathcal{F}$ a class of n-ary functions from $\mathcal{M}^n$ to $\mathcal{X}$. We also denote by $\mathcal{P}^{\mathtt{m}} \subset \{0,1\}^*$ a class of polynomially-time-decidable predicates for message to encrypt and by $\mathcal{P}^{\mathtt{f}} \subset \{0,1\}^*$ a class of polynomially-time-decidable predicates for function in a functional decryption key.

Definition 8 (Verifiable Decentralized Multi-Client Functional Encryption). *A verifiable decentralized multi-client functional encryption on $\mathcal{M}$ over $(\mathcal{F}, \mathcal{P}^{\mathtt{m}}, \mathcal{P}^{\mathtt{f}})$, and a set of n senders $(\mathcal{S}_i)_i$ consists of eight algorithms :*

- $\mathsf{SetUp}(\lambda)$*: Takes as input the security parameter λ. Outputs the public parameters* pp*. Those parameters are implicit arguments to all the other algorithms.*
- $\mathsf{KeyGen}()$*: This is a protocol between the senders $(\mathcal{S}_i)_i$ that eventually each generates its own secret key sk_i, its private encryption key ek_i. The protocol also outputs a verification key for ciphertexts $\mathsf{vk}_{\mathsf{CT}}$, a verification key for functional keys $\mathsf{vk}_{\mathsf{DK}}$, a public key pk. Similar to pp, pk can be an implicit argument.*
- $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell, \mathsf{P}^{\mathtt{m}}_i)$*: Takes as input an encryption key ek_i, a value x_i to encrypt, a label ℓ and a predicate $\mathsf{P}^{\mathtt{m}}_i \in \mathcal{P}^{\mathtt{m}}$. Outputs the ciphertext $C_{\ell,i}$.*
- $\mathsf{DKeyGenShare}(\mathsf{sk}_i, \ell_f, \mathsf{P}^{\mathtt{f}})$*: Takes as input a user secret key sk_i, a function label ℓ_f for $f \in \mathcal{F}$, and a predicate $\mathsf{P}^{\mathtt{f}} \in \mathcal{P}^{\mathtt{f}}$. Outputs a functional decryption key share $\mathsf{dk}_{f,i}$.*
- $\mathsf{VerifyDK}((\mathsf{dk}_{f,i})_{i\in[n]}, \mathsf{vk}_{\mathsf{DK}}, \mathsf{P}^{\mathtt{f}})$*: Takes as input functional decryption key shares $(\mathsf{dk}_{f,i})_{i\in[n]}$, a verification key $\mathsf{vk}_{\mathsf{DK}}$ and a predicate $\mathsf{P}^{\mathtt{f}} \in \mathcal{P}^{\mathtt{f}}$. Outputs 1 for accepting or 0 with a set of malicious senders $\mathcal{MS}_{\mathsf{dk}} \neq \emptyset$ for rejecting.*
- $\mathsf{VerifyCT}(\boldsymbol{C}_\ell, \mathsf{vk}_{\mathsf{CT}}, (\mathsf{P}^{\mathtt{m}}_i)_{i\in[n]})$*: Takes as input an n-vector ciphertext $\boldsymbol{C}_\ell = (C_{\ell,i})_{i\in[n]}$, a verification key $\mathsf{vk}_{\mathsf{CT}}$, and message predicates $(\mathsf{P}^{\mathtt{m}}_i)_{i\in[n]} \in (\mathcal{P}^{\mathtt{m}})^n$. Outputs 1 for accepting or 0 with a set of malicious senders $\mathcal{MS}_{\mathsf{ct}} \neq \emptyset$ for rejecting.*
- $\mathsf{DKeyComb}((\mathsf{dk}_{f,i})_{i\in[n]}, \ell_f)$*: Takes as input the functional decryption key shares $(\mathsf{dk}_{f,i})_{i\in[n]}$, a function label ℓ_f, and outputs the functional decryption key dk_f.*
- $\mathsf{Decrypt}(\mathsf{dk}_f, \boldsymbol{C}_\ell)$*: Takes as input a functional decryption key dk_f, an n-vector ciphertext $\boldsymbol{C}_\ell := (C_{\ell,i})_{i\in[n]}$. Outputs $f(\boldsymbol{x})$ or $\perp$.*

Correctness. Given any set of message predicates $(\mathsf{P}^{\mathtt{m}}_i)_{i\in[n]} \in (\mathcal{P}^{\mathtt{m}})^n$ *and any function predicate* $\mathsf{P}^{\mathtt{f}} \in \mathcal{P}^{\mathtt{f}}$*: for all functions* $f \in \mathcal{F}$ *such that* $\mathsf{P}^{\mathtt{f}}(f) = 1$*, and all sets of values* $(x_1, ..., x_n) \in \mathcal{M}^n$ *such that* $\mathsf{P}^{\mathtt{m}}_i(x_i) = 1$ *for all* $i \in [n]$*, and*

$$\begin{cases} \mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda) \\ \big((\mathsf{sk}_i, \mathsf{ek}_i)_{i\in[n]}, \mathsf{vk}_{\mathsf{CT}}, \mathsf{vk}_{\mathsf{DK}}, \mathsf{pk}\big) \leftarrow \mathsf{KeyGen}() \\ C_{\ell,i} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell, \mathsf{P}^{\mathtt{m}}_i) \forall i \in [n] \\ \mathsf{dk}_{f,i} \leftarrow \mathsf{DKeyGenShare}(\mathsf{sk}_i, \ell_f, \mathsf{P}^{\mathtt{f}}) \forall i \in [n] \end{cases}$$

then

$$\begin{cases} \mathsf{VerifyDK}((\mathsf{dk}_{f,i})_{i\in[n]}, \mathsf{vk}_{\mathsf{DK}}, \mathsf{P}^{\mathtt{f}}) = 1 \\ \mathsf{VerifyCT}(\boldsymbol{C}_\ell, \mathsf{vk}_{\mathsf{CT}}, (\mathsf{P}^{\mathtt{m}}_i)_{i\in[n]}) = 1 \\ \mathsf{Decrypt}(\mathsf{dk}_f, \boldsymbol{C}_\ell) = f(x_1, ..., x_n) \end{cases}$$

with probability 1.

Verifiability with Malicious Sender Identification. For all PPT adversaries $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{\mathsf{verif}}_{\mathsf{VDMCFE}}(\mathcal{A})$ *in the following game is negligible in* λ*.*

- *Initialization: Challenger initializes by choosing classes of predicates* $\mathcal{P}^{\mathtt{m}}, \mathcal{P}^{\mathtt{f}}$ *and running* $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$*. It sends* $(\mathsf{pp}, \mathcal{P}^{\mathtt{m}}, \mathcal{P}^{\mathtt{f}})$ *to* $\mathcal{A}$*.*
- *Key generation queries* $\mathsf{QKeyGen}()$*: For only one time in the game,* $\mathcal{A}$ *can play on behalf of corrupted senders and call other non-corrupted senders to join a key generation protocol and together compute* $(\mathsf{vk}_{\mathsf{CT}}, \mathsf{vk}_{\mathsf{DK}}, \mathsf{pk})$*.*
- *Corruption queries* $\mathsf{QCorrupt}(i)$*:* $\mathcal{A}$ *can make an unlimited number of adaptive corruption queries on input an index* i*, to play on behalf of sender* $\mathcal{S}_i$ *in the protocol. If* i *was queried after* $\mathsf{QKeyGen}$*, then* $\mathcal{A}$ *additionally receives* $(\mathsf{sk}_i, \mathsf{ek}_i, \mathsf{vk}_{\mathsf{CT}}, \mathsf{vk}_{\mathsf{DK}})$ *and cannot play on behalf of* $\mathcal{S}_i$ *in the key generation process anymore.*
- *Encryption queries* $\mathsf{QEncrypt}(i, \ell, \mathsf{P}^{\mathtt{m}}_i)$*:* $\mathcal{A}$ *has unlimited and adaptive access to call an honest (non-corrupted) sender* $\mathcal{S}_i$ *to provide a correct encryption* $C_{\ell,i} = \mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell, \mathsf{P}^{\mathtt{m}}_i)$ *for some* x_i *such that* $\mathsf{P}^{\mathtt{m}}_i(x_i) = 1$*.* $\mathcal{A}$ *can only choose any message predicate* $\mathsf{P}^{\mathtt{m}}_i \in \mathcal{P}^{\mathtt{m}}$*, otherwise the query is ignored.*
- *Functional key share queries* $\mathsf{QDKeyGen}(i, f, \mathsf{P}^{\mathtt{f}})$*:* $\mathcal{A}$ *has unlimited and adaptive access to call an honest (non-corrupted) sender* $\mathcal{S}_i$ *to provide a correct functional key share* $\mathsf{dk}_{f,i} = \mathsf{DKeyGenShare}(\mathsf{sk}_i, f, \mathsf{P}^{\mathtt{f}})$*.* $\mathcal{A}$ *can only choose any message predicate* $\mathsf{P}^{\mathtt{f}} \in \mathcal{P}^{\mathtt{f}}$ *and then a function* f *such that* $\mathsf{P}^{\mathtt{f}}(f) = 1$*, otherwise the query is ignored.*
- *Finalize: let* $\mathcal{MS}_{\mathcal{A}}$ *be the set of corrupted senders, then* $\mathcal{A}$ *has to output verification keys* $(\mathsf{vk}_{\mathsf{CT}}, \mathsf{vk}_{\mathsf{DK}})$ *and public key* pk *from* $\mathsf{QKeyGen}()$*, message predicates* $(\mathsf{P}^{\mathtt{m}}_i)_{i\in[n]} \in (\mathcal{P}^{\mathtt{m}})^n$*, a function predicate* $\mathsf{P}^{\mathtt{f}} \in \mathcal{P}^{\mathtt{f}}$*, a label* ℓ*, malicious ciphertexts* $(C_{\ell,i})_{i\in\mathcal{MS}_{\mathcal{A}}}$*, and malicious functional key shares* $(\mathsf{dk}_{f_j,i})_{j,i\in\mathcal{MS}_{\mathcal{A}}}$ *for a polynomially number of functions* f_j *such that* $\mathsf{P}^{\mathtt{f}}(f_j) = 1$*. The ciphertexts of honest senders* $(C_{\ell,i})_{i\notin\mathcal{MS}_{\mathcal{A}}}$ *and their functional key shares* $(\mathsf{dk}_{f_j,i})_{j,i\notin\mathcal{MS}_{\mathcal{A}}}$ *are automatically completed by using the oracles* $\mathsf{QEncrypt}$ *and* $\mathsf{QDKeyGen}$*.*
- $\mathcal{A}$ *wins the game if one of the following cases happens:*
  - *If* $\mathsf{VerifyCT}(\boldsymbol{C}_\ell, \mathsf{vk}_{\mathsf{CT}}, (\mathsf{P}^{\mathtt{m}}_i)_{i\in[n]}) = 1$ *and, for all function queries* f_j*,* $\mathsf{VerifyDK}((\mathsf{dk}_{f_j,i})_{i\in[n]}, \mathsf{vk}_{\mathsf{DK}}, \mathsf{P}^{\mathtt{f}}) = 1$*: there does not exist a tuple of messages* $(x_i)_{i\in[n]}$ *such that, for all* $i \in [n]$*,* $\mathsf{P}^{\mathtt{m}}_i(x_i) = 1$*, and*
  $$\mathsf{Decrypt}(\mathsf{dk}_{f_j}, \boldsymbol{C}_\ell) = f_j(x_1, ..., x_n)$$
  with $\mathsf{dk}_{f_j} = \mathsf{DKeyComb}((\mathsf{dk}_{f_j,i})_{i\in[n]}, f_j)$ *for all functions* f_j*.*
  - *If* $\mathsf{VerifyCT}(\boldsymbol{C}_\ell, \mathsf{vk}_{\mathsf{CT}}, (\mathsf{P}^{\mathtt{m}}_i)_{i\in[n]}) = 0$ *or* $\mathsf{VerifyDK}((\mathsf{dk}_{f_j,i})_{i\in[n]}, \mathsf{vk}_{\mathsf{DK}}, \mathsf{P}^{\mathtt{f}}) = 0$ *for some* f_j*: the union* $\mathcal{MS} = \mathcal{MS}_{\mathsf{ct}} \cup \mathcal{MS}_{\mathsf{dk}}$ *contains an honest sender* $\mathcal{S}_i$*, in other words* $i \in \mathcal{MS}$ *but* $i \notin \mathcal{MS}_{\mathcal{A}}$*.*

In our definition of verifiable decentralized MCFE, each functional key share $\mathsf{dk}_{f,i}$ is assumed to contain the description of its corresponding function f, and then a receiver can easily detect if $\mathsf{P}^{\mathtt{f}}(f) \neq 1$ in VerifyDK and reject the key share. Therefore, the condition that $\mathsf{P}^{\mathtt{f}}(f_j) = 1$ in the finalization phase of the verifiability game makes sense. The first winning condition is determined statistically: the validity of ciphertext verification and functional key verification guarantees that an adversary could not produce (maliciously generated) ciphertexts $(C_{\ell,i})_{i\in[n],i\in\mathcal{MS}_\mathcal{A}}$ and functional key shares $(\mathsf{dk}_{f_j,i})_{i\in\mathcal{MS}_\mathcal{A}}$, such that there exists no tuple of inputs $(x_1,..,x_n)$ that satisfy all message predicates and are consistent in the decryption to $f_j(x_1,...,x_n)$ for all f_j. The second winning case guarantees that if any verification fails, then there is a negligible chance that an honest sender is accused.

Our definition of verifiability for decentralized MCFE, in particular the first winning condition, is partially inspired by the definition of verifiability for MIFE in [BGJS16]. The intuition of verifiability in [BGJS16] guarantees that no matter how the setup is done, for (possibly maliciously generated) every n-vector ciphertext $\boldsymbol{C}$ that is valid to a publicly known verification, there must exist an n-vector plaintext $\boldsymbol{x}$ such that for (possibly maliciously generated) every functional decryption key $\mathsf{dk}_f = (\mathsf{dk}, f)$ that is valid to another publicly known verification, the decryption algorithm on input $(\boldsymbol{C}, \mathsf{dk}_f)$ must output $f(\boldsymbol{x})$. By introducing predicates for messages to encrypt and a predicate for function to generate a functional decryption key, our definition additionally validates the content of messages within ciphertexts and the content of functions within functional decryption keys. Furthermore, we formalize the property of malicious sender identification in a general context where multiple independent clients join the protocol.

To be more detailed, using our syntax for the verifiability game, the definition of verifiable MIFE in [BGJS16] differs in the following points:

- **Functional encryption.** Multi-input setting is considered instead of multi-client setting, i.e. in MIFE, there is no restriction that only ciphertexts under the same label ℓ can decrypt.
- **Message and function predicates.** In verifiable MIFE, it is fixed from the initialization that $\mathsf{P}^{\mathtt{m}}_i(x) = 1$ iff $x \in \mathcal{M}$ for all $i \in [n]$ and $\mathsf{P}^{\mathtt{f}}(f) = 1$ iff $f \in \mathcal{F}$.
- **Malicious Sender Identification**: In verifiable MIFE, each ciphertext is verified separately, and the functional decryption keys dk_f to be verified are given by a central key authority.
- **Adversary assumption.** In verifiable MIFE, the verifiability game is defined for any adversary that has unlimited computing power and this adversary is allowed to choose pp. The advantage of such adversary in the game is 0 (verifiability with no trusted party and perfect soundness).

Our verifiability requires computational soundness and the adversary is not allowed to create all the setup parameters (pp must be chosen by the challenger). This relaxation might help us to obtain verifiable MCFE schemes with practical efficiency and it might be reasonable in practice to have minimal public parameters that only consist of computational assumptions and random oracle.

If we restrict the functionality of verifiable decentralized MCFE to be the sum of encrypted inputs, then we can obtain a protocol with the same feature as the validation for secure aggregation with input validation in [BGL+22]. In their ACORN protocols, each encrypted input is guaranteed to satisfy pre-defined predicates.

Indistinguishability Security. In addition to verifiability, privacy is still an essential security goal: it is derived from the indistinguishability security notion of decentralized MCFE [CDG+18] as follows.

Definition 9 (IND-Security Game for Verifiable DMCFE). *Let us consider a Verifiable DMCFE scheme over a set of n senders, a class function predicates $\mathcal{P}^{\mathtt{f}}$, and a class of message predicates $\mathcal{P}^{\mathtt{m}}$. No adversary $\mathcal{A}$ should be able to win the following security game with a non-negligible probability against a challenger $\mathcal{C}$:*

- *Initialization: the challenger $\mathcal{C}$ runs the setup algorithm $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$ and the key generation $\big((\mathsf{sk}_i, \mathsf{ek}_i)_{i\in[n]}, \mathsf{vk}_{\mathsf{CT}}, \mathsf{vk}_{\mathsf{DK}}, \mathsf{pk}\big) \leftarrow \mathsf{KeyGen}()$ and chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It sends $(\mathsf{vk}_{\mathsf{CT}}, \mathsf{vk}_{\mathsf{DK}}, \mathsf{pk})$ to the adversary $\mathcal{A}$.*
- *Encryption queries $\mathsf{QEncrypt}(i, x^0, x^1, \ell, \mathsf{P}^{\mathtt{m}}_i)$: $\mathcal{A}$ has unlimited and adaptive access to a Left-or-Right encryption oracle. If $\mathsf{P}^{\mathtt{m}}_i \in \mathcal{P}^{\mathtt{m}}$ and $\mathsf{P}^{\mathtt{m}}_i(x^0_i) = \mathsf{P}^{\mathtt{m}}_i(x^1_i) = 1$, then $\mathcal{A}$ receives the ciphertext $C_{\ell,i}$ generated by $\mathsf{Encrypt}(\mathsf{ek}_i, x^b_i, \ell, \mathsf{P}^{\mathtt{m}}_i)$. Otherwise, the query is ignored. We note that any further query for the same pair (ℓ, i) will later be ignored.*
- *Functional decryption key queries $\mathsf{QDKeyGen}(i, f, \mathcal{P}^{\mathtt{f}})$: $\mathcal{A}$ has unlimited and adaptive access to the senders running $\mathsf{DKeyGenShare}(\mathsf{sk}_i, \ell_f, \mathsf{P}^{\mathtt{f}})$ algorithm for any input function f of its choice. If $\mathsf{P}^{\mathtt{f}} \in \mathcal{P}^{\mathtt{f}}$ and $\mathsf{P}^{\mathtt{f}}(f) = 1$, it is given back the functional decryption key share $\mathsf{dk}_{f,i}$. Otherwise, the query is ignored.*
- *Corruption queries $\mathsf{QCorrupt}(i)$: $\mathcal{A}$ can make an unlimited number of adaptive corruption queries on input index i, to get the secret and encryption keys $(\mathsf{sk}_i, \mathsf{ek}_i)$ of any sender i of its choice.*

– *Finalize: $\mathcal{A}$ provides its guess b' on the bit b, and this procedure outputs the result β of the security game, according to the analysis given below.*

The output β of the game depends on some conditions, where $\mathcal{CS}$ is the set of corrupted senders (the set of indexes i input to QCorrupt *during the whole game), and $\mathcal{HS}$ is the set of honest (non-corrupted) senders. We set the output to $\beta \leftarrow b'$, unless one of the three cases below is true, in which case we set $\beta \xleftarrow{\$} \{0,1\}$:*

1. *some* $\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$*-query has been asked for an index $i \in \mathcal{CS}$ with $x_i^0 \neq x_i^1$;*
2. *for some label ℓ, an encryption-query* $\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$ *has been asked for some $i \in \mathcal{HS}$, but encryption-queries* $\mathsf{QEncrypt}(j, x_j^0, x_j^1, \ell)$ *have not all been asked for all $j \in \mathcal{HS}$;*
3. *for some label ℓ and for some function f asked to* QDKeyGen*, there exists a pair of vectors $(\boldsymbol{x}^0 = (x_i^0)_i, \boldsymbol{x}^1 = (x_i^1)_i)$ such that $f(\boldsymbol{x}^0) \neq f(\boldsymbol{x}^1)$, when*
   – *$x_i^0 = x_i^1$, for all $i \in \mathcal{CS}$;*
   – $\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$*-queries have been asked for all $i \in \mathcal{HS}$.*

We say this verifiable DMCFE *is* IND*-secure with respect to $\mathcal{P}^{\mathtt{f}}$ and $\mathcal{P}^{\mathtt{m}}$ if for any adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{VDMCFE}}(\mathcal{A}) = |P[\beta = 1 | b = 1] - P[\beta = 1 | b = 0]|$$

is negligible.

In this work, we also use the following weaker notion:

– Static Security ($\mathtt{sta} - \mathtt{IND}$): the corruption queries (QCorrupt) are sent before the initialization, while encryption queries can be sent adaptively during the game.

5 A Range-Verifiable **DMCFE** for Inner Product

5.1 Ciphertext Verification

For each encryption label ℓ, the ciphertext of the MCFE scheme in [CDG+18] is $[c_i] = [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i]$ where $[\boldsymbol{u}_\ell] \in \mathbb{G}^2$ is the output of a random oracle taking label ℓ as input, and $\boldsymbol{s}_i$ is a private encryption key that is chosen uniformly from $\mathbb{Z}_p^2$, and $x_i \in \mathbb{Z}_p$ is the value to encrypt. The ciphertext is in the form of a Pedersen commitment, where x_i is the committed value and $\boldsymbol{s}_i$ is a two-dimensional opening. There is a number of efficient range proof schemes [BBB+18, CKLR21, CGKR22] for the committed value in the Pedersen commitment:

$$\mathcal{R}_{\mathsf{range}}([c], l, r; \boldsymbol{s}, x) = 1 \leftrightarrow [c] = [\boldsymbol{u}^\top] \cdot \boldsymbol{s} + [x] \wedge x \in [l, r]$$

The functional key for an inner product with $\boldsymbol{y}$ in the MCFE scheme is $\mathsf{dk}_{\boldsymbol{y}} = (\mathsf{dk} := \sum_{i=1}^n y_i \cdot \boldsymbol{s}_i \in \mathbb{Z}_p^2, \boldsymbol{y})$. To avoid encryption under a false encryption key that is not consistent with the share $\boldsymbol{s}_i y_i$ (and vice versa), our scheme will require each sender to publish a commitment of his private encryption key as $\mathsf{com}_{\mathsf{ek}} = ([\boldsymbol{u}^\top_{\ell_{\mathsf{MCFE},b}}] \cdot \boldsymbol{s}_i)_{b \in [2]} \in \mathbb{G}^2$ during the key generation process, where $([\boldsymbol{u}^\top_{\ell_{\mathsf{MCFE},b}}])_{b \in [2]}$ are generated by a random oracle taking initialization labels $(\ell_{\mathsf{MCFE},b})_{b \in [2]}$ as input. This commitment is perfectly binding, which later makes proofs for ciphertexts and functional keys become proofs of membership. By using the soundness of these proofs, we can avoid a large security loss from multiple rewinding-based extractions [PS96, SG98] when proving the verifiability of our inner-product decentralized MCFE scheme.

Now each sender is required to provide a proof for the relation $\mathcal{R}_{\mathsf{Encrypt}}$:

$$\mathcal{R}_{\mathsf{Encrypt}}([c], \mathsf{com}_{\mathsf{ek}}, l, r; \boldsymbol{s}, x) = 1 \leftrightarrow \begin{cases} \quad [c] = [\boldsymbol{u}^\top] \cdot \boldsymbol{s} + [x] \\ \wedge\, x \in [l, r] \\ \wedge\, \mathsf{com}_{\mathsf{ek}} = ([\boldsymbol{u}^\top_{\ell_{\mathsf{MCFE},b}}] \cdot \boldsymbol{s})_{b \in [2]} \end{cases}$$

The above relation defines a non-trivial language $L_{\mathsf{Encrypt}} \subsetneq \mathbb{G}^3$ for $([c], \mathsf{com}_{\mathsf{ek}})$. On the other hand, a Σ-protocol, denoted by $\mathsf{NIZK}_{\mathsf{key}}$, can be used to prove the relation $\mathcal{R}_{\mathsf{key}}$:

$$\mathcal{R}_{\mathsf{key}}([c], \mathsf{com}_{\mathsf{ek}}; \boldsymbol{s}, x) = 1 \leftrightarrow \begin{cases} \quad [c] = [\boldsymbol{u}^\top] \cdot \boldsymbol{s} + [x] \\ \wedge\, \mathsf{com}_{\mathsf{ek}} = ([\boldsymbol{u}^\top_{\ell_{\mathsf{MCFE},b}}] \cdot \boldsymbol{s})_{b \in [2]} \end{cases}$$

One can combine a Σ-protocol and a range proof scheme to obtain a NIZK for the relation $\mathcal{R}_{\mathsf{Encrypt}}$ by Lemma 1.

5.2 Functional Key Share Verification

The MCFE scheme in [CDG+18] can be transformed into a decentralized MCFE by allowing each sender to use a DSUM scheme [CDSG+20] to encrypt his share of functional key $\boldsymbol{s}_{\mathsf{MCFE},i} \cdot y_i$, so that $\mathsf{dk}_{\boldsymbol{y}}$ will be revealed as the sum of all senders' shares. A requirement is that the DSUM scheme must support multi-label encryption, that is, only ciphertexts under the same label can be combined to decrypt the sum of encrypted inputs. For the context of decentralized MCFE, by controlling the (inner-product function) label in the decryption, an adversary cannot mixes shares of different functional keys of a sender and matches them with other senders' in the DSUM decryption to obtain valid functional keys of non-agreed functions. As shown in Section 3.2, the ODSUM scheme does not have this property, since we removed the pseudo-random function in the encryption to obtain efficiency for the proof of correct encryption.

From ODSUM *to Label-Supporting* DSUM. To solve the problem that ODSUM does not support multi-label encryption, we leverage again the inner-product MCFE scheme in [CDG+18] that has this property. We first give an intuitive construction for a DSUM scheme that both supports multi-label encryption and preserves the efficiency for the proof of correct encryption. We call this scheme LDSUM to differentiate with ODSUM and other DSUM schemes.

- **Key Generation**: Each client i generates its own secret key $\mathsf{sk}_{\mathsf{MCFE},i}$ for the MCFE scheme. He joins the key generation of ODSUM with other senders to obtain a secret key $\mathsf{sk}_{\mathsf{ODSUM},i}$ and public key ODSUM.pk in a decentralized manner. His secret key is now $(\mathsf{sk}_{\mathsf{MCFE},i}, \mathsf{sk}_{\mathsf{ODSUM},i})$. He uses ODSUM to encrypt $\mathsf{sk}_{\mathsf{MCFE},i}$ under the keys $(\mathsf{sk}_{\mathsf{ODSUM},i}, \mathsf{ODSUM.pk})$. The resulting ciphertext, denoted by pk_i, is public.
- **Encryption**: Each sender $\mathcal{S}_i$ uses the MCFE scheme to encrypt his message x_i under the key $\mathsf{sk}_{\mathsf{MCFE},i}$ and a label ℓ. The resulting ciphertext is denoted by $c_{i,\ell}$.
- **Decryption**: The receiver first collects all pk_i and uses ODSUM to decrypt dk. Then he collects all MCFE ciphertexts $c_{i,\ell}$ under the same label and uses the MCFE scheme to decrypt $\sum_i x_i$ with the key dk.

The correctness of the above scheme comes from the fact that an MCFE functional key for sum, which is presented by vector $\mathbf{1} = (1, ..., 1)$, is the sum of all senders' MCFE secret keys. We have $\mathsf{dk} = \mathsf{ODSUM.Decrypt}((\mathsf{pk}_i)_i) = \sum_i \mathsf{sk}_{\mathsf{MCFE},i}$. Therefore, the correctness is implied by the correctness of ODSUM and MCFE.

A formal description of the LDSUM scheme is given as follows.

- $\mathsf{SetUp}(\lambda)$:
  1. It generates a prime-order group $\mathcal{G} := (\mathbb{G}, p, P) \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, and $\mathcal{H}$ a full-domain hash function onto $\mathbb{G}^2$.
  2. It generates the setup of One-time Decentralized Sum $\mathsf{ODSUM.SetUp}(\lambda) = (\tilde{s}, f, \hat{g}_p, g_p, \hat{G}, F, p)$ (a class group).
  3. The public parameters pp consist of $((\mathcal{G}, \mathcal{H}), (\tilde{s}, f, \hat{g}_p, g_p, \hat{G}, F, p))$ and are implicit arguments to all other algorithms.
- $\mathsf{KeyGen}()$:
  1. Each sender generates $\boldsymbol{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$ for all $i \in [n]$.
  2. Each sender joins $\mathsf{ODSUM.KeyGen}()$ and obtains two instances $(t_{i,b}, T_{i,b}, \mathsf{ODSUM.pk}_b)_{b\in[2]}$.
  3. Each sender computes and publishes a global key share for the sum:
  $$\mathsf{dk}_i = (\mathsf{ODSUM.Encrypt}(s_{i,b}, \mathsf{ODSUM.pk}_b, t_{i,b}))_{b\in[2]} \in G^2$$
  4. For each sender, the secret key is $\mathsf{sk}_i = (\boldsymbol{s}_i, \boldsymbol{t}_i := (t_{i,1}, t_{i,2}))$. The public key is $\mathsf{pk} = ((\mathsf{ODSUM.pk}_b)_{b\in[2]}, (\mathsf{dk}_i)_{i\in[n]})$.
- $\mathsf{Encrypt}(\mathsf{sk}_i, x_i, \ell)$:
  1. It parses $\mathsf{sk}_i = (\boldsymbol{s}_i, \boldsymbol{t}_i)$.
  2. It computes $[\boldsymbol{u}_\ell] = \mathcal{H}(\ell)$, and computes $[c_{\ell,i}] = [\boldsymbol{u}_\ell^\top \boldsymbol{s}_i + x_i] \in \mathbb{G}$.
  3. The ciphertext is $C_{\ell,i} := (\ell, [c_{\ell,i}])$.
- $\mathsf{Decrypt}(\boldsymbol{C}_\ell, \mathsf{pk})$:
  1. It parses $\boldsymbol{C}_\ell := (C_{\ell,i})_{i\in[n]}$ and $\mathsf{pk} = ((\mathsf{ODSUM.pk}_b)_{b\in[2]}, (\mathsf{dk}_i)_{i\in[n]})$.

2. It recovers the (public) decryption key for the sum:
$$\mathsf{dk}_{\mathbf{1}} \leftarrow (\mathsf{ODSUM.Decrypt}((\mathsf{dk}_{i,b})_{i\in[n]}))_{b\in[2]} \in \mathbb{Z}_p^2.$$
3. Decryption for the sum: from $C_{\ell,i} = (\ell, [c_{\ell,i}])$, it computes
$$[\alpha] = \sum_i [c_i] - [\boldsymbol{u}_\ell^\top] \cdot \mathsf{dk}_{\mathbf{1}},$$
and eventually solves the discrete logarithm to extract and return α. For efficient decryption, we require α to be small enough.

A formal proof of correctness and a security analysis of the above scheme are provided in Theorem 4. It is also more convenient to leave the relation for proof of correct generation and the corresponding Σ-protocol in the description of the verifiable inner-product DMCFE scheme.

5.3 Description of Range-Verifiable Inner-Product DMCFE Scheme

Let n be the number of senders. The message predicate $\mathsf{P}^{\mathtt{m}}(x) = 1 \leftrightarrow x \in [0, 2^m - 1]$ and the function predicate $\mathsf{P}^{\mathtt{f}}(\boldsymbol{y}) = 1 \leftrightarrow y_i \in [0, 2^m - 1]$ for all $i \in [n]$ are parameterized by a polynomially bounded m.

– $\mathsf{SetUp}(\lambda)$:
  1. It generates a pairing group $\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p) \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$, and $\mathcal{H}_b$ a full-domain hash function onto $\mathbb{G}_b^2$ for $b \in [2]$.
  2. It generates initialization labels $(\ell_{\mathsf{DMCFE},b})_{b\in[2]} := (\{0,1\}^*)^2$.
  3. It generates the setup of LDSUM in $\mathbb{G}_2$: $\mathsf{LDSUM.pp} = \mathsf{LDSUM.SetUp}(\lambda, \mathbb{G}_2)$.
  4. The public parameters pp consist of $(\mathcal{PG}, (\mathcal{H}_b)_{b\in[2]}, \mathsf{LDSUM.pp}, \ell_{\mathsf{DMCFE}})$ and are implicit arguments to all other algorithms.

– $\mathsf{KeyGen}()$:
  1. Each sender joins $\mathsf{LDSUM.KeyGen}()$ to obtain $(\mathsf{LDSUM.sk}_i, \mathsf{LDSUM.pk})$.
  2. Each sender generates $\boldsymbol{s}_{\mathsf{DMCFE},i} \xleftarrow{\$} \mathbb{Z}_p^2$ and commits $\boldsymbol{s}_{\mathsf{DMCFE},i}$ as
  $$\mathsf{com}_{\mathsf{DMCFE},i} = ([\boldsymbol{v}_{\mathsf{DMCFE},b}^\top \cdot \boldsymbol{s}_{\mathsf{DMCFE},i}]_1)_{b\in[2]};$$
  where $[\boldsymbol{v}_{\mathsf{DMCFE},b}]_1 = \mathcal{H}_1(\ell_{\mathsf{DMCFE},b})$ for $b \in [2]$.
  3. For each sender, the encryption key is $\mathsf{ek}_i = \boldsymbol{s}_{\mathsf{DMCFE},i}$ and the secret key is $\mathsf{sk}_i = (\boldsymbol{s}_{\mathsf{DMCFE},i}, \mathsf{LDSUM.sk}_i)$.
  4. The verification key for ciphertexts is $\mathsf{vk}_{\mathsf{CT}} = (\mathsf{com}_{\mathsf{DMCFE},i})_{i\in[n]}$, while for functional keys it is $\mathsf{vk}_{\mathsf{DK}} = (\mathsf{LDSUM.pk}, (\mathsf{com}_{\mathsf{DMCFE},i})_{i\in[n]})$.
  5. The public key is $\mathsf{pk} = \mathsf{LDSUM.pk}$.

– $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell, m)$:
  1. It parses $\mathsf{ek}_i = \boldsymbol{s}_{\mathsf{DMCFE},i}$ and computes $[c_{\ell,i}]_1 = [\boldsymbol{u}_\ell^\top \boldsymbol{s}_{\mathsf{DMCFE},i} + x_i]_1$ where $[\boldsymbol{u}_\ell]_1 = \mathcal{H}_1(\ell)$.
  2. It re-computes $\mathsf{com}_{\mathsf{DMCFE},i}$ and a proof $\pi_{\mathsf{Encrypt},i}$ for the relation $\mathcal{R}_{\mathsf{Encrypt}}$ on input $(\ell, [c_{\ell,i}]_1, \mathsf{com}_{\mathsf{DMCFE},i}, m; x_i, \mathsf{ek}_i)$.
  3. It outputs the ciphertext $C_{\ell,i} = (\ell, [c_{\ell,i}]_1, \pi_{\mathsf{Encrypt},i})$.

– $\mathsf{DKeyGenShare}(\mathsf{sk}_i, \ell_{\boldsymbol{y}}, m, \mathsf{pk})$:
  1. It parses $\mathsf{sk}_i = (\boldsymbol{s}_{\mathsf{DMCFE},i}, \mathsf{LDSUM.sk}_i)$, $\ell_{\boldsymbol{y}} = (\ell_{\boldsymbol{y},b})_{b\in[2]}$ and $\mathsf{pk} = \mathsf{LDSUM.pk}$.
  2. It computes $\mathsf{dk}_i = (\mathsf{LDSUM.Encrypt}(\mathsf{LDSUM.sk}_i, s_{\mathsf{DMCFE},i,b} \cdot y_i, \ell_{\boldsymbol{y},b}))_{b\in[2]}$.
  3. It re-computes $\mathsf{com}_{\mathsf{DMCFE},i}$ and a proof $\pi_{\mathsf{DKeyGenShare},i}$ for the relation $\mathcal{R}_{\mathsf{DKeyGenShare}}$ on input $(\mathsf{LDSUM.pk}, \mathsf{com}_{\mathsf{DMCFE},i}, \mathsf{dk}_i, \ell_{\boldsymbol{y}}; \mathsf{sk}_i)$.
  4. It outputs the functional key share $\mathsf{dk}_{i,\boldsymbol{y}} = (\mathsf{dk}_i, \ell_{\boldsymbol{y}}, \pi_{\mathsf{DKeyGenShare},i})$.

– $\mathsf{VerifyCT}((C_{\ell,i})_{i\in[n]}, \mathsf{vk}_{\mathsf{CT}}, m)$:
  1. It parses $C_{\ell,i} = (\ell, [c_{\ell,i}]_1, \pi_{\mathsf{Encrypt},i})$ for $i \in [n]$, and $\mathsf{vk}_{\mathsf{CT}} = (\mathsf{com}_{\mathsf{DMCFE},i})_{i\in[n]}$.
  2. For $i \in [n]$: it verifies the proof $\pi_{\mathsf{Encrypt},i}$ for the relation $\mathcal{R}_{\mathsf{Encrypt}}$ on input $(\ell, [c_{\ell,i}]_1, \mathsf{com}_{\mathsf{DMCFE},i}, m)$.
  3. It outputs 1 for accepting if $\pi_{\mathsf{Encrypt},i}$ is valid for all $i \in [n]$, otherwise outputs 0 with the set $\mathcal{MS} = \{i : \pi_{\mathsf{Encrypt},i} \text{ is not valid}\}$ for rejecting.

– $\mathsf{VerifyDK}((\mathsf{dk}_{i,\boldsymbol{y}})_{i\in[n]}, \mathsf{vk}_{\mathsf{DK}})$:
  1. It parses the keys $\mathsf{dk}_{i,\boldsymbol{y}} = (\mathsf{dk}_i, \ell_{\boldsymbol{y}}, \pi_{\mathsf{DKeyGenShare},i})$ and $\mathsf{vk}_{\mathsf{DK}} = (\mathsf{LDSUM.pk}, (\mathsf{com}_{\mathsf{DMCFE},i})_{i\in[n]})$.

  2. From the function label $\ell_{\boldsymbol{y}}$, it verifies that $y_i \in [0, 2^m - 1]$ for $i \in [n]$. It stops and outputs 0 if $\boldsymbol{y}$ is not valid.
  3. For $i \in [n]$: it verifies $\pi_{\mathsf{DKeyGenShare},i}$ for the relation $\mathcal{R}_{\mathsf{DKeyGenShare}}$ on input $(\mathsf{LDSUM.pk}, \mathsf{com}_{\mathsf{DMCFE},i}, \mathsf{dk}_i, \ell_{\boldsymbol{y}})$.
  4. It outputs 1 for accepting if $\pi_{\mathsf{DKeyGenShare},i}$ is valid for all $i \in [n]$, otherwise outputs 0 with the set $\mathcal{MS} = \{i : \pi_{\mathsf{DKeyGenShare},i} \text{ is not valid}\}$ for rejecting.

– $\mathsf{DKeyComb}((\mathsf{dk}_{i,\boldsymbol{y}})_{i\in[n]}, \ell_{\boldsymbol{y}}, \mathsf{pk})$:
  1. It parses the keys $\mathsf{dk}_{i,\boldsymbol{y}} = (\mathsf{dk}_i, \ell_{\boldsymbol{y}}, \pi_{\mathsf{DKeyGenShare},i})$ and $\mathsf{pk} = \mathsf{LDSUM.pk}$.
  2. It outputs $[\mathsf{dk}_{\boldsymbol{y}}]_2 = (\mathsf{LDSUM.Decrypt}((\mathsf{dk}_{i,b})_{i\in[n]}, \ell_{\boldsymbol{y},b}, \mathsf{LDSUM.pk}))_{b\in[2]} \in \mathbb{G}_2^2$. In the LDSUM decryption, it is hard to obtain $\mathsf{dk}_{\boldsymbol{y}} \in \mathbb{Z}_p^2$ from $[\mathsf{dk}_{\boldsymbol{y}}]_2$, since $\mathsf{dk}_{\boldsymbol{y}}$ is random over $\mathbb{Z}_p^2$. Therefore, we stop the LDSUM decryption once obtaining $[\mathsf{dk}_{\boldsymbol{y}}]_2$.

– $\mathsf{Decrypt}(\boldsymbol{C}_\ell, [\mathsf{dk}_{\boldsymbol{y}}]_2)$: It gets $[\alpha]_T = \sum_{i\in[n]} e([c_{\ell,i}]_1, [y_i]_2) - e([\boldsymbol{u}_\ell]_1^\top, [\mathsf{dk}_{\boldsymbol{y}}]_2)$, and eventually solves the discrete logarithm in basis $[1]_T$ to return α.

The relation $\mathcal{R}_{\mathsf{Encrypt}}$ that guarantees a correct encryption of a valid input x_i under a committed encryption key $\boldsymbol{s}_{\mathsf{DMCFE},i}$ is defined as in Section 5.1. We can use $\mathsf{NIZK}_{\mathsf{Encrypt}}$ that is constructed as in Section 5.1 to prove this relation. Since the LDSUM scheme uses the ODSUM as a sub-protocol (see Section 5.2) in its key generation, we express all the terms $\mathsf{LDSUM.pk}$, $\mathsf{LDSUM.sk}_i$ explicitly as follows

– $\mathsf{LDSUM.pk} = ((T_{\mathsf{ODSUM},i})_{i\in[n]}, (\mathsf{dk}_{\mathsf{LDSUM},i})_{i\in[n]}) \in G^{2\times n} \times G^n$;
– $\mathsf{LDSUM.sk}_i = (\boldsymbol{s}_{\mathsf{LDSUM},i}, \boldsymbol{t}_{\mathsf{ODSUM},i}) \in \mathbb{Z}_p^2 \times \mathbb{Z}^2$.

The relation $\mathcal{R}_{\mathsf{DKeyGenShare}}$ is defined in Figure 1 and proved by $\mathsf{NIZK}_{\mathsf{DKeyGenShare}}$ in Figure 2. A remark is that the key $s_{\mathsf{LDSUM},i}$ is verified to be consistent in between $\mathsf{dk}_{\mathsf{LDSUM},i}$ (in class group) and dk_i (in pairing group), so we need a Σ-protocol that proves the DL equality between these two groups.

$\mathcal{R}_{\mathsf{DKeyGenShare}}$

Parameters: $i, \ell_{\boldsymbol{y}}, \ell_{\mathsf{DMCFE}}$
Statement: $(T_{\mathsf{ODSUM},j})_{j\in[n]}, \mathsf{dk}_{\mathsf{LDSUM},i}, \mathsf{com}_{\mathsf{DMCFE},i}, \mathsf{dk}_i$
Witness: $\boldsymbol{s}_{\mathsf{DMCFE},i}, \boldsymbol{s}_{\mathsf{LDSUM},i}, \boldsymbol{t}_{\mathsf{ODSUM},i}$
Relation:

1. $T_{\mathsf{ODSUM},i} = (g^{t_{\mathsf{ODSUM},i,1}}, g^{t_{\mathsf{ODSUM},i,2}})$
2. $\mathsf{dk}_{\mathsf{LDSUM},i} = (f^{s_{\mathsf{LDSUM},i,b}}(\prod_{i<j} T_{\mathsf{ODSUM},j,b} \cdot \prod_{i>j} T^{-1}_{\mathsf{ODSUM},j,b})^{t_{\mathsf{ODSUM},i,b}})_{b\in[2]}$
3. $\mathsf{com}_{\mathsf{DMCFE},i} = ([\boldsymbol{v}^\top_{\mathsf{DMCFE},b} \cdot \boldsymbol{s}_{\mathsf{DMCFE},i}]_1)_{b\in[2]}$ with $[\boldsymbol{v}_{\mathsf{DMCFE},b}]_1 = \mathcal{H}_1(\ell_{\mathsf{DMCFE},b})$
4. $\mathsf{dk}_i = ([\boldsymbol{u}^\top_{\ell_{\boldsymbol{y}},b} \boldsymbol{s}_{\mathsf{LDSUM},i} + s_{\mathsf{DMCFE},i,b} \cdot y_i]_2)_{b\in[2]}$ with $[\boldsymbol{u}_{\ell_{\boldsymbol{y}},b}]_2 = \mathcal{H}_2(\ell_{\boldsymbol{y},b})$

Fig. 1. The relation defines the correct generation of each functional key share

The above scheme is compatible with the definition of verifiable DMCFE in Section 4 by the following theorem.

Theorem 2. *The decentralized MCFE for inner product scheme described in Section 5.3 has correctness and verifiability for range predicates in the random oracle, as in Definition 8. More precisely,*

$$\mathsf{Adv}^{\mathsf{verif}}_{\mathsf{DMCFE}}(t, q_C, q_F) \leq q_C \cdot \max\{\mathsf{Adv}^{\mathsf{snd}}_{\mathit{NIZK}_{\mathsf{Encrypt}}}(t), q_F \cdot \mathsf{Adv}^{\mathsf{snd}}_{\mathit{NIZK}_{\mathsf{DKeyGenShare}}}(t)\}$$

where

– $\mathsf{Adv}^{\mathsf{verif}}_{\mathsf{DMCFE}}(t, q_c, q_F)$ *is the best advantage of any PPT adversary running in time t against the verifiability game in Definition 8 with q_C corruption queries and q_F functions for the finalization phase;*
– $\mathsf{Adv}^{\mathsf{snd}}_{\mathit{NIZK}_{\mathsf{Encrypt}}}(t)$ *is the best advantage of any PPT adversary running in time t against the soundness of $\mathit{NIZK}_{\mathsf{Encrypt}}$.*
– $\mathsf{Adv}^{\mathsf{snd}}_{\mathit{NIZK}_{\mathsf{DKeyGenShare}}}(t)$ *is the best advantage of any PPT adversary running in time t against the soundness of $\mathit{NIZK}_{\mathsf{DKeyGenShare}}$.*

Proof. We start with the correctness and then with the range-verifiability.

Σ-protocol for the relation $\mathcal{R}_{\mathsf{DKeyGenShare}}$:

Parameters: $(\mathcal{PG}, \mathcal{H}_1, \mathcal{H}_2)$, $(\hat{G}, F, g_p, f, S, p)$, $(i, \ell_{\boldsymbol{y}}, \ell_{\mathsf{DMCFE}})$

Statement: $(T_{\mathsf{ODSUM},j})_{j\in[n]}, \mathsf{dk}_{\mathsf{LDSUM},i}, \mathsf{com}_{\mathsf{DMCFE},i}, \mathsf{dk}_i$

Witness: $\boldsymbol{s}_{\mathsf{DMCFE},i}, \boldsymbol{s}_{\mathsf{LDSUM},i}, \boldsymbol{t}_{\mathsf{ODSUM},i}$

Output: 1 if $\mathcal{V}$ accepts, and 0 otherwise.

Protocol:

- $\mathcal{V}$ verifies that $\mathsf{dk}_{\mathsf{LDSUM},i}, T_{\mathsf{ODSUM},j} \in \hat{G}^2$ for $j \in [n]$.
- $\mathcal{P}$ commits the randomness:
  1. $\boldsymbol{\rho}_{\mathsf{LDSUM},i}, \boldsymbol{\rho}_{\mathsf{DMCFE},i} \xleftarrow{\$} \mathbb{Z}_p^2$, $\boldsymbol{\rho}_{\mathsf{ODSUM},i} \xleftarrow{\$} [0, 2^{\lambda} pS]^2$
  2. $R_{\mathsf{ODSUM},i} = (g_p^{\rho_{\mathsf{ODSUM},i,b}})_{b\in[2]}$
  3. $R_{\mathsf{dk}_i} = ([\boldsymbol{u}_{\ell_{\boldsymbol{y}},b}^{\top} \boldsymbol{\rho}_{\mathsf{LDSUM},i} + \rho_{\mathsf{DMCFE},i,b} \cdot y_i]_2)_{b\in[2]}$
  4. $R_{\mathsf{DMCFE},i} = ([\boldsymbol{v}_{\mathsf{DMCFE},b}^{\top} \cdot \boldsymbol{\rho}_{\mathsf{DMCFE},i}]_1)_{b\in[2]}$
  5. Let $K_{\Sigma,i} := \prod_{i<j} T_{\mathsf{ODSUM},j,b} \cdot (\prod_{i>j} T_{\mathsf{ODSUM},j,b})^{-1} \in G$, then $R_{\mathsf{dk}_{\mathsf{LDSUM},i}} = (f^{\rho_{\mathsf{LDSUM},i,b}} K_{\Sigma,i}^{\rho_{\mathsf{ODSUM},i,b}})_{b\in[2]}$
- $\mathcal{P}$ sends $(R_{\mathsf{ODSUM},i}, R_{\mathsf{dk}_i}, R_{\mathsf{DMCFE},i}, R_{\mathsf{dk}_{\mathsf{LDSUM},i}})$ to $\mathcal{V}$
- $\mathcal{V}$ chooses a random challenge $\alpha \xleftarrow{\$} [0, p-1]$ and sends it to $\mathcal{P}$.
- $\mathcal{P}$ computes the response:
  1. $\boldsymbol{z}_{\mathsf{ODSUM},i} = \alpha \cdot \boldsymbol{t}_{\mathsf{ODSUM},i} + \boldsymbol{\rho}_{\mathsf{ODSUM},i} \in \mathbb{Z}$
  2. $\boldsymbol{z}_{\mathsf{LDSUM},i} = \alpha \cdot \boldsymbol{s}_{\mathsf{LDSUM},i} + \boldsymbol{\rho}_{\mathsf{LDSUM},i} \in \mathbb{Z}_p$
  3. $\boldsymbol{z}_{\mathsf{DMCFE},i} = \alpha \cdot \boldsymbol{s}_{\mathsf{DMCFE},i} + \boldsymbol{\rho}_{\mathsf{DMCFE},i} \in \mathbb{Z}_p$
- $\mathcal{P}$ sends $(\boldsymbol{z}_{\mathsf{ODSUM},i}, \boldsymbol{z}_{\mathsf{LDSUM},i}, \boldsymbol{z}_{\mathsf{DMCFE},i})$ to $\mathcal{V}$.
- $\mathcal{V}$ verifies that:
  1. $\boldsymbol{z}_{\mathsf{ODSUM},i} \in [0, (2^{\lambda}+1)pS]^2$ and $\boldsymbol{z}_{\mathsf{LDSUM},i}, \boldsymbol{z}_{\mathsf{DMCFE},i} \in \mathbb{Z}_p^2$
  2. $T_{\mathsf{ODSUM},i}^{\alpha} \cdot R_{\mathsf{ODSUM},i} = (g_p^{z_{\mathsf{ODSUM},i,b}})_{b\in[2]}$
  3. $\alpha \cdot \mathsf{dk}_i + R_{\mathsf{dk}_i} = ([\boldsymbol{u}_{\ell_{\boldsymbol{y}},b}^{\top} \boldsymbol{z}_{\mathsf{LDSUM},i} + z_{\mathsf{DMCFE},i,b} \cdot y_i]_2)_{b\in[2]}$
  4. $\alpha \cdot \mathsf{com}_{\mathsf{DMCFE},i} + R_{\mathsf{DMCFE},i} = ([\boldsymbol{v}_{\mathsf{DMCFE},b}^{\top} \cdot \boldsymbol{z}_{\mathsf{DMCFE},i}]_1)_{b\in[2]}$
  5. Let $K_{\Sigma,i} := \prod_{i<j} T_{\mathsf{ODSUM},j,b} \cdot (\prod_{i>j} T_{\mathsf{ODSUM},j,b})^{-1} \in G$, then $\mathsf{dk}_{\mathsf{LDSUM},i}^{\alpha} \cdot R_{\mathsf{dk}_{\mathsf{LDSUM},i}} = (f^{z_{\mathsf{LDSUM},i,b}} K_{\Sigma,i}^{z_{\mathsf{ODSUM},i,b}})_{b\in[2]}$
- $\mathcal{V}$ rejects and stops the protocol if any check is invalid, and accepts otherwise.

Fig. 2. Note that $S = 2^{\lambda-2} \cdot \tilde{s}$ and $\mathcal{D}_p$ is a uniform distribution over $\{0, ..., S\}$. We let $\boldsymbol{\rho}_{\mathsf{ODSUM},i}$ be uniform in $[0, 2^{\lambda} pS]$ to obtain statistical zero-knowledge as in [GPS06].

Correctness. Given a range predicate for input and for inner product function $[0, 2^m - 1]$, a vector $\boldsymbol{y}$ such that $y_i \in [0, 2^m - 1]$, an n-vector plaintext $\boldsymbol{x}$ such that $x_i \in [0, 2^m - 1]$. We consider the following case

$$\begin{cases} \mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda) \\ \big((\mathsf{sk}_i, \mathsf{ek}_i)_{i \in [n]}, \mathsf{vk}_{\mathsf{CT}}, \mathsf{vk}_{\mathsf{DK}}, \mathsf{pk}\big) \leftarrow \mathsf{KeyGen}() \\ C_{\ell,i} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell, m) \forall i \in [n] \\ \mathsf{dk}_{i,\boldsymbol{y}} \leftarrow \mathsf{DKeyGenShare}(\mathsf{sk}_i, \ell_{\boldsymbol{y}}, m) \forall i \in [n] \end{cases}$$

Parse $C_{\ell,i} = (\ell, [c_{\ell,i}]_1, \pi_{\mathsf{Encrypt},i})$ and $\mathsf{dk}_{i,\boldsymbol{y}} = (\mathsf{dk}_i, \ell_{\boldsymbol{y}}, \pi_{\mathsf{DKeyGenShare},i})$, by the completeness of $\mathsf{NIZK}_{\mathsf{Encrypt}}$ and $\mathsf{NIZK}_{\mathsf{DKeyGenShare}}$ respectively, we have that all $\pi_{\mathsf{Encrypt},i}$ and $\pi_{\mathsf{DKeyGenShare},i}$ are respectively valid. Therefore,

$$\mathsf{VerifyCT}((C_{\ell,i})_{i \in [n]}, \mathsf{vk}_{\mathsf{CT}}, m) = \mathsf{VerifyDK}((\mathsf{dk}_{i,\boldsymbol{y}})_{i \in [n]}, \mathsf{vk}_{\mathsf{DK}}) = 1.$$

For the decryption, we parse $\mathsf{dk}_{i,\boldsymbol{y}} = (\mathsf{dk}_i, \ell_{\boldsymbol{y}}, \pi_{\mathsf{DKeyGenShare},i})$ and $\mathsf{pk} = \mathsf{LDSUM.pk}$. By the correctness of LDSUM (Section 5.2) and the fact that LDSUM stops and outputs $[\mathsf{dk}_{\boldsymbol{y},b}]_2$ before computing the discrete logarithm, we have

$$\begin{aligned}[\mathsf{dk}_{\boldsymbol{y}}]_2 &= (\mathsf{LDSUM.Decrypt}((\mathsf{dk}_{i,b})_{i \in [n]}, \ell_{\boldsymbol{y},b}, \mathsf{LDSUM.pk}))_{b \in [2]} \\ &= ([\sum_{i \in [n]} s_{\mathsf{DMCFE},i,b} \cdot y_i]_2)_{b \in [2]}.\end{aligned}$$

Then we have $[\alpha]_T$ equal to

$$\begin{aligned}&\sum_{i \in [n]} e([c_{\ell,i}]_1, [y_i]_2) - e([\boldsymbol{u}_\ell]_1^\top, [\mathsf{dk}_{\boldsymbol{y}}]_2) \\ &= \sum_{i \in [n]} \left[(\boldsymbol{u}_\ell^\top \boldsymbol{s}_{\mathsf{DMCFE},i} + x_i) \cdot y_i\right]_T - \left[\boldsymbol{u}_\ell^\top \cdot (\sum_{i \in [n]} \boldsymbol{s}_{\mathsf{DMCFE},i} \cdot y_i)\right]_T = \left[\sum_{i \in [n]} x_i \cdot y_i\right]_T\end{aligned}$$

As the inner product $\sum_{i \in [n]} x_i \cdot y_i$ is small, computing α can be done efficiently.

Verifiability. We suppose that there exists a PPT adversary $\mathcal{A}$ that can win the verifiability game in Definition 8 with a non-negligible probability. Without loss of generality, the range predicate for input and inner product function can be fixed to be $[0, 2^m - 1]$.

Except using a trivial attack, $\mathcal{A}$ cannot win the game by making other honest senders accused of sending invalid ciphertexts or invalid functional key shares (the second winning condition). Indeed, to accuse an honest sender $\mathcal{S}_i$, $\mathcal{A}$ has to broadcast some malicious share that makes the proof of correct generation for $\mathcal{S}_i$'s ciphertext or for $\mathcal{S}_i$'s functional key share invalid. By the design of the scheme, the only broadcast elements among senders and the receiver are the ODSUM public keys $(T_{\mathsf{ODSUM},j})_{j \in [n]}$ (included in $\mathsf{LDSUM.pk}$), which are not used in the relation $\mathcal{R}_{\mathsf{Encrypt}}$. For the relation $\mathcal{R}_{\mathsf{DKeyGenShare}}$, the condition involving $(T_{\mathsf{ODSUM},j})_{j \in [n], j \neq i}$ is the generation of $\mathsf{dk}_{\mathsf{LDSUM},i}$, which only requires $(T_{\mathsf{ODSUM},j})_{j \in [n], j \neq i}$ to be group elements in class group $\hat{G}$. Therefore, sending an incorrect group-encoding $T_{\mathsf{ODSUM},j}$ can make the generation and then the proof fail. However, this is a trivial attack and can be excluded, as each $T_{\mathsf{ODSUM},j}$ can be efficiently verified to be in group by the public (and by the verifier in Figure 2 also) and the public will already know it is the corrupted sender $\mathcal{S}_j$ who broadcast a malicious share.

We now consider $\mathcal{A}$ that wins the game by winning the first condition. We let $(\mathsf{vk}_{\mathsf{CT}}, \mathsf{vk}_{\mathsf{DK}}, \mathsf{pk}, \ell, (C_{\ell,i})_{i \in \mathcal{MS}_\mathcal{A}}, (\mathsf{dk}_{\boldsymbol{y}_j,i})_{j,i \in \mathcal{MS}_\mathcal{A}})$ be the transcript that makes $\mathcal{A}$ win the game. In this case we have

$$\mathsf{VerifyCT}((C_{\ell,i})_{i \in [n]}, \mathsf{vk}_{\mathsf{CT}}, m) = \mathsf{VerifyDK}((\mathsf{dk}_{i,\boldsymbol{y}})_{i \in [n]}, \mathsf{vk}_{\mathsf{DK}}) = 1.$$

Suppose that the transcript output by $\mathcal{A}$ satisfies the relations $\mathcal{R}_{\mathsf{DKeyGenShare}}$ and $\mathcal{R}_{\mathsf{Encrypt}}$.

- From $\mathcal{R}_{\mathsf{DKeyGenShare}}$, $(\mathsf{vk}_{\mathsf{CT}}, \mathsf{vk}_{\mathsf{DK}}, \mathsf{pk})$ is generated from KeyGen with secret keys $\mathsf{sk}_i = (\boldsymbol{s}_{\mathsf{DMCFE},i}, (\boldsymbol{s}_{\mathsf{LDSUM},i}, \boldsymbol{t}_{\mathsf{ODSUM},i}))$ and encryption keys $\mathsf{ek}_i = \boldsymbol{s}_{\mathsf{DMCFE},i}$. For each $i \in [n]$ and each inner product function $\boldsymbol{y}_j$, $\mathsf{dk}_{\boldsymbol{y}_j,i}$ is generated from $\mathsf{DKeyGenShare}$ on input $(\mathsf{sk}_i, \ell_{\boldsymbol{y}}, \mathsf{pk})$.

– From $\mathcal{R}_{\mathsf{Encrypt}}$, $C_{\ell,i}$ is generated from $\mathsf{Encrypt}$ on input a message $x_i \in [0, 2^m - 1]$ and an encryption key $\boldsymbol{s'}_{\mathsf{DMCFE},i}$ for each $i \in [n]$.

We model the hash function $\mathcal{H}_1$ as a random oracle onto $\mathbb{G}^2$. Then $\mathsf{com}_{\mathsf{DMCFE},i}$ is perfectly binding. From above, we have $\boldsymbol{s}_{\mathsf{DMCFE},i} = \boldsymbol{s'}_{\mathsf{DMCFE},i}$. By the proved correctness of the scheme, the decryption process with input $\boldsymbol{C}_\ell = (C_{\ell,i})_{i\in[n]}$ and $[\mathsf{dk}_{\boldsymbol{y}}]_2 = \mathsf{DKeyComb}((\mathsf{dk}_{\boldsymbol{y}_j,i})_{i\in[n]}, \ell_{\boldsymbol{y}}, \mathsf{pk})$ will output the inner product $\langle \boldsymbol{x}, \boldsymbol{y}_j \rangle$ for all vectors $\boldsymbol{y}_j$. This contradicts the first winning condition, so $\mathcal{A}$ must break either the soundness of $\mathsf{NIZK}_{\mathsf{Encrypt}}$ or the soundness of $\mathsf{NIZK}_{\mathsf{DKeyGenShare}}$ with the same probability of winning the game.

If $\mathcal{A}$ wins the game by breaking the soundness $\mathsf{NIZK}_{\mathsf{Encrypt}}$ with a non-negligible probability, an adversary $\mathcal{B}$ against the soundness of $\mathsf{NIZK}_{\mathsf{Encrypt}}$ can be constructed as follows: $\mathcal{B}$ plays as a challenger in the game with $\mathcal{A}$, after $\mathcal{A}$ finalized the game, $\mathcal{B}$ guesses an index i from the corrupted set $\mathcal{MS}_{\mathcal{A}}$ and outputs the instance $(\ell, [c_{\ell,i}]_1, \mathsf{com}_{\mathsf{DMCFE},i}, \pi^i_{\mathsf{Encrypt}}, m)$ from $\mathcal{A}$'s transcript. Given q_C corrupted senders, in the case $\mathcal{A}$ wins the game, the probability that $\mathcal{B}$ breaks the soundness of $\mathsf{NIZK}_{\mathsf{Encrypt}}$ is $\frac{1}{q_C}$. Similarly for $\mathsf{NIZK}_{\mathsf{DKeyGenShare}}$, given q_F inner-product functions $\boldsymbol{y}_j$ to be finalized, $\mathcal{B}$ outputs one in $n \cdot q_F$ instances of $((T_{\mathsf{ODSUM},i})_{i\in[n]}, \mathsf{dk}_{\mathsf{LDSUM},i}, \mathsf{com}_{\mathsf{DMCFE},i}, \mathsf{dk}_i, \boldsymbol{y}_j)$ from $\mathcal{A}$, which incurs a security loss of $q_C \cdot q_F$. To finalize, we have

$$\mathsf{Adv}^{\mathsf{verif}}_{\mathsf{DMCFE}}(\mathcal{A}, t, q_C, q_F) \leq q_C \cdot \max\{\mathsf{Adv}^{\mathsf{snd}}_{\mathsf{NIZK}_{\mathsf{Encrypt}}}(t), q_F \cdot \mathsf{Adv}^{\mathsf{snd}}_{\mathsf{NIZK}_{\mathsf{DKeyGenShare}}}(t)\}.$$

As $\mathsf{Adv}^{\mathsf{snd}}_{\mathsf{NIZK}_{\mathsf{Encrypt}}}(t)$ and $\mathsf{Adv}^{\mathsf{snd}}_{\mathsf{NIZK}_{\mathsf{DKeyGenShare}}}(t)$ are negligible, and q_C and q_F are polynomially bounded, the proof is complete.

5.4 Indistinguishability Security

Theorem 3. *The Range-Verifiable DMCFE for Inner Product scheme described in Section 5.3 is* $\mathtt{sta} - \mathtt{IND}$*-secure under the SXDH and HSM assumptions, as in Definition 9.*

The proof is provided in Appendix B.5.

5.5 Efficiency Analysis

We assume that $\mathsf{NIZK}_{\mathsf{Encrypt}}$ is instantiated with the Σ-protocol $\mathsf{NIZK}_{\mathsf{key}}$ and the Bulletproof for range [BBB+18] (for the relation $\mathcal{R}_{\mathsf{range}}$ in Section 5.1). As far as we know, Bulletproof offers better efficiency for batch verification than other transparent-setup non-interactive range proof schemes. Since the scalar operation in $\mathbb{Z}_p$ is cheap compared to the group exponentiation, we do not detail them here. Let n be the number of senders and m be a binary upper bound of a input range $[0, 2^m - 1]$.

– **Proving time**: $\mathsf{NIZK}_{\mathsf{Encrypt}}$ costs about $12m + 17$ exponentiations with $O(m)$ scalar operations, while $\mathsf{NIZK}_{\mathsf{DKeyGenShare}}$ costs 16 exponentiations with $O(1)$ scalar operations.
– **Proof size**: each $\pi_{\mathsf{Encrypt},i}$ has the size of $2\log_2\lceil m \rceil + 7$ group elements and 10 scalars, while each $\pi_{\mathsf{DKeyGenShare}}$ has the size of 8 group elements and 6 scalars.
– **Verifying time**: $\mathsf{NIZK}_{\mathsf{Encrypt}}$ costs about a single multi-exponentiation of size $2m + 2\log_2\lceil m \rceil + 19$ with $O(m)$ scalar operations for each ciphertext, while $\mathsf{NIZK}_{\mathsf{DKeyGenShare}}$ costs 24 exponentiations for each key share.

For a practical parameter, one can have $n = 2^{10}$ and $m \leq 16$. Then the costs for functional key share are even more efficient than those for ciphertext. Compared to the DMCFE in [CDG+18], the overhead costs from verifiability for each sender and each receiver asymptotically depend only on m (range proof costs). Therefore, the approach of verifying each functional key share, which has the advantage of identifying up to all n malicious senders in a non-interactive manner, is no more prohibitively expensive in our scheme.

On the other hand, this approach is avoided in [BGL+22] for the ACORN-robust protocol since each key share for sum can not be verified efficiently. Their alternative approach requires the help of checking from $\log(n)$ neighboring senders, which incurs more interaction during verification and overhead costs additionally depending on $\log(n)$ (besides range proof costs) for each sender. Their approach also assumes that at most $1/3$ number of senders misbehave.

6 Discussions

6.1 Batch Verification

In our inner-product DMCFE scheme in Section 5.3, if a receiver wants to verify that the combined functional decryption key $\mathsf{dk}_{\boldsymbol{y}}$ is generated correctly with respect to a vector $\boldsymbol{y}$, there is a more efficient way than verifying each sender's functional key share. The receiver can directly check the following equalities

$$e(\sum_{i=1}^{n} \mathsf{com}_{\mathsf{DMCFE},i,b} \cdot y_i, [1]_2) = e([\boldsymbol{v}_{\mathsf{DMCFE},b}^{\top}]_1, [\mathsf{dk}_b]_2)$$

for $b \in [2]$. When $\mathsf{dk}_{\boldsymbol{y}}$ is correct, the above equalities are equivalent to

$$\left[\sum_{i\in[n]} (\boldsymbol{v}_{\mathsf{DMCFE},b}^{\top} \cdot \boldsymbol{s}_{\mathsf{DMCFE},i}) \cdot y_i\right]_T = \left[\boldsymbol{v}_{\mathsf{DMCFE},b}^{\top} \cdot (\sum_{i\in[n]} \boldsymbol{s}_{\mathsf{DMCFE},i} \cdot y_i)\right]_T$$

for $b \in [2]$. If any equality does not hold, then $\mathsf{dk}_{\boldsymbol{y}}$ is maliciously generated. This verification has perfect soundness under the condition that $(\boldsymbol{v}_{\mathsf{DMCFE},b})_{b\in[2]} \in \mathbb{Z}_p^{2\times 2}$ are linearly independent. The verification time is $2n$ exponentiations and 6 pairings compared to $24n$ exponentiations for verifying each of n key shares. In a hybrid use, a receiver can first use this quick verification to see if $\mathsf{dk}_{\boldsymbol{y}}$ is correct. If it is not the case, he can continue to verify each key share to identify malicious senders.

Similarly, for batch verification of n independent ciphertexts under the same label for a range $[0, 2^m - 1]$, $\mathsf{NIZK}_{\mathsf{Encrypt}}$ when instantiated with Bulletproof costs about 3 multi-exponentiations of size $3 + 2n$, a multi-exponentiation of size $2m + 3 + n(2\log_2 \lceil m \rceil + 5)$, and $\mathcal{O}(n \cdot m)$ scalar operations.

6.2 Privacy Improvement with AoNE

The All-or-Nothing Encapsulation AoNE in [CDSG+20] is an encryption which guarantees that a receiver can reveal either *all* encrypted messages under the same label of senders by collecting all their ciphertexts, or *nothing*. By adding a AoNE encryption layer on DSUM or DMCFE ciphertexts, the leakage from incomplete ciphertexts can be ruled out. Due to space constraints, we put an heuristic of applying AoNE to the verifiable DMCFE scheme while still preserving the efficiency of malicious sender identification in Appendix C.

6.3 Perspectives

Natural questions from our work include improving the static security of the verifiable DMCFE scheme for inner product and allowing dynamic join of new users as in [CDSG+20]. Furthermore, obtaining practical overhead costs from verifiability for function-hiding DMCFE schemes is an interesting direction.

Acknowledgements. This work was supported in part by the France 2030 ANR Project ANR-22-PECY-003 SecureCompute and by Beyond5G, a project funded by the French government as part of the plan "France Relance".

References

ABG19. M. Abdalla, F. Benhamouda, and R. Gay. From single-input to multi-client inner-product functional encryption. In *ASIACRYPT 2019, Part III*, *LNCS* 11923, pages 552–582. Springer, Heidelberg, December 2019.

ABKW19. M. Abdalla, F. Benhamouda, M. Kohlweiss, and H. Waldner. Decentralizing inner-product functional encryption. In *PKC 2019, Part II*, *LNCS* 11443, pages 128–157. Springer, Heidelberg, April 2019.

AGM18. S. Agrawal, C. Ganesh, and P. Mohassel. Non-interactive zero-knowledge proofs for composite statements. In *CRYPTO 2018, Part III*, *LNCS* 10993, pages 643–673. Springer, Heidelberg, August 2018.

ALS16. S. Agrawal, B. Libert, and D. Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In *CRYPTO 2016, Part III*, *LNCS* 9816, pages 333–362. Springer, Heidelberg, August 2016.

BBB$^+$18. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

BFM88. M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988.

BGJS16. S. Badrinarayanan, V. Goyal, A. Jain, and A. Sahai. Verifiable functional encryption. In *ASIACRYPT 2016, Part II*, *LNCS* 10032, pages 557–587. Springer, Heidelberg, December 2016.

BGL$^+$22. J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun. ACORN: Input validation for secure aggregation. Cryptology ePrint Archive, Report 2022/1461, 2022. `https://eprint.iacr.org/2022/1461`.

BSW11. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC 2011*, *LNCS* 6597, pages 253–273. Springer, Heidelberg, March 2011.

CCL$^+$19. G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 191–221. Springer, Heidelberg, August 2019.

CCL$^+$20. G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Bandwidth-efficient threshold EC-DSA. In *PKC 2020, Part II*, *LNCS* 12111, pages 266–296. Springer, Heidelberg, May 2020.

CDG$^+$18. J. Chotard, E. Dufour Sans, R. Gay, D. H. Phan, and D. Pointcheval. Decentralized multi-client functional encryption for inner product. In *ASIACRYPT 2018, Part II*, *LNCS* 11273, pages 703–732. Springer, Heidelberg, December 2018.

CDSG$^+$20. J. Chotard, E. Dufour-Sans, R. Gay, D. H. Phan, and D. Pointcheval. Dynamic decentralized functional encryption. In *CRYPTO 2020, Part I*, *LNCS* 12170, pages 747–775. Springer, Heidelberg, August 2020.

CGKR22. G. Couteau, D. Goudarzi, M. Klooß, and M. Reichle. Sharp: Short relaxed range proofs. In *ACM CCS 2022*, pages 609–622. ACM Press, November 2022.

CKLR21. G. Couteau, M. Klooß, H. Lin, and M. Reichle. Efficient range proofs with transparent setup from bounded integer commitments. In *EUROCRYPT 2021, Part III*, *LNCS* 12698, pages 247–277. Springer, Heidelberg, October 2021.

CL15. G. Castagnos and F. Laguillaumie. Linearly homomorphic encryption from DDH. In *CT-RSA 2015*, *LNCS* 9048, pages 487–505. Springer, Heidelberg, April 2015.

CLT18. G. Castagnos, F. Laguillaumie, and I. Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In *ASIACRYPT 2018, Part II*, *LNCS* 11273, pages 733–764. Springer, Heidelberg, December 2018.

FLS90. U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990.

FS87. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, *LNCS* 263, pages 186–194. Springer, Heidelberg, August 1987.

GGG$^+$14. S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou. Multi-input functional encryption. In *EUROCRYPT 2014*, *LNCS* 8441, pages 578–602. Springer, Heidelberg, May 2014.

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

GPS06. M. Girault, G. Poupard, and J. Stern. On the fly authentication and signature schemes based on groups of unknown order. *Journal of Cryptology*, 19(4):463–487, October 2006.

LT19. B. Libert and R. Titiu. Multi-client functional encryption for linear functions in the standard model from LWE. In *ASIACRYPT 2019, Part III*, *LNCS* 11923, pages 520–551. Springer, Heidelberg, December 2019.

PS96. D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT'96*, *LNCS* 1070, pages 387–398. Springer, Heidelberg, May 1996.

SG98. V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *EUROCRYPT'98*, *LNCS* 1403, pages 1–16. Springer, Heidelberg, May / June 1998.

A More Definitions

A.1 Additional Assumptions in Class Groups

To prove the soundness of our Σ-protocol in class group, we need the following additional assumptions in [CCL$^+$20].

Definition 10 (Low Order Assumption). *Consider a security parameter $\lambda \in \mathbb{N}$, and $\gamma \in \mathbb{N}$. The γ-low order problem (LOP_γ) is $(t(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$-secure for* Gen *if, given the output of* Gen*, no algorithm $\mathcal{A}$ running in time $\leq t(\lambda)$ can output a γ-low order element in $\hat{G}$ with probability greater than $\epsilon_{\mathsf{LO}}(\lambda)$. More precisely,*

$$\epsilon_{\mathsf{LO}}(\lambda) := \Pr\left[\begin{matrix}(\tilde{s}, f, \hat{g}_p, F, \hat{G}) \leftarrow \mathsf{Gen}(1^\lambda, p) \\ (\mu, d) \leftarrow \mathcal{A}(\tilde{s}, f, \hat{g}_p, \hat{G}, F)\end{matrix} : \mu^d = 1, 1 \neq \mu \in \hat{G}, 1 < d < \gamma\right]$$

Definition 11 (Strong Root Assumption for Class Groups). *Consider a security parameter $\lambda \in \mathbb{N}$, and let $\mathcal{A}$ be a PPT adversary. We run* Gen *from on input $(1^\lambda, p)$ to get $(\tilde{s}, f, \hat{g}_p, \hat{G}, F)$ and we give this output and a random $Y \in \langle \hat{g}_p \rangle$ as an input to $\mathcal{A}$. We say that $\mathcal{A}$ solves the strong root problem for class groups (*SRP*) if $\mathcal{A}$ outputs a positive integer $e \neq 2^k$ for all k and $X \in \hat{G}$, such that $Y = X^e$. In particular, the* SRP *is $(t(\lambda), \epsilon_{\mathsf{SR}}(\lambda))$-secure for* Gen *if any adversary $\mathcal{A}$, running in time $\leq t(\lambda)$, solves the* SRP *with probability at most $\epsilon_{\mathsf{SR}}(\lambda)$.*

A.2 Commitments

Definition 12 (Commitment). *A non-interactive commitment scheme* com *over a message space $\mathcal{M}_{\mathsf{com}}$, a commitment space $\mathcal{C}_{\mathsf{com}}$ and a opening space $\mathcal{O}_{\mathsf{com}}$ is defined by a tuple of three algorithms* (SetUp, Commit, Verify)*:*

- SetUp(λ)*: Takes as input a security parameter λ , outputs public parameters* pp *(which are implicit to other algorithms);*
- Commit(m)*: Takes as input a message $m \in \mathcal{M}_{\mathsf{com}}$, generates a uniformly random $r \in \mathcal{O}_{\mathsf{com}}$. Outputs a commitment $c \in \mathcal{C}_{\mathsf{com}}$ and the opening value r.*
- Verify(c, r, m)*: Takes as input a commitment c, an opening r and a message m. Verifies if c is a commitment to m with the opening r. Outputs $b \in \{0, 1\}$.*

Definition 13 (Hiding Commitment). *A commitment scheme* com *is said to be hiding if for any PPT adversary $\mathcal{A}$, there is a negligible function $\mu(\lambda)$ such that*

$$\left|\Pr\left[\begin{matrix}\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda), \quad (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}), \\ b \xleftarrow{\$} \{0,1\} \quad c \leftarrow \mathsf{Commit}(m_b), \quad b' \xleftarrow{\$} \mathcal{A}(\mathsf{st}, c)\end{matrix} : b = b'\right] - \frac{1}{2}\right| \leqslant \mu(\lambda)$$

where the probability is over random coins in SetUp*, $\mathcal{A}$,* Commit *and in choosing b. The commitment scheme is said to be perfectly hiding if $\mu(\lambda) = 0$.*

Definition 14 (Binding Commitment). *A commitment scheme* com *is said to be binding if for any PPT adversary $\mathcal{A}$, there is a negligible function $\mu(\lambda)$ such that*

$$\Pr\left[\begin{matrix}\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda), \\ (c, m, m', r, r') \leftarrow \mathcal{A}(\mathsf{pp})\end{matrix} : \begin{matrix}\mathsf{Verify}(c, r, m) = \mathsf{Verify}(c, r', m') = 1 \\ \wedge (m \neq m')\end{matrix}\right] \leqslant \mu(\lambda)$$

where the probability is over random coins in SetUp*, $\mathcal{A}$, and* Commit*. The commitment scheme is said to be perfectly binding if $\mu(\lambda) = 0$.*

Definition 15 (Pedersen Commitment). *. Let $\mathcal{M}_{\mathsf{com}} = \mathcal{O}_{\mathsf{com}} = \mathbb{Z}_p$ and $\mathcal{C}_{\mathsf{com}} = \mathbb{G}$ of order p.*

- SetUp*: Outputs $[h], [g] \xleftarrow{\$} \mathbb{G}$.*
- Commit(m)*: Outputs $r \xleftarrow{\$} \mathbb{Z}_p$ and $c = [g] \cdot m + [h] \cdot r$.*
- Verify(c, r, m)*: Outputs 1 if $c = [g] \cdot m + [h] \cdot r$, and 0 otherwise.*

The Pedersen commitment is perfectly hiding and computationally binding under the discrete logarithm assumption.

A.3 Non-interactive Zero-Knowledge Proofs

Definition 16 (NIZK Argument of Knowledge [AGM18]). *A NIZK argument of knowledge for an NP relation $\mathcal{R}$ is a NIZK argument for $\mathcal{R}$ with the following additional extractability property:*

Computational Extraction. For any PPT adversary $\mathcal{A}$, random string $r \xleftarrow{\$} \{0,1\}^*$, there exists a PPT algorithm K such that there is a negligible function $\mu(\lambda)$:

$$\Pr\left[\begin{array}{l}\sigma \leftarrow \mathsf{SetUp}(\lambda),\\ (u,\pi) \leftarrow \mathcal{A}(\sigma; r)\\ w \leftarrow K(\sigma, u, \pi; r)\end{array} : \begin{array}{c}\mathsf{Verify}(u,\pi)=1\\ \wedge\, \mathcal{R}(u;w)=0\end{array}\right] \leq \mu(\lambda).$$

K is called a knowledge extractor of $\mathcal{A}$.

Definition 17 (Non-interactive Range Proof). *Given a non-interactive commitment scheme* $\mathsf{com} = (\mathsf{SetUp}, \mathsf{Commit}, \mathsf{Verify})$*, a non-interactive range proof over* com *is a NIZK argument of knowledge for the following relation $\mathcal{R}_{\mathsf{range}}$:*

$$\mathcal{R}_{\mathsf{range}}((\mathsf{com}, l, r); (m, r)) = 1 \leftrightarrow \mathsf{com} = \mathsf{Commit}(m; r) \wedge l \leq m \leq r$$

Fiat-Shamir Transformation. Fiat-Shamir heuristic [FS87] is used to transforming an interactive zero-knowledge argument of knowledge scheme to a non-interactive one. The original scheme must have the property of being public-coin, i.e. verifier's random coins are independent of the prover's messages. All random challenges are replaced by hashes of the transcript up to that point, including the statement itself. The transformed NIZK argument is sound (or knowledge sound) and zero-knowledge in the random oracle model.

B A Range-Verifiable MCFE for Inner Product (More)

B.1 Σ-Protocol for Encryption Key Validation

In this part, we describe the Σ-protocol $\mathsf{NIZK}_{\mathsf{key}}$ for $\mathcal{R}_{\mathsf{key}}$ in Section 5.1. For the sake of clarity, we describe the scheme and prove the properties in its interactive mode.
Given public parameters $([\boldsymbol{u}], [\boldsymbol{v}], [\boldsymbol{v'}]) \in (\mathbb{G}^2)^3$. Let $\mathcal{P}$ and $\mathcal{V}$ be respectively the prover and the verifier in an argument for the relation $\mathcal{R}_{\mathsf{key}}$:

$$\mathcal{R}_{\mathsf{key}}(\mathsf{com}_{\mathsf{Ped}}, \mathsf{com}_{\mathsf{ek}}; \boldsymbol{s}, x) = 1 \leftrightarrow \mathsf{com}_{\mathsf{Ped}} = [\boldsymbol{u}^\top] \cdot \boldsymbol{s} + [x] \wedge \mathsf{com}_{\mathsf{ek}} = ([\boldsymbol{v}^\top] \cdot \boldsymbol{s}, [\boldsymbol{v'}^\top] \cdot \boldsymbol{s})$$

$\mathcal{P}(\mathsf{com}_{\mathsf{Ped}}, \mathsf{com}_{\mathsf{ek}}; \boldsymbol{s}, x)$		$\mathcal{V}(\mathsf{com}_{\mathsf{Ped}}, \mathsf{com}_{\mathsf{ek}})$
Commits the randomness:		
$r_x, r, r' \xleftarrow{\$} \mathbb{Z}_p$		
$\boldsymbol{r} := (r, r')$		
$R = [\boldsymbol{v}^\top] \cdot \boldsymbol{r}$		
$R' = [\boldsymbol{v'}^\top] \cdot \boldsymbol{r}$		
$R_x = [\boldsymbol{u}^\top] \cdot \boldsymbol{r} + [r_x]$	$\xrightarrow{R, R', R_x}$	
	$\xleftarrow{\alpha}$	$\alpha \xleftarrow{\$} \mathbb{Z}_p$
Computes the responses:		
$\boldsymbol{t} = \boldsymbol{s} \cdot \alpha + \boldsymbol{r}$		
$t_x = x \cdot \alpha + r_x$	$\xrightarrow{\boldsymbol{t}, t_x}$	Verifies the following equalities:
		$\alpha \cdot \mathsf{com}_{\mathsf{Ped}} + R_x = [\boldsymbol{u}^\top] \cdot \boldsymbol{t} + [t_x]$
		$\alpha \cdot \mathsf{com}_{\mathsf{ek},1} + R = [\boldsymbol{v}^\top] \cdot \boldsymbol{t}$
		$\alpha \cdot \mathsf{com}_{\mathsf{ek},2} + R' = [\boldsymbol{v'}^\top] \cdot \boldsymbol{t}$
		If all equalities hold,
		returns 1 for accepting.
		Otherwise,
		returns 0 for rejecting.

Completeness. In an honest execution, one has

$$\begin{aligned}
\alpha \cdot \mathsf{com}_{\mathsf{Ped}} + R_x &= \alpha \cdot ([\boldsymbol{u}^\top] \cdot \boldsymbol{s} + [x]) + ([\boldsymbol{u}^\top] \cdot \boldsymbol{r} + [r_x]) \\
&= [\boldsymbol{u}^\top] \cdot (\boldsymbol{s} \cdot \alpha + \boldsymbol{r}) + [x \cdot \alpha + r_x] \\
&= [\boldsymbol{u}^\top] \cdot \boldsymbol{t} + [t_x], \\
\alpha \cdot \mathsf{com}_{\mathsf{ek},1} + R &= \alpha \cdot ([\boldsymbol{v}^\top] \cdot \boldsymbol{s}) + [\boldsymbol{v}^\top] \cdot \boldsymbol{r} \\
&= [\boldsymbol{v}^\top] \cdot (\boldsymbol{s} \cdot \alpha + \boldsymbol{r}) \\
&= [\boldsymbol{v}^\top] \cdot \boldsymbol{t}, \\
\alpha \cdot \mathsf{com}_{\mathsf{ek},2} + R' &= \alpha \cdot ([\boldsymbol{v}'^\top] \cdot \boldsymbol{s}) + [\boldsymbol{v}'^\top] \cdot \boldsymbol{r} \\
&= [\boldsymbol{v}'^\top] \cdot (\boldsymbol{s} \cdot \alpha + \boldsymbol{r}) \\
&= [\boldsymbol{v}'^\top] \cdot \boldsymbol{t}
\end{aligned}$$

Then $\mathcal{V}(\mathsf{com}_{\mathsf{Ped}}, \mathsf{com}_{\mathsf{ek}}) = 1$ with probability 1.

Soundness. Assume that $(R, R', R_x, \alpha_1, \boldsymbol{t}_1, t_{x,1})$ and $(R, R', R_x, \alpha_2, \boldsymbol{t}_2, t_{x,2})$ are two accepting transcripts such that $\alpha_1 \neq \alpha_2$. Then one has

$$\begin{cases}
\alpha_1 \cdot \mathsf{com}_{\mathsf{ek},1} + R &= [\boldsymbol{v}^\top] \cdot \boldsymbol{t}_1 \\
\alpha_2 \cdot \mathsf{com}_{\mathsf{ek},1} + R &= [\boldsymbol{v}^\top] \cdot \boldsymbol{t}_2,
\end{cases}
\begin{cases}
\alpha_1 \cdot \mathsf{com}_{\mathsf{ek},2} + R' &= [\boldsymbol{v}'^\top] \cdot \boldsymbol{t}_1 \\
\alpha_2 \cdot \mathsf{com}_{\mathsf{ek},2} + R' &= [\boldsymbol{v}'^\top] \cdot \boldsymbol{t}_2,
\end{cases}
\begin{cases}
\alpha_1 \cdot \mathsf{com}_{\mathsf{Ped}} + R_x &= [\boldsymbol{u}^\top] \cdot \boldsymbol{t}_1 + [t_{x,1}] \\
\alpha_2 \cdot \mathsf{com}_{\mathsf{Ped}} + R_x &= [\boldsymbol{u}^\top] \cdot \boldsymbol{t}_2 + [t_{x,2}]
\end{cases}$$

which respectively implies that

$$\begin{aligned}
\mathsf{com}_{\mathsf{ek},1} &= [\boldsymbol{v}^\top] \cdot ((\alpha_1 - \alpha_2)^{-1} \cdot (\boldsymbol{t}_1 - \boldsymbol{t}_2)), \\
\mathsf{com}_{\mathsf{ek},2} &= [\boldsymbol{v}'^\top] \cdot ((\alpha_1 - \alpha_2)^{-1} \cdot (\boldsymbol{t}_1 - \boldsymbol{t}_2)), \\
\mathsf{com}_{\mathsf{Ped}} &= [\boldsymbol{u}^\top] \cdot ((\alpha_1 - \alpha_2)^{-1} \cdot (\boldsymbol{t}_1 - \boldsymbol{t}_2)) + (\alpha_1 - \alpha_2)^{-1} \cdot [t_{x,1} - t_{x,2}].
\end{aligned}$$

Then, $\boldsymbol{s} = (\alpha_1 - \alpha_2)^{-1} \cdot (\boldsymbol{t}_1 - \boldsymbol{t}_2)$ and $x = (\alpha_1 - \alpha_2)^{-1} \cdot [t_{x,1} - t_{x,2}]$ is a valid witness. Therefore, there exists a PPT extractor $\mathcal{E}$ that takes two valid transcripts $(R, R', R_x, \alpha_1, \boldsymbol{t}_1, t_{x,1})$ and $(R, R', R_x, \alpha_2, \boldsymbol{t}_2, t_{x,2})$ and produces a valid witness $(\boldsymbol{s}, x)$ in polynomial time.

Zero-Knowledge. On input a challenge α and a statement $(\mathsf{com}_{\mathsf{Ped}}, \mathsf{com}_{\mathsf{ek}})$, a simulator $\mathcal{S}$ runs as follows:

- Choose $\boldsymbol{t}^0 \xleftarrow{\$} \mathbb{Z}_p^2$ and $t_x^0 \xleftarrow{\$} \mathbb{Z}_p$.
- Computes $R_x^0 \leftarrow [\boldsymbol{u}^\top] \cdot \boldsymbol{t}^0 + [t_x^0] - \alpha \cdot \mathsf{com}_{\mathsf{Ped}}$.
- Computes $R^0 \leftarrow [\boldsymbol{v}^\top] \cdot \boldsymbol{t}^0 - \alpha \cdot \mathsf{com}_{\mathsf{ek},1}$.
- Computes $R'^0 \leftarrow [\boldsymbol{v}'^\top] \cdot \boldsymbol{t}^0 - \alpha \cdot \mathsf{com}_{\mathsf{ek},2}$.
- Outputs the transcript $(R_x^0, R^0, R'^0, \alpha, \boldsymbol{t}^0, t_x^0)$.

On the other hand, an accepting transcript from an honest execution $(R, R', R_x, \alpha, \boldsymbol{t}, t_x)$ has $\boldsymbol{t}$ uniformly chosen from $\mathbb{Z}_p^2$ and t_x uniformly chosen from $\mathbb{Z}_p$. In both transcripts, (R, R', R_x) and (R^0, R'^0, R_x^0) are determined by $(\boldsymbol{t}, t_x)$ and $(\boldsymbol{t}^0, t_x^0)$ respectively. Then the distributions of two transcripts are the same.

To transform the above schemes into non-interactive mode, one can apply the Fiat-Shamir heuristic.

B.2 Combination between Range Proof and Σ-protocol

Lemma 1. *On public parameters $([\boldsymbol{u}], ([\boldsymbol{u}^\top_{\ell_{\mathsf{MCFE},b}}])_{b\in[2]})$, for any input $([c], \mathsf{com}_{\mathsf{ek}}, l, r)$, the composition of $\mathsf{NIZK}_{\mathsf{range}}$ on the statement $([c], l, r)$ and $\mathsf{NIZK}_{\mathsf{key}}$ on the statement $([c], \mathsf{com}_{\mathsf{ek}})$, which is denoted by $\mathsf{NIZK}_{\mathsf{Encrypt}}$, is a zero-knowledge argument for the language L_{Encrypt}, defined by the relation $\mathcal{R}_{\mathsf{Encrypt}}$, on the statement $([c], \mathsf{com}_{\mathsf{ek}}, l, r)$.*

Proof. We note that a transcript of $\mathsf{NIZK}_{\mathsf{Encrypt}}$ is accepting if and only if it consists of an accepting transcript for $\mathsf{NIZK}_{\mathsf{range}}$ and an accepting transcript for $\mathsf{NIZK}_{\mathsf{key}}$.

- *Completeness*: The completeness comes from the completeness of $\mathsf{NIZK}_{\mathsf{key}}$ and $\mathsf{NIZK}_{\mathsf{range}}$.
- *Soundness*: A knowledge extractor $K^{\mathcal{P}}_{\mathsf{Encrypt}}$ for $\mathcal{P}$ is constructed as follows:
  1. It takes as input $([c], \mathsf{com}_{\mathsf{ek}}, l, r)$.
  2. As both $\mathsf{NIZK}_{\mathsf{range}}$ and $\mathsf{NIZK}_{\mathsf{key}}$ have knowledge extractors, it calls the extractor $K^{\mathcal{P}}_{\mathsf{range}}$ on input $([c], l, r)$ and the extractor $K^{\mathcal{P}}_{\mathsf{key}}$ on input $([c], \mathsf{com}_{\mathsf{ek}})$.
  3. When $K^{\mathcal{P}}_{\mathsf{range}}([c], l, r) = (\boldsymbol{s}, x)$ and $K^{\mathcal{P}}_{\mathrm{Schnorr}}([c], l, r) = (\boldsymbol{s'}, x')$, it outputs $(\boldsymbol{s}, x)$.

  Note that $\boldsymbol{s} = \boldsymbol{s'}$ since $\mathsf{com}_{\mathsf{ek}}$ is perfectly binding. Then we have $x = x'$.
  Therefore, $(\boldsymbol{s}, x) = (\boldsymbol{s'}, x')$ is a valid witness for the relation $\mathcal{R}_{\mathsf{Encrypt}}$. This implies that $([c], \mathsf{com}_{\mathsf{ek}}, l, r) \in L_{\mathsf{Encrypt}}$. On the other hand, since $K^{\mathcal{P}}_{\mathsf{range}}$ and $K^{\mathcal{P}}_{\mathrm{Schnorr}}$ are PPT algorithms, then $K^{\mathcal{P}}_{\mathsf{Encrypt}}$ runs in polynomial time. The existence of a knowledge extractor $K^{\mathcal{P}}_{\mathsf{Encrypt}}$ implies the soundness of the protocol.
- *Zero-Knowledge*: Since $\mathsf{NIZK}_{\mathsf{range}}$ and $\mathsf{NIZK}_{\mathsf{key}}$ are both zero-knowledge, so they have S_{range} and S_{key} as simulators respectively. The simulator S_{Encrypt} can output the concatenation $(S_{\mathsf{range}}([c], l, r), S_{\mathsf{key}}([c], \mathsf{com}_{\mathsf{ek}}))$ as the simulated transcript. This simulated transcript is then indistinguishable from the transcript of an honest execution. S_{range} and S_{key} run in polynomial time, so S_{Encrypt} also runs in polynomial time.

B.3 Correctness and Security Analysis of the LDSUM scheme

Correctness. Given $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$, $\big((\mathsf{sk}_i)_{i\in[n]}, \mathsf{pk}\big) \leftarrow \mathsf{KeyGen}()$, and $C_{\ell,i} \leftarrow \mathsf{Encrypt}(\mathsf{sk}_i, x_i, \ell)$ for $i \in [n]$. By the correctness of ODSUM, we have

$$\mathsf{dk}_{\mathbf{1}} = \sum_{i\in[n]} \boldsymbol{s}_i \in \mathbb{Z}_p^2.$$

Then we have

$$[\alpha] = \sum_i [c_i] - [\boldsymbol{u}_\ell^\top] \cdot \mathsf{dk}_{\mathbf{1}} = \sum_i [\boldsymbol{u}_\ell^\top \boldsymbol{s}_i + x_i] - [\boldsymbol{u}_\ell^\top \cdot \sum_{i\in[n]} \boldsymbol{s}_i] = [\sum_i x_i].$$

Given the condition that $\sum_i x_i$ is sufficiently small, then α can be found efficiently.

Theorem 4. *The LDSUM scheme described in Section 5.2 is sta-IND-secure (see Definition 7) under the DDH and HSM assumptions, in the random oracle model. More precisely, we have*

$$\mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{LDSUM}}(t, q_E) \leq \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{ODSUM}}(t) + \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{MCFE}}(t, q_E).$$

where

- $\mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{LDSUM}}(t, q_E)$ *is the best advantage of any PPT adversary running in time t with q_E encryption queries against the* `sta−IND` *security game of the LDSUM scheme;*
- $\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{ODSUM}}(t)$ *is the best advantage of any PPT adversary running in time t against the* `IND`*-security game of the ODSUM scheme;*
- $\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{MCFE}}(t, q_E)$ *is the best advantage of any PPT adversary running in time t with q_E encryption queries against the* `IND`*-security game of the MCFE scheme [CDG+18].*

Proof. We proceed by using a hybrid argument. Let $\mathcal{A}$ be a PPT adversary running in time t with q_E encryption queries. For any game $\mathbf{G}$, we write $\mathsf{Adv}_{\mathbf{G}}$ the advantage of $\mathcal{A}$ in the game $\mathbf{G}$.

Game $\mathbf{G}_0$: this is the `sta − IND` security game as given in Definition 7, with the set $\mathcal{CS}$ of corrupted senders known from the beginning. Let $\mathcal{HS}$ be the set of non-corrupted senders.

Game $\mathbf{G}_1$: this game is as $\mathbf{G}_0$, except for the KeyGen process in the initialization phase:
- if i is the last non-corrupted index, then the challenger computes

$$\mathsf{dk}_{i,b} = \mathsf{ODSUM.Encrypt}(s_{i,b} + \sum_{j\in\mathcal{HS},j\neq i} s_{j,b}, \mathsf{ODSUM.pk}_b, t_{i,b})$$

  for $b \in [2]$;
- for other non-corrupted index i, the challenger computes

$$\mathsf{dk}_{i,b} = \mathsf{ODSUM.Encrypt}(0, \mathsf{ODSUM.pk}_b, t_{i,b})$$

  for $b \in [2]$;

and answers $\mathsf{dk}_i = (\mathsf{dk}_{i,1}, \mathsf{dk}_{i,2})$. By the (static) $\mathtt{IND}$-security of the ODSUM scheme, we have

$$|\mathsf{Adv}_{\mathbf{G}_0} - \mathsf{Adv}_{\mathbf{G}_1}| \leq \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{ODSUM}}(t).$$

We now reduce the $\mathtt{IND}$-security of the MCFE scheme in [CDG+18] to the game in $\mathbf{G}_1$. We construct an adversary $\mathcal{B}_{\mathsf{MCFE}}$ against the security of the MCFE scheme as in Figure 3.
From the reduction, $\mathsf{Adv}_{\mathbf{G}_1} \leq \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{MCFE}}(t, q_E)$. To complete the proof, we have

$$\mathsf{Adv}_{\mathbf{G}_0} \leq \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{ODSUM}}(t) + \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{MCFE}}(t, q_E).$$

B.4 Σ-protocol in class groups for VerifyDK

Theorem 5. *The protocol $\mathsf{NIZK}_{\mathsf{DKeyGenShare}}$, as defined in Figure 2, is a zero-knowledge argument for the relation $\mathcal{R}_{\mathsf{DKeyGenShare}}$. More specifically, this protocol has perfect completeness, statistically zero-knowledge, and a computational soundness under the Strong Root Assumption and the p-Low Order Assumption.*

Proof. We prove the completeness, the soundness and the zero-knowledge property in the interactive mode.

Completeness. In an honest execution, one has

$$\begin{aligned}
T^{\alpha}_{\mathsf{ODSUM},i} \cdot R_{\mathsf{ODSUM},i} &= (g_p^{\alpha t_{\mathsf{ODSUM},i,b}} \cdot g_p^{\rho_{\mathsf{ODSUM},i,b}})_{b\in[2]} \\
&= (g_p^{z_{\mathsf{ODSUM},i,b}})_{b\in[2]} \\
\alpha \cdot \mathsf{dk}_i + R_{\mathsf{dk}_i} &= ([\boldsymbol{u}^{\top}_{\ell_{\boldsymbol{y}},b} \alpha \boldsymbol{s}_{\mathsf{LDSUM},i} + \alpha s_{\mathsf{DMCFE},i,b} \cdot y_i]_2 \\
&\qquad + [\boldsymbol{u}^{\top}_{\ell_{\boldsymbol{y}},b} \boldsymbol{\rho}_{\mathsf{LDSUM},i} + \rho_{\mathsf{DMCFE},i,b} \cdot y_i]_2)_{b\in[2]} \\
&= ([\boldsymbol{u}^{\top}_{\ell_{\boldsymbol{y}},b} \boldsymbol{z}_{\mathsf{LDSUM},i} + z_{\mathsf{DMCFE},i,b} \cdot y_i]_2)_{b\in[2]} \\
\alpha \cdot \mathsf{com}_{\mathsf{DMCFE},i} + R_{\mathsf{DMCFE},i} &= ([\boldsymbol{v}^{\top}_{\mathsf{DMCFE},b} \cdot \alpha \boldsymbol{s}_{\mathsf{DMCFE},i}]_1 + [\boldsymbol{v}^{\top}_{\mathsf{DMCFE},b} \cdot \boldsymbol{\rho}_{\mathsf{DMCFE},i}]_1)_{b\in[2]} \\
&= ([\boldsymbol{v}^{\top}_{\mathsf{DMCFE},b} \cdot \boldsymbol{z}_{\mathsf{DMCFE},i}]_2)_{b\in[2]} \\
\alpha \cdot \mathsf{dk}_{\mathsf{LDSUM},i} + R_{\mathsf{dk}_{\mathsf{LDSUM},i}} &= (f^{\alpha s_{\mathsf{LDSUM},i,b}} K_{\Sigma,i}^{\alpha t_{\mathsf{ODSUM},i,b}} \cdot f^{\rho_{\mathsf{LDSUM},i,b}} K_{\Sigma,i}^{\rho_{\mathsf{ODSUM},i,b}})_{b\in[2]} \\
&= (f^{z_{\mathsf{LDSUM},i,b}} K_{\Sigma,i}^{z_{\mathsf{ODSUM},i,b}})_{b\in[2]}
\end{aligned}$$

Then $\mathcal{V}$ accepts with probability 1.

Soundness. Given two accepting transcripts

$$\begin{aligned}
&(R_{\mathsf{ODSUM},i}, R_{\mathsf{dk}_i}, R_{\mathsf{DMCFE},i}, R_{\mathsf{dk}_{\mathsf{LDSUM},i}}, \alpha, \boldsymbol{z}_{\mathsf{ODSUM},i}, \boldsymbol{z}_{\mathsf{LDSUM},i}, \boldsymbol{z}_{\mathsf{DMCFE},i}); \\
&(R_{\mathsf{ODSUM},i}, R_{\mathsf{dk}_i}, R_{\mathsf{DMCFE},i}, R_{\mathsf{dk}_{\mathsf{LDSUM},i}}, \alpha', \boldsymbol{z}'_{\mathsf{ODSUM},i}, \boldsymbol{z}'_{\mathsf{LDSUM},i}, \boldsymbol{z}'_{\mathsf{DMCFE},i})
\end{aligned}$$

with $\alpha \neq \alpha'$, by putting $\delta_\alpha := \alpha - \alpha'$, one can obtain the witness $\boldsymbol{s}_{\mathsf{DMCFE},i}$ and $\boldsymbol{s}_{\mathsf{LDSUM},i}$ as in a Schnorr's protocol over a standard DDH group:

$$\boldsymbol{s}_{\mathsf{DMCFE},i} = \delta_\alpha^{-1}(\boldsymbol{z}_{\mathsf{DMCFE},i} - \boldsymbol{z}'_{\mathsf{DMCFE},i}) \in \mathbb{Z}_p^2; \qquad \boldsymbol{s}_{\mathsf{LDSUM},i} = \delta_\alpha^{-1}(\boldsymbol{z}_{\mathsf{LDSUM},i} - \boldsymbol{z}'_{\mathsf{LDSUM},i}) \in \mathbb{Z}_p^2$$

Reduction from $\mathsf{IND}-\mathsf{MCFE}$ to $\mathbf{G}_1$:
$\mathcal{B}_{\mathsf{MCFE}}$ calls $\mathcal{A}$ and plays as the challenger in $\mathbf{G}_1$.
$\mathcal{A}$ sends the corruption queries to $\mathcal{B}_{\mathsf{MCFE}}$ before the initialization. $\mathcal{B}_{\mathsf{MCFE}}$ then sends the same corruption queries to the `IND`-security game of MCFE.
After the output $b'_{\mathcal{A}} \leftarrow \mathcal{A}^{\mathsf{QEncrypt}(\cdot,\cdot,\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{QDKeyGen}(\cdot)}$, $\mathcal{B}_{\mathsf{MCFE}}$ outputs $b'_{\mathcal{B}_{\mathsf{MCFE}}} \leftarrow b'_{\mathcal{A}}$.

Initialization:

- SetUp (λ): On receiving the MCFE public parameters $\mathsf{MCFE.pp} = (\mathcal{G}, \mathcal{H})$, adversary $\mathcal{B}_{\mathsf{MCFE}}$ generates $\mathsf{ODSUM.pp} = \mathsf{ODSUM.SetUp}(\lambda)$.
- KeyGen $()$: $\mathcal{B}_{\mathsf{MCFE}}$ generates $(t_{i,b}, T_{i,b}, \mathsf{ODSUM.pk}_b) \leftarrow \mathsf{ODSUM.KeyGen}()$ for $b \in [2]$. The generation of key shares for the sum is as follows
  - If i is the last non-corrupted index, $\mathcal{B}_{\mathsf{MCFE}}$ first sends a key query for the sum, namely $\mathsf{MCFE.QDKeyGen}(\mathbf{1})$, to obtain $\mathsf{dk}_{\mathbf{1}} := (\sum_{j\in\mathcal{CS}} s_{j,b} + \sum_{j\in\mathcal{HS}} s_{j,b})_{b\in[2]} \in \mathbb{Z}_p^2$. Then $\mathcal{B}_{\mathsf{MCFE}}$ obtains $\boldsymbol{s}_j \leftarrow \mathsf{MCFE.QCorrupt}(j)$ for all $j \in \mathcal{CS}$ and computes
  $$\mathsf{dk}_{i,b} = \mathsf{ODSUM.Encrypt}(\mathsf{dk}_{\mathbf{1},b} - \sum_{j\in\mathcal{CS}} s_{j,b}, \mathsf{ODSUM.pk}_b, t_{i,b})$$
  for $b \in [2]$;
  - For other non-corrupted index i, the challenger computes
  $$\mathsf{dk}_{i,b} = \mathsf{ODSUM.Encrypt}(0, \mathsf{ODSUM.pk}_b, t_{i,b})$$
  for $b \in [2]$;
  - If i is a corrupted index, the challenger computes
  $$\mathsf{dk}_{i,b} = \mathsf{ODSUM.Encrypt}(s_{i,b}, \mathsf{ODSUM.pk}_b, t_{i,b})$$
  for $b \in [2]$.

Returns $\mathsf{pp} = (\mathsf{MCFE.pp}, \mathsf{ODSUM.pp})$ and $\mathsf{pk} = ((\mathsf{ODSUM.pk}_b)_{b\in[2]}, (\mathsf{dk}_i)_{i\in[n]})$.

$\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$:
Returns $C_{\ell,i} \leftarrow \mathsf{MCFE.QEncrypt}(i, x_i^0, x_i^1, \ell)$

$\mathsf{QCorrupt}(i)$:
$\boldsymbol{s}_i \leftarrow \mathsf{MCFE.QCorrupt}(i)$
Returns $(\boldsymbol{s}_i, t_{i,1}, t_{i,2})$.

Fig. 3. Reduction for the proof of Theorem 4. $\mathcal{A}$ is the adversary in game $\mathbf{G}_1$, while $\mathcal{B}_{\mathsf{MCFE}}$ is the adversary in the `IND`-security game for the MCFE scheme in [CDG+18].

The extracted value $\boldsymbol{s}_{\mathsf{DMCFE},i}$ is the opening to the commitment $\mathsf{com}_{\mathsf{DMCFE},i}$. In addition, $\boldsymbol{s}_{\mathsf{DMCFE},i}$ and $\boldsymbol{s}_{\mathsf{LDSUM},i}$ are witness for the functional key share dk_i.
For the public input in class group G, it suffices to prove that with an overwhelming probability, there exists a witness $\boldsymbol{t}_{\mathsf{ODSUM},i}$ satisfying the relations in $\mathcal{R}_{\mathsf{DKeyGenShare}}$. From the accepting transcripts, one has the following equalities:

$$T^{\delta_\alpha}_{\mathsf{ODSUM},i} = (g_p^{z_{\mathsf{ODSUM},i,b} - z'_{\mathsf{ODSUM},i,b}})_{b\in[2]};$$
$$\mathsf{dk}^{\delta_\alpha}_{\mathsf{LDSUM},i} = (f^{z_{\mathsf{LDSUM},i,b} - z'_{\mathsf{LDSUM},i,b}} K_{\Sigma,i}^{z_{\mathsf{ODSUM},i,b} - z'_{\mathsf{ODSUM},i,b}})_{b\in[2]}$$

Following a similar argument of extraction as in [CCL+20], we first put

$$\delta_{\mathsf{ODSUM},i,b} := z_{\mathsf{ODSUM},i,b} - z'_{\mathsf{ODSUM},i,b}, \qquad \delta_{\mathsf{LDSUM},i,b} := z_{\mathsf{LDSUM},i,b} - z'_{\mathsf{LDSUM},i,b},$$
$$d_b = \gcd(\delta_{\mathsf{ODSUM},i,b}, \delta_\alpha)$$

for $b \in [2]$. Note that $d_b < p$ and $d_b \neq 0$. From the last equalities, we have

$$\boldsymbol{v}_1 := (g_p^{\frac{\delta_{\mathsf{ODSUM},i,b}}{d_b}} \cdot T_{\mathsf{ODSUM},i,b}^{-\frac{\delta_\alpha}{d_b}})_{b\in[2]};$$
$$\boldsymbol{v}_2 := (f^{\delta_{\mathsf{LDSUM},i,b}\cdot d_b^{-1}} K_{\Sigma,i}^{\frac{\delta_{\mathsf{ODSUM},i,b}}{d_b}} \mathsf{dk}_{\mathsf{LDSUM},i,b}^{-\frac{\delta_\alpha}{d_b}})_{b\in[2]}$$

Here we use the fraction symbol to distinguish between division in $\mathbb{Z}_p$ and in $\mathbb{Z}$. In the case $\boldsymbol{v}_1 = \boldsymbol{v}_2 = (1_{\hat{G}}, 1_{\hat{G}})$ and $\frac{\delta_\alpha}{d_b} = 2^{\mu_b}$ for some $\mu_b \in \mathbb{N}$ and $b \in [2]$, as the verifier already checked that $T_{\mathsf{ODSUM},i}, \mathsf{dk}_{\mathsf{LDSUM},i}, K_{\Sigma,i} \in \hat{G}^2$ and the group order of $\hat{G}$ is odd, one obtain the following mathematical fact

$$(g_p^{\frac{\delta_{\mathsf{ODSUM},i,b}}{d_b}\cdot 2^{-\mu_b}})_{b\in[2]} = (T_{\mathsf{ODSUM},i,b})_{b\in[2]};$$
$$(f^{\delta_{\mathsf{LDSUM},i,b}\cdot\delta_\alpha^{-1}} K_{\Sigma,i}^{\frac{\delta_{\mathsf{ODSUM},i,b}}{d_b}\cdot 2^{-\mu_b}})_{b\in[2]} = (\mathsf{dk}_{\mathsf{LDSUM},i,b})_{b\in[2]}.$$

Therefore, the existence of a valid $\boldsymbol{t}_{\mathsf{ODSUM},i} := (\frac{\delta_{\mathsf{ODSUM},i,b}}{d_b} \cdot 2^{-\mu_b} \mod s_p)_{b\in[2]}$ where s_p is the order of G^p is implicitly implied in this case. The statement is correct with respect to the relation $\mathcal{R}_{\mathsf{DKeyGenShare}}$.
For the other cases, we provide an analysis as below:

- Either $\boldsymbol{v}_1 \neq (1_{\hat{G}}, 1_{\hat{G}})$ or $\boldsymbol{v}_2 \neq (1_{\hat{G}}, 1_{\hat{G}})$: WLOG, we assume that $v_{1,b} \neq 1_{\hat{G}}$. Since $v_{1,b}^{d_b} = 1$ and $d_b | \delta_\alpha < p$. Then the extractor finds a solution $(v_{1,b}, d_b)$ to the p-Low Order Assumption in $\hat{G}$, which happens with a negligible probability.
- $\boldsymbol{v}_1 = \boldsymbol{v}_2 = (1_{\hat{G}}, 1_{\hat{G}})$ but there exists a $b \in [2]$ such that $\frac{\delta_\alpha}{d_b} \neq 2^{\mu_b}$ for all $\mu_b \in \mathbb{N}$: WLOG, we assume that $b = 1$ in this case. One can find $(u_1, v_1) \in \mathbb{Z}^2$ such that $u_1 \cdot \delta_{\mathsf{ODSUM},i,1} + v_1 \cdot \delta_\alpha = d_1$. Then one has
$$g_p^{d_1} = g_p^{u_1\cdot\delta_{\mathsf{ODSUM},i,1} + v_1\cdot\delta_\alpha} = (T^{u_1}_{\mathsf{ODSUM},i} \cdot g_p^{v_1})^{\delta_\alpha}.$$
We denote that $h := g_p^{-1}(T^{u_1}_{\mathsf{ODSUM},i} \cdot g_p^{v_1})^{\frac{\delta_\alpha}{d_1}}$, then $h^{d_1} = 1_{\hat{G}}$. If $h \neq 1_{\hat{G}}$, then (h, d_1) is a solution to the p-Low Order Problem. If $h = 1_{\hat{G}}$, then $(T^{u_1}_{\mathsf{ODSUM},i} \cdot g_p^{v_1}, \frac{\delta_\alpha}{d_1})$ is a not-power-of-2 root of g_p, and hence a solution to the Strong Root Problem with a randomized instance g_p. Therefore, this case happens with a negligible probability.

From above, we can conclude that given two accepting transcripts with two different challenges, an extractor has an overwhelming probability of extracting valid $(\boldsymbol{s}_{\mathsf{DMCFE},i}, \boldsymbol{s}_{\mathsf{LDSUM},i})$ and values $(\delta_{\mathsf{ODSUM},i}, (d_b, \mu_b)_{b\in[2]})$ that imply the existence of a valid $\boldsymbol{t}_{\mathsf{ODSUM},i}$. This extractor can run in polynomial time. Therefore, the soundness holds under the Strong Root Assumption and the p-Low Order Assumption.

Zero-Knowledge. Given a statement that satisfies $\mathcal{R}_{\mathsf{DKeyGenShare}}$:

$$(T_{\mathsf{ODSUM},j,b})_{j\in[n],b\in[2]}, \mathsf{dk}_{\mathsf{LDSUM},i}, \mathsf{com}_{\mathsf{DMCFE},i}, \mathsf{dk}_i,$$

a simulator chooses

- a challenge $\alpha \xleftarrow{\$} \mathbb{Z}_p$;
- responses $\boldsymbol{z}_{\mathsf{LDSUM},i}, \boldsymbol{z}_{\mathsf{DMCFE},i} \xleftarrow{\$} \mathbb{Z}_p^2$;
- a response $\boldsymbol{z}_{\mathsf{ODSUM},i} \xleftarrow{\$} [pS, 2^\lambda pS]^2$;

and computes

- $R_{\mathsf{ODSUM},i} = (T^{-\alpha}_{\mathsf{ODSUM},i,b} \cdot g_p^{z_{\mathsf{ODSUM},i,b}})_{b\in[2]}$;
- $R_{\mathsf{dk}_i} = ([\boldsymbol{u}^\top_{\ell_{\boldsymbol{y}},b} \boldsymbol{z}_{\mathsf{LDSUM},i} + z_{\mathsf{DMCFE},i,b} \cdot y_i]_2 - \alpha \cdot \mathsf{dk}_{i,b})_{b\in[2]}$;
- $R_{\mathsf{DMCFE},i} = ([\boldsymbol{v}^\top_{\mathsf{DMCFE},b} \cdot \boldsymbol{z}_{\mathsf{DMCFE},i}]_1 - \alpha \cdot \mathsf{com}_{\mathsf{DMCFE},i,b})_{b\in[2]}$;
- $R_{\mathsf{dk}_{\mathsf{LDSUM},i}} = (f^{z_{\mathsf{LDSUM},i,b}} K^{z_{\mathsf{ODSUM},i,b}}_{\Sigma,i} \cdot \mathsf{dk}^{-\alpha}_{\mathsf{LDSUM},i,b})_{b\in[2]}$
  where $K_{\Sigma,i} = \prod_{i<j} T_{\mathsf{ODSUM},j,b} \cdot (\prod_{i>j} T_{\mathsf{ODSUM},j,b})^{-1} \in G$.

and outputs $(R_{\mathsf{ODSUM},i}, R_{\mathsf{dk}_i}, R_{\mathsf{DMCFE},i}, R_{\mathsf{dk}_{\mathsf{LDSUM},i}}, \alpha, \boldsymbol{z}_{\mathsf{ODSUM},i}, \boldsymbol{z}_{\mathsf{LDSUM},i}, \boldsymbol{z}_{\mathsf{DMCFE},i})$ as the simulated transcript. For the part in standard DDH group, namely
$(R_{\mathsf{dk}_i}, R_{\mathsf{DMCFE},i}, \boldsymbol{z}_{\mathsf{LDSUM},i}, \boldsymbol{z}_{\mathsf{DMCFE},i})$, the indistinguishability from those of an honest execution is implied from the perfect zero-knowledge property of Schnorr's protocol. On the other hand, by having $\frac{pS}{2^\lambda pS}$ negligible in λ, then we achieve the statistical zero-knowledge for the part of transcript in unknown order group, namely $(R_{\mathsf{ODSUM},i}, R_{\mathsf{dk}_{\mathsf{LDSUM},i}}, \boldsymbol{z}_{\mathsf{ODSUM},i})$, from Theorem 2 in [GPS06].

B.5 Theorem 3 (Security for Verifiable DMCFE)

The Range-Verifiable Decentralized MCFE for Inner Product scheme described in Section 5.3 is $\mathtt{sta-IND}$*-secure under the SXDH and HSM assumptions, as in Definition 9. More precisely, we have*

$$\mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{DMCFE}}(t, q_E) \leq q_K \mathsf{Adv}^{\mathsf{zk}}_{\mathit{NIZK}_{\mathsf{DKeyGenShare}}}(t) + q_E \mathsf{Adv}^{\mathsf{zk}}_{\mathit{NIZK}_{\mathsf{Encrypt}}}(t) + \mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{LDSUM}}(t, q_K) + \mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{MCFE}}(t, q_E + 2n).$$

where

- $\mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{DMCFE}}(t, q_E, q_K)$ *is the best advantage of any PPT adversary running in time t with q_E encryption queries and q_K key share queries against the* $\mathtt{IND}$*-security game of the verifiable* DMCFE *scheme;*
- $\mathsf{Adv}^{\mathsf{zk}}_{\mathit{NIZK}_{\mathsf{DKeyGenShare}}}(t)$ *is the best advantage of any PPT adversary running in time t against the zero-knowledge property of the* $\mathit{NIZK}_{\mathsf{DKeyGenShare}}$ *scheme;*
- $\mathsf{Adv}^{\mathsf{zk}}_{\mathit{NIZK}_{\mathsf{Encrypt}}}(t)$ *is the best advantage of any PPT adversary running in time t against the zero-knowledge property of the* $\mathit{NIZK}_{\mathsf{Encrypt}}$ *scheme;*
- $\mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{LDSUM}}(t, q_K)$ *is the best advantage of any PPT adversary running in time t with q_K encryption queries against the* $\mathtt{IND}$*-security game of the* LDSUM *scheme.*
- $\mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{MCFE}}(t, q_E + 2n)$ *is the best advantage of any PPT adversary running in time t with $q_E + 2n$ encryption queries against the* $\mathtt{IND}$*-security game of the* MCFE *scheme in [CDG+18].*

Proof. We proceed by using a hybrid argument. Let $\mathcal{A}$ be a PPT adversary running in time t with q_E encryption queries. For any game $\mathbf{G}$, we write $\mathsf{Adv}_{\mathbf{G}}$ the advantage of $\mathcal{A}$ in the game $\mathbf{G}$.

Game $\mathbf{G}_0$: this is the sta-$\mathtt{IND}$-security game as given in Definition 9, with the set $\mathcal{CS}$ of corrupted senders known from the beginning.

Game $\mathbf{G}_1$: this game is as $\mathbf{G}_0$, except that for generating proofs:
- $\pi_{\mathsf{DKeyGenShare},i}$ is produced by a simulator $\mathsf{Sim}_{\mathsf{DKeyGenShare}}$ on the input $(\mathsf{LDSUM.pk}, \mathsf{com}_{\mathsf{DMCFE},i}, \mathsf{dk}_i, \ell_{\boldsymbol{y}})$ if i is queried in $\mathsf{QDKeyGen}(i, \boldsymbol{y})$ such that $y_i \in [2^m - 1]$ for all $i \in [n]$.
- $\pi_{\mathsf{Encrypt},i}$ is produced by a simulator $\mathsf{Sim}_{\mathsf{Encrypt}}$ on the input $(\ell, [c_{\ell,i}]_1, \mathsf{com}_{\mathsf{DMCFE},i}, m)$ if i is queried in $\mathsf{QEncrypt}(i, x^0, x^1, \ell)$ for $x^0, x^1 \in [0, 2^m - 1]$.

Given q_E encryption queries and q_K functional key share queries, by the zero-knowledge property of NIZK proofs, we have

$$|\mathsf{Adv}_{\mathbf{G}_0} - \mathsf{Adv}_{\mathbf{G}_1}| \leq q_K \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{DKeyGenShare}}}(t) + q_E \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{Encrypt}}}(t).$$

Game $\mathbf{G}_2$: this game is as $\mathbf{G}_1$, except for functional key share queries $\mathsf{QDKeyGen}$ (i, $\boldsymbol{y}$):

– if i is the last non-corrupted index, the challenger computes

$$\mathsf{dk}_i = (\mathsf{LDSUM.Encrypt}(\mathsf{LDSUM.ek}_i, s_{\mathsf{DMCFE},i,b} + \boxed{\textstyle\sum_{j \in \mathcal{HS}, j \neq i} s_{\mathsf{DMCFE},j,b} y_j}, \ell_{\boldsymbol{y},b}))_{b \in [2]};$$

– for other non-corrupted index i, the challenger computes

$$\mathsf{dk}_i = (\mathsf{LDSUM.Encrypt}(\mathsf{LDSUM.ek}_i, \boxed{0}, \ell_{\boldsymbol{y},b}))_{n \in [2]};$$

Given q_K key share queries, by the (static) `IND`-security of the LDSUM scheme, we have

$$|\mathsf{Adv}_{\mathbf{G}_2} - \mathsf{Adv}_{\mathbf{G}_1}| \leq \mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{LDSUM}}(t, q_K).$$

We now reduce the `IND`-security of the MCFE scheme in [CDG+18] to the game in $\mathbf{G}_2$. We construct an adversary $\mathcal{B}_{\mathsf{MCFE}}$ against the security of the MCFE scheme as in Figure 4. From the reduction,

$$\mathsf{Adv}_{\mathbf{G}_2}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{MCFE}}(t, q_E + 2n).$$

To complete the proof, we have

$$\begin{aligned}\mathsf{Adv}_{\mathbf{G}_0} \leq q_K \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{DKeyGenShare}}}(t) + q_E \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{Encrypt}}}(t) \\ + \mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{LDSUM}}(t, q_K) + \mathsf{Adv}^{\mathsf{sta-ind}}_{\mathsf{MCFE}}(t, q_E + 2n).\end{aligned}$$

C Privacy Improvement with AoNE

Using a AoNE layer on the DMCFE ciphertexts, we can remove the condition 2 in the security model in Definition 9. We recall the pairing-based AoNE construction in [CDSG+20].

– $\mathsf{AoNE.SetUp}(\lambda)$: Generates a pairing group $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, P_1, P_2, e) \stackrel{\$}{\leftarrow} \mathsf{PGGen}(1^\lambda)$, a full domain hash function $\mathcal{H}$ from $\{0,1\}^*$ onto $\mathbb{G}_1$, a symmetric encryption scheme $\mathsf{SKE} = (\mathsf{SEnc}, \mathsf{SDec})$, and outputs $\mathsf{pp} = (\mathcal{PG}, \mathcal{H}, \mathsf{SKE})$. We denote by $[h_x]$ the hash value of $\mathcal{H}$ on any message x, and pp is implicit input to other algorithms.
– $\mathsf{AoNE.KeyGen}()$: Samples $t_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and outputs $(\mathsf{pk}_i, \mathsf{sk}_i) = ([t_i]_2, t_i)$.
– $\mathsf{AoNE.Encrypt}(\mathsf{sk}_i, m)$: Parses $\mathsf{sk}_i = t_i$ and $m = (x_i, \ell)$. Samples $r_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and computes the symmetric key $K_{i,\ell}$ as

$$e\left(\mathcal{H}(\ell), r_i \cdot \left(\sum_{i \in [n]} \mathsf{pk}_i\right)\right) = \left[h_\ell \cdot r_i \cdot \sum_{i \in [n]} t_i\right]_T,$$

and uses it to encrypt x_i as $c_i = \mathsf{SEnc}(K_{i,\ell}, x_i)$. Computes its share $S_{i,\ell} = t_i \cdot \mathcal{H}(\ell) = [t_i \cdot h_\ell]_1$, and outputs the ciphertext $\mathsf{ct}_i = (c_i, [r_i]_2, S_{i,\ell}, \ell)$.
– $\mathsf{AoNE.Decrypt}((\mathsf{ct}_i)_{i \in [n]})$: Parses the ciphertexts as $\mathsf{ct}_i = (c_i, [r_i]_2, S_{i,\ell}, \ell)$ for all $i \in [n]$. For each $i \in [n]$, computes

$$K_{i,\ell} = e\left(\sum_{i \in [n]} S_{i,\ell}, [r_i]_2\right) = \left[h_\ell \cdot r_i \cdot \sum_{i \in [n]} t_i\right]_T$$

and recovers x_i as $x_i = \mathsf{SDec}(K_{i,\ell}, c_i)$.

The ciphertext verification will consist of two steps when a AoNE layer is added on the DMCFE ciphertexts:

– AoNE share validation: each sender sends additionally with his encapsulation a Σ-proof $\pi_{i,\mathsf{AoNE}}$, which proves a DDH relation

$$\mathcal{R}_{\mathsf{AoNE}}(S_{i,\ell}, \mathsf{pk}_i; t) = 1 \leftrightarrow S_{i,\ell} = t \cdot \mathcal{H}(\ell) \wedge \mathsf{pk}_i = [t]_2.$$

When all $\pi_{i,\mathsf{AoNE}}$ for $i \in [n]$ are valid, the receiver decrypts encapsulations by running $\mathsf{Decrypt}((\mathsf{ct}_i)_{i \in [n]})$ and continues. Otherwise, returns 0.

Reduction from $\texttt{IND} - \mathsf{MCFE}$ to $\mathbf{G}_2$:

$\mathcal{B}_{\mathsf{MCFE}}$ calls $\mathcal{A}$ and plays as the challenger in $\mathbf{G}_2$.

$\mathcal{A}$ sends the corruption queries to $\mathcal{B}_{\mathsf{MCFE}}$ before the initialization. $\mathcal{B}_{\mathsf{MCFE}}$ then sends the same corruption queries to the $\texttt{IND}$-security game of MCFE.

After the output $b'_{\mathcal{A}} \leftarrow \mathcal{A}^{\mathsf{QEncrypt}(\cdot,\cdot,\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{QDKeyGen}(\cdot)}$, $\mathcal{B}_{\mathsf{MCFE}}$ outputs $b'_{\mathcal{B}_{\mathsf{MCFE}}} \leftarrow b'_{\mathcal{A}}$.

Initialization:

- SetUp (λ): On receiving the MCFE public parameters $\mathsf{MCFE.pp}$ in $\mathbb{G}_1$, adversary $\mathcal{B}_{\mathsf{MCFE}}$ generates
$$\mathsf{pp} = (\mathcal{PG}, (\mathcal{H}_b)_{b\in[2]}, \mathsf{LDSUM.SetUp}(\lambda, \mathbb{G}_2), \ell_{\mathsf{LDSUM}}, \ell_{\mathsf{DMCFE}}).$$
- KeyGen (): $\mathcal{B}_{\mathsf{MCFE}}$ generates $(\mathsf{LDSUM.sk}_i)_{i\in[n]}$ and $\mathsf{LDSUM.pk}$. For all $i \in [n]$, it generates $\mathsf{com}_{\mathsf{DMCFE},i} \leftarrow (\mathsf{MCFE.QEncrypt}(i, 0, 0, \ell_{\mathsf{DMCFE},b}))_{b\in[2]}$;

Returns pp, $\mathsf{vk}_{\mathsf{DK}} = (\mathsf{LDSUM.pk}, (\mathsf{com}_{\mathsf{DMCFE},i})_{i\in[n]})$, $\mathsf{vk}_{\mathsf{CT}} = (\mathsf{com}_{\mathsf{DMCFE},i})_{i\in[n]}$ and $\mathsf{pk} = \mathsf{LDSUM.pk}$.

$\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$:

$[c_{\ell,i}]_1 \leftarrow \mathsf{MCFE.QEncrypt}(i, x_i^0, x_i^1, \ell)$.

$\mathcal{B}_{\mathsf{MCFE}}$ calls $\mathsf{Sim}_{\mathsf{Encrypt}}(\ell, [c_{\ell,i}]_1, \mathsf{com}_{\mathsf{DMCFE},i}, m)$ to simulate $\pi_{\mathsf{Encrypt},i}$.

Returns $C_{\ell,i} = (\ell, [c_{\ell,i}]_1, \pi_{\mathsf{Encrypt},i})$.

$\mathsf{QCorrupt}(i)$:

$\boldsymbol{s}_{\mathsf{DMCFE},i} \leftarrow \mathsf{MCFE.QCorrupt}(i)$

Returns $\mathsf{sk}_i = (\boldsymbol{s}_{\mathsf{DMCFE},i}, \mathsf{LDSUM.sk}_i)$.

$\mathsf{QDKeyGen}(i, \boldsymbol{y})$:

- If i is the last non-corrupted index, $\mathcal{B}_{\mathsf{MCFE}}$ first sends a key query $\mathsf{MCFE.QDKeyGen}(\boldsymbol{y})$ to obtain $\mathsf{dk}_{\boldsymbol{y}} \in \mathbb{Z}_p^2$. Note that $\mathsf{dk}_{\boldsymbol{y}} = (\sum_{j\in\mathcal{CS}} s_{\mathsf{DMCFE},j,b} y_j + \sum_{j\in\mathcal{HS}} s_{\mathsf{DMCFE},j,b} y_j)_{b\in[2]}$. Then $\mathcal{B}_{\mathsf{MCFE}}$ computes
$$\mathsf{dk}_i = (\mathsf{LDSUM.Encrypt}(\mathsf{LDSUM.ek}_i, \mathsf{dk}_{\boldsymbol{y},b} - \sum_{j\in\mathcal{CS}} s_{\mathsf{DMCFE},j,b} y_j, \ell_{\boldsymbol{y},b}))_{b\in[2]}.$$
- For other non-corrupted index i, the challenger computes
$$\mathsf{dk}_i = (\mathsf{LDSUM.Encrypt}(\mathsf{LDSUM.ek}_i, 0, \ell_{\boldsymbol{y},b}))_{b\in[2}.$$
- If i is a corrupted index, then the challenger computes
$$\mathsf{dk}_i = (\mathsf{LDSUM.Encrypt}(\mathsf{LDSUM.ek}_i, s_{\mathsf{DMCFE},j,b} y_j, \ell_{\boldsymbol{y},b})_{b\in[2]}.$$

$\mathcal{B}_{\mathsf{MCFE}}$ calls $\mathsf{Sim}_{\mathsf{DKeyGenShare}}(\mathsf{LDSUM.pk}, \mathsf{com}_{\mathsf{DMCFE},i}, \mathsf{dk}_i, \ell_{\boldsymbol{y}})$ to simulate $\pi_{\mathsf{DKeyGenShare},i}$.

Returns $\mathsf{dk}_{i,\boldsymbol{y}} = (\mathsf{dk}_i, \ell_{\boldsymbol{y}}, \pi_{\mathsf{DKeyGenShare},i})$.

Fig. 4. Reduction for the proof of Theorem 3. Here we leverage the MCFE encryption of 0 under a specific label ℓ_{DMCFE} to obtain a commitment of private encryption keys in the KeyGen () of the initilization phase.

– DMCFE ciphertext verification: each x_i decrypted from AoNE is parsed as an DMCFE ciphertext under the label ℓ, namely $x_i = [c_{\ell,i}]$. The receiver then verifies $C_{\ell,i} = (\ell, [c_{\ell,i}], \pi_{\mathsf{Encrypt},i})$.

In this AoNE scheme, each share $S_{i,\ell}$ is used as global input to reveal the symmetric key $K_{j,\ell}$ for all $j \in [n]$, while $[r_i]_2$ is used for $K_{i,\ell}$ only. Therefore, the first step is to identify malicious senders who want to distort the symmetric key of other honest senders. The second step is to identify malicious senders who give maliciously generated DMCFE ciphertexts.

In terms of the $\mathtt{IND}$-security, the Σ-proof for $\mathcal{R}_{\mathsf{AoNE}}$ can be simulated by a zero-knowledge simulator. Therefore, the $\mathtt{IND}$-security of the verifiable DMCFE scheme with the two-step verification (as described above) can be reduced to that of the DMCFE scheme with a layer of AoNE encapsulation on the ciphertexts.