Verifiable Multi-Client Functional Encryption for Inner Product

Dinh Duy Nguyen[1], Duong Hieu Phan[1], and David Pointcheval[2]

[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 theory, multi-party computation (MPC) allows for secure computation, but it is often impractical due to intensive interactions between users. 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 design a concrete construction of verifiable MCFE for inner-product computations where the inputs are within a range. Verifiability cannot be easily obtained from classical proof systems only because the encryption key is usually secret in MCFE and the encryptor can maliciously perform the encryption without being detected. So we need to effectively combine different techniques such as commitments and range proofs to achieve the verifiability. Our approach can also be applied to input validation for secure aggregation as a special case.

Keywords: Verifiability, multi-client, functional encryption, inner product.

1 Introduction

1.1 Context

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.

In [CDG+18], Chotard et al. provided a decentralized MCFE for the inner product where all clients only need to run an MPC protocol once during the setup and remove the trusted

authority. With this setup, each client can have complete control over their individual data and the generation of the functional decryption key over their individual encrypted data. As a follow-up to their work, the MPC protocol was removed by a decentralized sum protocol in [CDSG+20], making the decentralized MCFE for inner products completely non-interactive and eliminating the need for pairings in the groups.

The main differences between MCFE and MIFE are that MCFE limits the possible combinations to build a global ciphertext with a common label in each individual ciphertext, but on the other hand, MCFE allows the ciphertexts to be generated by independent clients, and some of them can be corrupted. The latter point makes MCFE more general, while the former is usually more restrictive as one usually limits to one ciphertext per client and per label. Allowing repetitions would make MCFE more general than MIFE. But in this paper, we follow definitions from [CDG+18].

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 MCFE 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 keys, so that once all are valid, the decryption result is guaranteed to not be significantly biased. Beyond the inner product, we define a verifiable MCFE, which provides security for the receiver in the multi-client case setting of Badrinarayanan et al [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 MCFE for Inner Product. Verifiability for MCFE 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-MCFE 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 MCFE 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 ciphertexts and 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. This is technically feasible as households are now equipped with smart metering devices which have the ability to record the energy consumption of a particular measuring point in intervals of fifteen minutes or even less (real-time). 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 multi-client functional encryption to send their data in an encrypted form. However, nothing guarantees that the electricity supplier receive a correct aggregate of the metered energy consumption or at least an approximation of this value. In fact, some customers may encrypt arbitrary value, spoiling the joint-input function by making it undefined or be far away from the correct value. Moreover, the authority, who are not on the same side with the supplier, may also produce malicious parameters and malicious functional decryption keys which do not correspond to the asked functions. Therefore, if we can enforce each client to encrypt a value in some valid range and enforce the authority to produce setup and functional decryption keys honestly, then the noise made by malicious clients can be mitigated when the aggregate value is among a large number of clients.

The fact that this scenario has not been captured in prior work of MCFE is historically reasonable: in single-input FE, there is only one encryptor. When this encryptor wants to spoil 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, the decryptor can trivially detect the invalidity of the input via the invalidity 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 multi-client FE. On the other hand, a decryptor in MCFE, can only learn the joint function evaluated on the joint input, then it seems not trivial to detect invalid individual inputs of the malicious clients out of the valid ones. We stress that using functional encryption schemes for modular inner product over $\mathbb{Z}_p$ [ALS16] where p can be any prime 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, it seems that verifying (or even reducing) the value of each encrypted input is a must to tackle this issue.

The challenge is that as each individual input must be kept private from any adversary, so it is encrypted such that the encryptions between it and any other (possibly invalid) value, must be indistinguishable. Moreover, the plaintext domain is usually defined by the security parameter, then it is not simple to restrict the plaintext domain to a specific set of inputs without compromising the security.

1.2 Contributions

Our contributions for verifiable MCFE can be listed as:

- **Definition**: We introduce the definition of verifiable MCFE, which guarantees that the decryption process, given a vector of ciphertexts and functional decryption keys, always returns the agreed functions evaluated on a vector of inputs lying in a specific range. The verifiability is formalized as a security game against any probabilistic polynomial time (PPT) adversary that can corrupt all senders and the functional decryption key authority.
- **Construction**: We provide a concrete construction of verifiable MCFE for inner product. We deal with subtle security problems in the proof for verifiability to provide a construction that is simple and efficient: the verifiability requires no structured common reference string but random oracle, and is based only on extensively employed zero-knowledge proof of membership such as Schnorr-based proofs and any Pedersen-commitment-based range proofs. When

instantiated with Bulletproofs [BBB+18] and assuming n senders and the range to verify is $[0, 2^m]$, the ciphertext size has an overhead of $\mathcal{O}(\log_2 \lceil m \rceil)$, the overhead in encryption time is $\mathcal{O}(m)$, and the verification time is $\mathcal{O}(m + n \log_2 \lceil m \rceil)$. Our verifiable MCFE scheme can be automatically adapted to become verifiable decentralized MCFE scheme.

– **Privacy Improvement**: We improve the IND-security model of our verifiable MCFE scheme by adding a layer of All or Nothing Encapsulation on the ciphertext, we then provide a two-step verification process for ciphertexts to guarantee the ability of detecting malicious senders.

1.3 Technical Overview

We briefly show the path to our concrete construction of range-verifiable MCFE for inner product. Our construction focuses on the practical efficiency in terms of time complexity for any party and of the ciphertext size. Therefore, our building blocks are restricted to practical primitives only: the MCFE scheme for inner product in [CDG+18], the Schnorr-based protocols, and the range proofs on Pedersen commitment.

The initial observation is that the encryption in the MCFE scheme [CDG+18] is computed in the form of a Pedersen commitment with message x_i and a two-dimensional opening $\boldsymbol{s}_i$, namely $[c_i] = [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i]$ 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.

First Attempt (naive approach). To focus on the ciphertext verification, we first assume that the authority acts honestly in generating functional decryption keys. Each sender is required to send a range proof π, which is a proof of knowledge for the message and the opening of a Pedersen commitment, on a ciphertext $[c_i]$ for the relation:

$$\mathcal{R}_{\mathsf{range}+}([c_i], m; \boldsymbol{s}_i, x_i) = 1 \leftrightarrow [c_i] = [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i] \wedge x_i \in [0, 2^m - 1].$$

A range proof is, however, not enough to guarantee the verifiability in this case. In an MCFE scheme, the encryption key is private to each sender, and it is strictly correlated to the secret key used to generate a functional decryption key. A malicious sender can then encrypt any valid $x_i' \in [0, 2^m - 1]$, but under a false encryption key $\boldsymbol{s}_i' \neq \boldsymbol{s}_i$ and use the witness $(\boldsymbol{s}_i', x_i')$ to generate a valid range proof. One may have $[c_i] = [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i' + [x_i'] = [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i]$ where $x_i \notin [0, 2^m - 1]$, and then the invalid value x_i will be injected into the function during the decryption.

Second Attempt. Our solution is to publish an encryption-key commitment for each sender during the key generation before giving it to the sender. The trick is encrypting a message 0 under an initilization label ℓ_0, i.e. $\mathsf{vk}_i = [\boldsymbol{u}_{\ell_0}^\top] \cdot \boldsymbol{s}_i$. By using this commitment, we can reduce the IND-security of the verifiable MCFE scheme to the MCFE scheme in [CDG+18]. The range proof has to be transformed into an argument of knowledge for the relation

$$\mathcal{R}([c_i], \mathsf{vk}_i, m; \boldsymbol{s}_i, x_i) = 1 \leftrightarrow [c_i] = [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i] \wedge \mathsf{vk}_i = [\boldsymbol{u}_{\ell_0}^\top] \cdot \boldsymbol{s}_i \wedge x_i \in [0, 2^m - 1].$$

To obtain a zero-knowledge argument of knowledge for $\mathcal{R}$, our strategy is to efficiently combine a Schnorr-based proof and a range proof. On the other hand, the functional decryption key for an inner-product function $\boldsymbol{y}$ is $\mathsf{dk}_{\boldsymbol{y}} = \langle \boldsymbol{y}, \boldsymbol{x} \rangle$. Therefore, the decryption key verification is designed based on the additively homomorphic property of vk_i: one will use $(\boldsymbol{y}, (\mathsf{vk}_i)_i)$ to check the equality $\sum_i y_i \cdot \mathsf{vk}_i = [\boldsymbol{u}_{\ell_0}^\top] \cdot \mathsf{dk}_{\boldsymbol{y}}$.

To prove the verifiability in this case, a simulator may have to extract all messages and encryption keys $(x_i, \boldsymbol{s}_i)$ in parallel, since $([c_i], \mathsf{vk}_i)$ is perfectly hiding for $(\boldsymbol{s}_i, x_i)$. However, the extraction from a proof for $\mathcal{R}$ may require rewinding oracle. Therefore, extracting multiple instances may cause an exponential loss in the security reduction, which is shown in [PS96] and [SG98].

Our Solution: We prove by using the soundness only. The encryption key commitment is first transformed into $\mathsf{vk}_i = ([\boldsymbol{u}_{\ell_0}^\top] \cdot \boldsymbol{s}_i, [\boldsymbol{u}_{\ell_0'}^\top] \cdot \boldsymbol{s}_i)$, which is a perfectly binding commitment for $\boldsymbol{s}_i$ as long as $\boldsymbol{u}_{\ell_0}$ and $\boldsymbol{u}_{\ell_0'}$ are linearly independent. The purpose of this two-line commitment is to make the relation

$$\begin{aligned}&\mathcal{R}_{\mathsf{range-key}}([c_i], \mathsf{vk}_i, m; \boldsymbol{s}_i, x_i) = 1\\ &\leftrightarrow [c_i] = [\boldsymbol{u}_\ell^\top] \cdot \boldsymbol{s}_i + [x_i] \wedge \mathsf{vk}_i = ([\boldsymbol{u}_{\ell_0}^\top] \cdot \boldsymbol{s}_i, [\boldsymbol{u}_{\ell_0'}^\top] \cdot \boldsymbol{s}_i) \wedge x_i \in [0, 2^m - 1]\end{aligned}$$

define a non-trivial language $L_{\mathsf{range-key}}$ for $([c_i], \mathsf{vk}_i) \in \mathbb{G}^3$. Any argument of knowledge for $\mathcal{R}_{\mathsf{range-key}}$ is also an argument of membership for $L_{\mathsf{range-key}}$. The verification for functional decryption keys consequently verifies for two equalities: $\sum_i y_i \cdot \mathsf{vk}_i = ([\boldsymbol{u}_{\ell_0}^\top] \cdot \mathsf{dk}_{\boldsymbol{y}}, [\boldsymbol{u}_{\ell_0'}^\top] \cdot \mathsf{dk}_{\boldsymbol{y}})$ and obtains a perfect soundness.

Two-Step Verification for Detecting Malicious Senders. For the IND-security game, we can remove the condition that under the same label, if an encryption query has been asked for an honest client, then all encryption queries under the same label for other honest clients must have been asked. The technique for dealing with incomplete ciphertexts is to add one more layer of AoNE encryption on the verifiable MCFE ciphertext, as proposed in [CDSG+20]. The problem is that to guarantee the ability to detect malicious senders, which is naturally offered by the verifiable inner-product MCFE scheme, one may have to implement an expensive zero-knowledge argument to prove that each encapsulation of $[c_i]$ is generated honestly and $[c_i]$ is a valid ciphertext. To overcome this difficulty, we propose a two-step verification based on the pairing-based AoNE from [CDSG+20]:

- First Step: For each encapsulation, only the share that is used to open other senders encapsulation is verified. This step can be done efficiently by asking each sender to provide a Schnorr-based proof of Diffie-Hellman pairs. This step is to protect the underlying VMCFE ciphertexts $[c_i]$ of honest senders, and any malicious sender who tries to distort these $[c_i]$ will be identified.
- Second Step: The receiver proceeds the ciphertext verification as in the construction of verifiable inner-product MCFE. Under the assumption that the VMCFE ciphertexts of all senders are successfully revealed, any sender who sends invalid $[c_i]$ will be identified. This step maintains the verifiability for the scheme.

1.4 Related Work and Comparisons

Formalization. Our definition of verifiability for MCFE could be seen as a generalized computational version of the verifiability for MIFE in [BGJS16] with some differences. By introducing predicates for messages to be encrypted and a predicate for the function, our definition captures a larger class of verification and additionally validates the content of messages within ciphertexts and the content of functions within functional decryption keys. A detailed comparison will be given in Section 7 when we introduce the formalization for our notion.

Comparing with [BGL+22], if we restrict the functionality to be a sum of encrypted inputs, then we can obtain an analog input validation for secure aggregation. In their ACORN-detect protocol, each encrypted input is guaranteed to be in a small range. On the other hand, our VMCFE does not cover the case of dropout senders (who cannot send their ciphertexts) as in [BGL+22]. The reason is that we want to maintain a basic property of an encryption scheme: no interaction between senders and a receiver during the decryption process should be required. If the receiver can decrypt with incomplete ciphertexts without any agreement (which requires interaction) from all senders, then this may simultaneously violate the security of an MCFE scheme: only when ciphertexts under the same label of all senders are sent, then the receiver is able to decrypt.

Malicious sender detection is an interesting feature but is not obvious to obtain from the verifiability. An example is the ACORN-detect in [BGL+22]: this protocol allows validating the aggregated key, which is combined from all key shares of senders, by using an interactive Schnorr-based protocol between the server and each sender. However, it is expensive to prove and verify that each key share is honestly generated, so it is difficult to identify the malicious sender in the case the aggregated key is invalid. The authors in [BGL+22] propose another protocol, called ACORN-robust, that can detect malicious senders and remove their inputs, but requires a bounded number of malicious senders and more interactions. In our verifiable inner-product MCFE, there is usually an authority who generates functional decryption keys. If a key is invalid, then he must be the malicious one. Our VMCFE scheme when extended to decentralized VMCFE scheme also allows each sender to control the generation of his own functional decryption key share as in the ACORN-detect protocol, but an important difference is that our scheme is totally non-interactive, even in the verification process for the functional decryption key. An efficient approach to verify each key share is yet still an open problem.

In short, the main advantages that our verifiable (decentralized) MCFE scheme has over the ACORN-detect protocol in [BGL+22] are the non-interactivity and the possibility of decryption for inner products, which is a more general class of functions than the sum.

Efficiency. As [BGJS16] does not yield a construction for the inner product, we compare some significant costs that are needed for the verifiability of our schemes with those for the input validation of ACORN-detect protocol in [BGL+22].

For the comparison, we assume a security parameter λ and a range $[0, 2^m - 1]$ to verify the inputs. We do not count scalar operations and scalar elements, since the cost and the size are small compared to the cost of group exponentiation and the size of group element respectively. For the ciphertexts, we will focus only on the costs for proofs of smallness on encrypted inputs, which has the costs that dominate the costs of other tasks for verifiability. The ACORN-detect allows aggregating vectors of size ℓ, then the costs are optimized for ℓ.

Round Complexity: The ACORN-detect protocol requires a 5-round interactive variant of Schnorr's protocol between the server and each client during the verification of the aggregated key. In our decentralized verifiable inner-product MCFE, the functional decryption key can be verified without any interaction.

Computational Cost: Although aiming at different primitives (summation is a special case of inner product function), the computational costs are essentially the same in both protocols. Both require each of its senders to produce a proof of smallness on his encrypted input and can be instantiated with Bulletproof [BBB+18]. The cost of generating a proof of smallness in ACORN-detect is $6k+8(\log_2 \lceil k \rceil + 1)$ group exponentiations and a multi-exponentiation of size k for $k = 8\ell + \lambda$. Meanwhile, a sender in our VMCFE scheme, which supports computing inner products with encrypted scalar inputs, needs about $12m + 10$ group exponentiations for the proof of smallness.

For the batch verification of proofs of smallness, the cost for ACORN-detect is a multi-exponentiation of size $2k + 2 + n \log_2 \lceil 2k + 4 \rceil$ and two multi-exponentiations of size 8ℓ, while the cost for VMCFE scheme is a multi-exponentiation of size $2m + 3 + n(2 \log_2 \lceil m \rceil + 5)$.

The verification for the aggregated key in ACORN-detect is $n \cdot (5 + \ell)$ group exponentiations, while the verification for the functional decryption key in VMCFE scheme is $2n + 4$ group exponentiations.

Communication Cost: Both protocols enjoy the efficiency from the Bulletproof: for ACORN-detect, the size of proof of smallness is $2 \log_2 \lceil k \rceil + 4$ group elements for $k = 8\ell + \lambda$. For inner-product VMCFE scheme, the proof size is $2 \log_2 \lceil m \rceil + 7$ group elements in total.

2 Preliminaries

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

2.1 Computational 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 $\mathcal{G} \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, no PPT adversary can distinguish between the two following distributions with non-negligible advantage:*

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

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

2.2 Commitments

Definition 2 (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 $(\mathsf{SetUp}, \mathsf{Commit}, \mathsf{Verify})$:*

- $\mathsf{SetUp}(\lambda)$*: Takes as input a security parameter λ , outputs public parameters pp (which are implicit to other algorithms);*
- $\mathsf{Commit}(m)$*: 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.*
- $\mathsf{Verify}(c, r, m)$*: 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 3 (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*

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

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 4 (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*

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

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

2.3 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 6 (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* $\mathcal{R}(u; w) = 1$,
$$\Pr\begin{bmatrix}\sigma \leftarrow \mathsf{SetUp}(\lambda), \\ \pi \leftarrow \mathsf{Prove}(u, w)\end{bmatrix} : \mathsf{Verify}(u, \pi) = 1 = 1.$$
2. *Computational Soundness. For all PPT adversaries* $\mathcal{A}$*, there is a negligible function* $\mu(\lambda)$ *such that*
$$\Pr\begin{bmatrix}\sigma \leftarrow \mathsf{SetUp}(\lambda), \\ (u, \pi) \leftarrow \mathcal{A}(\sigma)\end{bmatrix} : \mathsf{Verify}(u, \pi) = 1 \wedge u \notin L \leq \mu(\lambda).$$
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*
$$\left|\Pr\begin{bmatrix}\sigma \leftarrow \mathsf{SetUp}(\lambda), \\ (u, w, \mathsf{st}) \leftarrow \mathcal{A}_1(\sigma) : \\ \pi \leftarrow \mathsf{Prove}(u, w)\end{bmatrix}\begin{matrix}\mathcal{A}_2(\sigma, \pi, \mathsf{st}) = 1 \\ \wedge \mathcal{R}(u; w) = 1\end{matrix}\right.$$
$$\left.-\Pr\begin{bmatrix}(\sigma, \tau) \leftarrow \mathcal{S}_1(\lambda), \\ (u, w, \mathsf{st}) \leftarrow \mathcal{A}_1(\sigma) : \\ \pi \leftarrow \mathcal{S}_2(\sigma, u, \tau)\end{bmatrix}\begin{matrix}\mathcal{A}_2(\sigma, \pi, \mathsf{st}) = 1 \\ \wedge \mathcal{R}(u; w) = 1\end{matrix}\right| \leq \mu(\lambda).$$

where τ *is a trapdoor for* σ *and* st *is an internal state.*

3 Definition and Security Models

3.1 Definition

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 7 (Verifiable Multi-Client Functional Encryption). *A verifiable multi-client functional encryption on $\mathcal{M}$ over $(\mathcal{F}, \mathcal{P}^{\mathtt{m}}, \mathcal{P}^{\mathtt{f}})$, and a set of n senders consists of seven 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;*
- $\mathsf{KeyGen}()$*: Outputs the n encryption keys* ek_i*, the secret key* sk*, the public key* pk *and the n public verification keys* vk_i*;*
- $\mathsf{Encrypt}(\mathsf{ek}_i, \mathsf{vk}_i, x_i, \ell, \mathsf{P}_i^{\mathtt{m}})$*: Takes as input an encryption key* ek_i*, a verification key* vk_i*, a value x_i to encrypt, a label ℓ and a predicate $\mathsf{P}_i^{\mathtt{m}} \in \mathcal{P}^{\mathtt{m}}$. Outputs the ciphertext $C_{\ell,i}$;*
- $\mathsf{DKeyGen}(\mathsf{sk}, \mathsf{pk}, f, \mathsf{P}^{\mathtt{f}})$*: Takes as input the secret key* sk*, the public key* pk*, a function $f \in \mathcal{F}$, and a predicate $\mathsf{P}^{\mathtt{f}} \in \mathcal{P}^{\mathtt{f}}$. Outputs a functional decryption key* dk_f*;*
- $\mathsf{VerifyDK}(\mathsf{pk}, \mathsf{dk}_f, \mathsf{P}^{\mathtt{f}})$*: Takes as input the public key* pk*, a functional decryption key* dk_f*, and a predicate $\mathsf{P}^{\mathtt{f}} \in \mathcal{P}^{\mathtt{f}}$. Outputs 1 or 0;*
- $\mathsf{VerifyCT}(\mathsf{vk}_i, C_{\ell,i}, \mathsf{P}_i^{\mathtt{m}})$*: Takes as input a verification key* vk_i*, a ciphertext $C_{\ell,i}$, and a predicate $\mathsf{P}_i^{\mathtt{m}} \in \mathcal{P}^{\mathtt{m}}$. Outputs 1 or 0;*
- $\mathsf{Decrypt}(\mathsf{dk}_f, \boldsymbol{C}_\ell)$*: Takes as input a functional decryption key* dk_f*, an n-vector ciphertext $\boldsymbol{C}_\ell := (C_{\ell,i})_{i=1}^n$. Outputs $f(\boldsymbol{x})$ or $\perp$.*

Correctness. Given any set of message predicates $(\mathsf{P}_i^{\mathtt{m}})_{i=1}^n \in (\mathcal{P}^{\mathtt{m}})^n$ 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 $\mathsf{P}_i^{\mathtt{m}}(x_i) = 1$ for all $i \in [n]$, and

$$\begin{cases} \mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda) \\ ((\mathsf{ek}_i)_{i=1}^n, \mathsf{sk}, \mathsf{pk}, (\mathsf{vk}_i)_{i=1}^n) \leftarrow \mathsf{KeyGen}() \\ C_{\ell,i} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, \mathsf{vk}_i, x_i, \ell, \mathsf{P}_i^{\mathtt{m}}) \quad \forall i \in [n] \\ \mathsf{dk}_f \leftarrow \mathsf{DKeyGen}(\mathsf{sk}, \mathsf{pk}, f, \mathsf{P}^{\mathtt{f}}) \end{cases}$$

then

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

with probability 1.

Verifiability. For all PPT adversaries $\mathcal{A}$, the advantage in the following game is negligible in λ:

1. *Challenger initializes by running* $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$.
2. *On input* pp*, $\mathcal{A}$ outputs the following:*
   - *Message predicates $(\mathsf{P}_i^{\mathtt{m}})_{i=1}^n$, and a function predicate $\mathsf{P}^{\mathtt{f}}$.*
   - *A public key* pk *and n public verification keys* vk_i*.*
   - *A label ℓ and n ciphertexts under the same label $\boldsymbol{C}_\ell := (C_{\ell,i})_{i=1}^n$.*
   - *A polynomially number of functional decryption keys $\mathsf{dk}_{f_j} = (\mathsf{dk}_j, f_j)$.*
3. *Challenger verifies the following conditions:*
   - *$\mathsf{P}_i^{\mathtt{m}} \in \mathcal{P}^{\mathtt{m}}$ and $\mathsf{P}^{\mathtt{f}} \in \mathcal{P}^{\mathtt{f}}$.*

– $\mathsf{VerifyCT}(\mathsf{vk}_i, C_{\ell,i}, \mathsf{P}^{\mathtt{m}}_i) = 1$ *for all* $i \in [n]$.
– $\mathsf{VerifyDK}(\mathsf{pk}, \mathsf{dk}_{f_j}, \mathsf{P}^{\mathtt{f}}) = 1$ *for all* j.

If any condition is not satisfied, then $\mathcal{A}$ *fails the game.*

4. $\mathcal{A}$ *wins the game if one of the following conditions is satisfied:*
   – *There exists* j *such that* $\mathsf{P}^{\mathtt{f}}(f_j) \neq 1$.
   – *There does not exist a tuple of messages* $(x_i)_{i=1}^n$ *such that*

$$\mathsf{P}^{\mathtt{m}}_i(x_i) = 1$$

   for all $i \in [n]$ *and*

$$\mathsf{Decrypt}(\mathsf{dk}_{f_j}, \boldsymbol{C}_\ell) = f_j(x_1, ..., x_n)$$

   for all j.

In the above definition of verifiability, each functional decryption key dk_{f_j} is assumed to contain the description of its corresponding function f_j, and then a receiver can easily detect if $\mathsf{P}^{\mathtt{f}}(f_j) \neq 1$. For the first winning condition, the verifiability guarantees that an adversary has a negligible chance to produce an accepting dk_{f_j} for an invalid f_j. The second winning condition is determined statistically: the verifiability guarantees that an adversary has a negligible chance to produce a maliciously generated n-vector ciphertext and maliciously generated functional keys dk_{f_j}, 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.

Our definition of verifiability for MCFE, in particular the second 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 key.

Formally, 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.
– **Message and function predicates.** 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 $f \in \mathcal{F}$. Adversary must output dk_{f_j} for $f_j \in \mathcal{F}$, i.e. $\mathsf{P}^{\mathtt{f}}(f_j) \neq 1$. The first winning condition in Step 4 is then never allowed, so it is omitted.
– **Adversary assumption.** The verifiability game is defined for any adversary that has unlimited computing power, is allowed to choose pp, and the advantage of such adversary in the game is 0 (verifiability with no trusted party and perfect soundness).

In our definition, the adversary can choose any message predicates and function predicate by itself, rather than fixing the condition that the encrypted message must be in the message space and function in functional decryption key must be in the function space. 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 pp that only consists of computational assumptions or random oracle.

If we restrict the functionality to be a summation of encrypted inputs, then we can obtain an analog input validation for secure aggregation as in [BGL+22]. In their ACORN-detect protocol, each encrypted input is guaranteed to be in a small range. On the other hand, our VMCFE does not cover the case of dropout senders (who cannot send their ciphertexts) as in [BGL+22]. The

reason is that we want to maintain a basic property of an encryption scheme: no interaction between senders and a receiver during the decryption process should be required. If the receiver can decrypt with incomplete ciphertexts without any agreement (which requires interaction) from all senders, then this may simultaneously violate the security of an MCFE scheme: only when ciphertexts under the same label of all senders are sent, then the receiver is able to decrypt.

3.2 Indistinguishability Security

The security of verifiable MCFE is derived from the indistinguishability security notion of MCFE [CDG+18] as follows.

Definition 8 (IND-Security Game for Verifiable MCFE). *Let us consider a Verifiable* MCFE *scheme over a set of n senders, a function predicate $\mathsf{P}^{\mathtt{f}} \in \mathcal{P}^{\mathtt{f}}$, and a set of message predicates $(\mathsf{P}_i^{\mathtt{m}})_{i=1}^n \in (\mathcal{P}^{\mathtt{m}})^n$. 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 $((\mathsf{ek}_i)_i, \mathsf{sk}, \mathsf{pk}, (\mathsf{vk}_i)_i)$ and chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It sends $(\mathsf{pp}, \mathsf{pk}, (\mathsf{vk}_i)_i)$ 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. If $\mathsf{P}_i^{\mathtt{m}}(x_i^0) = \mathsf{P}_i^{\mathtt{m}}(x_i^1) = 1$, then $\mathcal{A}$ receives the ciphertext $C_{\ell,i}$ generated by $\mathsf{Encrypt}(\mathsf{ek}_i, \mathsf{vk}_i, x_i^b, \ell, \mathsf{P}_i^{\mathtt{m}})$. 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 $\mathsf{QDKeyGen}(f)$: $\mathcal{A}$ has unlimited and adaptive access to the $\mathsf{DKeyGen}(\mathsf{sk}, \mathsf{pk}, f, \mathsf{P}^{\mathtt{f}})$ algorithm for any input function f of its choice. If $\mathsf{P}^{\mathtt{f}}(f) = 1$, it is given back the functional decryption key dk_f. 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 encryption key 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 MCFE *is IND-secure with respect to $\mathsf{P}^{\mathtt{f}}$ and $(\mathsf{P}_i^{\mathtt{m}})_{i=1}^n$ if for any adversary $\mathcal{A}$,*

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

is negligible.

4 Verification Schemes for Inner-Product MCFE

4.1 Consistency Between Encryption Keys and Secret Key

Our first effort to mitigate the effect of malicious inputs on the inner-product evaluation is to specify a data range, which is relatively small compared to the message space, for each individual input. To verify, each sender is supposed to commit its private input and the encryption key it uses in the encryption. Each sender is then required to provide a range proof for its encrypted input, and to send the proof along with its MCFE ciphertext and the commitment. The problem is that a sender does not need to encrypt an arbitrarily large value to bias the inner product: instead, it can encrypt a value in the data range, but under a false encryption key.

Formally, given a data range $[0, 2^m - 1]$ and a private encryption key ek_i from an honest setup, the adversary chooses a valid value $x'_i \in [0, 2^m - 1]$ and use a false encryption key $\mathsf{ek}'_i \neq \mathsf{ek}_i$ to encrypt x'_i as $\mathsf{Encrypt}(\mathsf{ek}'_i, x'_i, \ell)$ in an MCFE scheme. The adversary can easily provide a valid range proof for this ciphertext with a witness containing (ek'_i, x'_i). Since the false encryption key ek'_i may not be consistent with the secret key sk, then when $\mathsf{Encrypt}(\mathsf{ek}'_i, x'_i, \ell)$ is input in the decryption algorithm with other ciphertexts of honest senders and an honest functional decryption key dk_f, the receiver may receive an unexpected value y' rather than $f(\cdot, x'_i, \cdot)$ where $\cdot$ represents other honest senders' inputs.

Input malleability from key inconsistency. For a variety of inner-product MCFE constructions as in [CDG+18, LT19, ABM+20], the secret key sk usually consists of all encryption keys $\mathsf{ek}_i = s_i$, and a functional decryption key for an inner product represented by $\boldsymbol{y}$ is an inner product between $\boldsymbol{y}$ and $\boldsymbol{s} = (s_i)_i$. On the other hand, a ciphertext that encrypts a valid value but under a false encryption key as the above $\mathsf{Encrypt}(\mathsf{ek}'_i, x'_i, \ell)$ may be equal to ciphertext that encrypts an invalid value but under the correct encryption key, for example $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$ where $x'_i \notin [0, 2^n - 1]$. The smaller the valid input range is, the bigger the probability that this case happens. Therefore, it is necessary to publish a commitment of each encryption key and a commitment of the secret key during the setup process to verify for the consistency later.

4.2 Construction for VerifyCT

To construct a verifiable inner-product MCFE scheme later, we focus on the the MCFE scheme in [CDG+18]. To recall, the form of ciphertext 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 $\boldsymbol{s}_i = \mathsf{ek}_i$ 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.

A key observation is that as long as the value of $\boldsymbol{u}_\ell$ remains unknown, 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. A range proof for an interval $[l, r]$ on this form of ciphertext is formalized by the relation $\mathcal{R}_{\mathsf{range}+}$:

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

On the other hand, there is a number of efficient range proof schemes for the committed value in the Pedersen commitment with one scalar opening:

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

Namely, for logarithmic proof size, one has Bulletproof for range in [BBB+18] and range proofs for discrete logarithm setting in [CKLR21] and in [CGKR22]. For sublinear proof size and sublinear verification time, one has Flashproof for range in [WC22]. All these range proof schemes are zero-knowledge arguments of knowledge and require no-trusted setup. These range proof schemes can be transformed into non-interactive ones with Fiat-Shamir transformation.

Therefore, our work does not focus on constructing a more efficient range proof scheme, but to obtain any zero-knowledge argument of knowledge for the relation $\mathcal{R}_{\mathsf{range}+}$ from one for the relation $\mathcal{R}_{\mathsf{range}}$, all the operations on the Pedersen opening is remained and applied to each entry of the 2-dimensional vector opening. The proofs of completeness, extraction and zero-knowledge property are adapted in the same way.

To avoid encryption under a false encryption key, our verifiable inner-product MCFE scheme will require publishing the commitment of each private encryption key during the setup process as $\mathsf{com}_{\mathsf{ek}_i} = ([\boldsymbol{v}^\top] \cdot \mathsf{ek}_i, [\boldsymbol{v'}^\top] \cdot \mathsf{ek}_i) \in \mathbb{G}^2$, where the values $\boldsymbol{v}, \boldsymbol{v'}$ are generated by a random oracle and are unknown to all parties. This commitment is perfectly binding under the assumption that $\boldsymbol{v}$ and $\boldsymbol{v'}$ are linearly independent. In that case, any tuple $([c], \mathsf{com}_{\mathsf{ek}}) \in \mathbb{G}^3$ is a commitment to a pair of message and encryption key $(x, \mathsf{ek}) \in \mathbb{Z}_p^3$.

Each sender is required to provide a proof for the relation $\mathcal{R}_{\mathsf{range-key}}$:

$$\mathcal{R}_{\mathsf{range-key}}([c], \mathsf{com}_{\mathsf{ek}}, l, r; \boldsymbol{s}, x) = 1 \leftrightarrow \begin{cases} [c] = [\boldsymbol{u}^\top] \cdot \boldsymbol{s} + [x] \\ \wedge\, x \in [l, r] \\ \wedge\, \mathsf{com}_{\mathsf{ek}} = ([\boldsymbol{v}^\top] \cdot \boldsymbol{s}, [\boldsymbol{v'}^\top] \cdot \boldsymbol{s}) \end{cases}$$

Since any $([c], \mathsf{com}_{\mathsf{ek}}) \in \mathbb{G}^3$ commits to a unique (x, ek), the above relation defines a non-trivial language $L_{\mathsf{range-key}} \subsetneq \mathbb{G}^3$ for $([c], \mathsf{com}_{\mathsf{ek}})$. Then any argument for $\mathcal{R}_{\mathsf{range-key}}$ is an argument of membership for $L_{\mathsf{range-key}}$.

A Schnorr-based protocol (see Appendix B), denoted by $\mathsf{NIZK}_{\mathsf{key}}$, can be used to prove the relation $\mathcal{R}_{\mathsf{key}}$:

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

Our idea of constructing a zero-knowledge argument scheme for $\mathcal{R}_{\mathsf{range-key}}$ is composing $\mathsf{NIZK}_{\mathsf{key}}$ and $\mathsf{NIZK}_{\mathsf{range}+}$.

Theorem 9. *On public parameters $([\boldsymbol{u}], [\boldsymbol{v}], [\boldsymbol{v'}])$, 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 $\mathsf{NIZK}_{\mathsf{range-key}}$, is a zero-knowledge argument for the language $L_{\mathsf{range-key}}$, defined by the relation $\mathcal{R}_{\mathsf{range-key}}$, on the statement $([c], \mathsf{com}_{\mathsf{ek}}, l, r)$.*

Proof. We note that a transcript of $\mathsf{NIZK}_{\mathsf{range-key}}$ 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 $K^{\mathcal{P}}_{\mathsf{range-key}}$ 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 invokes the extractor $K^{\mathcal{P}}_{\mathsf{range}+}$ on input $([c], l, r)$ and the extractor $K^{\mathcal{P}}_{\mathrm{Schnorr}}$ 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 $\mathcal{R}_{\mathsf{range-key}}$. This implies that $([c], \mathsf{com}_{\mathsf{ek}}, l, r) \in L_{\mathsf{range-key}}$. On the other hand, since $K^{\mathcal{P}}_{\mathsf{range}+}$ and $K^{\mathcal{P}}_{\mathrm{Schnorr}}$ are PPT algorithms, then $K^{\mathcal{P}}_{\mathsf{range-key}}$ runs in polynomial time. The existence of a knowledge extractor $K^{\mathcal{P}}_{\mathsf{range-key}}$ implies the soundness of the protocol.
- *Zero-Knowledge*: Since $\mathsf{NIZK}_{\mathsf{range}+}$ and $\mathsf{NIZK}_{\mathsf{key}}$ are both zero-knowledge, then the simulator $\mathcal{S}_{\mathsf{range-key}}$ can output the concatenation $(\mathcal{S}_{\mathsf{range}+}([c], l, r), \mathcal{S}_{\mathrm{Schnorr}}([c], \mathsf{com}_{\mathsf{ek}}))$ as transcript. The simulated transcript is then indistinguishable from the transcript of an honest execution. The simulator $\mathcal{S}_{\mathsf{range-key}}$ also runs in polynomial time.

By using this approach, our ciphertext verification enjoys the communication and computational efficiency offered by the range proof schemes, since they dominate those that are constant of the Schnorr-based $\mathsf{NIZK}_{\mathsf{key}}$.

An efficient batch verification for ciphertexts is important in the context of verifiable MCFE, where a receiver has to spend time verifying ciphertexts from all senders. As far as we are aware of, Bulletproof for range [BBB+18] provides the most efficient batch verification that can cost only a group multi-exponentiation of size $2m + 2 + n \cdot (2 \log m + 5)$ along with $O(n \cdot m)$ scalar operations for n independent proofs. [3]

4.3 Construction for VerifyDK

In the inner-product MCFE scheme of Chotard et al. [CDG+18], the secret key consists of all private encryption keys. For the verifiable inner-product MCFE, we set the commitment of the secret key as the set of all encryption key commitments, i.e. $\mathsf{com}_{\mathsf{sk}} = (\mathsf{com}_{\mathsf{ek}_i})_i$ where $\mathsf{com}_{\mathsf{ek}_i} = ([\boldsymbol{v}^\top] \cdot \mathsf{ek}_i, [\boldsymbol{v'}^\top] \cdot \mathsf{ek}_i)$. On the other hand, the functional decryption key for an inner product presented by a vector $\boldsymbol{y}$ of the underlying MCFE scheme is $\mathsf{dk}_{\boldsymbol{y}} = (\mathsf{dk} = \sum_{i=1}^{n} y_i \cdot \mathsf{ek}_i \in \mathbb{Z}_p^2, \boldsymbol{y})$. Since $\boldsymbol{y}$ is described in the key $\mathsf{dk}_{\boldsymbol{y}}$, a receiver can verify if $\boldsymbol{y}$ satisfies the predicate for function in clear. The important point is to verify if dk is generated honestly with respect to $\boldsymbol{y}$.

The relation $\mathcal{R}_{\mathsf{dk}}$ that VerifyDK verifies is formalized as:

$$\mathcal{R}_{\mathsf{dk}}((\mathsf{com}_{\mathsf{ek}_i})_i, \mathsf{dk}_{\boldsymbol{y}}; (\boldsymbol{s}_i)_i) = 1 \leftrightarrow \begin{cases} \mathsf{com}_{\mathsf{ek}_i} = ([\boldsymbol{v}^\top] \cdot \boldsymbol{s}_i, [\boldsymbol{v'}^\top] \cdot \boldsymbol{s}_i) \quad \forall i \\ \wedge\ \mathsf{dk} = \sum_{i=1}^{m} y_i \cdot \boldsymbol{s}_i \end{cases}$$

As pointed out in the previous section, any $\mathsf{com}_{\mathsf{ek}_i}$ commits to a unique $\boldsymbol{s}_i$. Then the relation $\mathcal{R}_{\mathsf{dk}}$ defines a non-trivial language L_{dk} for $((\mathsf{com}_{\mathsf{ek}_i})_i, \mathsf{dk}_{\boldsymbol{y}})$. To verify $\mathsf{dk}_{\boldsymbol{y}}$, which is parsed as $(\mathsf{dk}, \boldsymbol{y})$, a receiver verifies the following equality in VerifyDK:

$$\sum_{i=1}^{n} y_i \cdot \mathsf{com}_{\mathsf{ek}_i} = ([\boldsymbol{v}^\top] \cdot \mathsf{dk}_{\boldsymbol{y}}, [\boldsymbol{v'}^\top] \cdot \mathsf{dk}_{\boldsymbol{y}})$$

If the equality does not hold, then $\mathsf{dk}_{\boldsymbol{y}}$ is considered maliciously generated. Note the left hand side of the equality is $\big([\boldsymbol{v}^\top] \cdot (\sum_{i=1}^{m} y_i \cdot \boldsymbol{s}_i), [\boldsymbol{v'}^\top] \cdot (\sum_{i=1}^{m} y_i \cdot \boldsymbol{s}_i)\big)$. The equality implies $\mathsf{dk}_{\boldsymbol{y}} = \sum_{i=1}^{m} y_i \cdot \boldsymbol{s}_i$. This verification has perfect soundness under the assumption that $\mathsf{com}_{\mathsf{ek}_i}$ is perfectly binding.

One may wonder why we need a two-line commitment for $\mathsf{ek}_i = \boldsymbol{s}_i$, i.e. $\mathsf{com}_{\mathsf{ek}_i} = ([\boldsymbol{v}^\top] \cdot \boldsymbol{s}_i, [\boldsymbol{v'}^\top] \cdot \boldsymbol{s}_i)$, instead of one-line one. The reason is that we want to avoid a subtle security problem when proving the verifiability of our scheme: the one-line commitment is perfectly hiding, then $([c], \mathsf{com}_{\mathsf{ek}_i})$ is perfectly hiding for $(x_i, \boldsymbol{s}_i)$. Then we may need a simulation-based proof where the simulator needs to extract $(x_i, \boldsymbol{s}_i)$ from each sender's accepting proof for $\mathcal{R}_{\mathsf{range-key}}$. However, any knowledge extractor in $\mathsf{NIZK}_{\mathsf{range-key}}$ may need rewinding oracle. Therefore, extracting all n senders' witnesses $(x_i, \boldsymbol{s}_i)$ could induce an exponential number of time in n for rewinding as in [PS96] and [SG98]. With the two-line commitment, we will later provide a mathematical argument that an adversary wins the verifiability game only if the soundness of $\mathsf{NIZK}_{\mathsf{range-key}}$ on at least one statement is broken.

No additional proofs need to be generated and sent along with the MCFE functional decryption key. The verification time of functional decryption key is $2n + 4$ group exponentiations.

5 Range-Verifiable MCFE for Inner Product

5.1 Description

Based on the MCFE scheme in [CDG+18], we construct a verifiable inner-product MCFE scheme over the condition of input range. In this scheme, the receiver is guaranteed that each encrypted

[3] There is no batch verification shown in [CKLR21] and [CGKR22].

input lies within a polynomially bounded data range, and the functional decryption key is generated honestly with respect to a publicly known inner-product function presented by a vector, where each entry is also polynomially bounded.

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. The scheme is constructed with the non-interactive zero-knowledge argument $\mathsf{NIZK}_{\mathsf{range-key}} = (\mathsf{NIZK.SetUp}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ for the language $L_{\mathsf{range-key}}$, as specified in Section 4.2.

- $\mathsf{SetUp}(\lambda)$: Takes as input the security parameter λ. It generates prime-order group $\mathcal{G} := (\mathbb{G}, p, P) \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, and $\mathcal{H}$ a full-domain hash function onto $\mathbb{G}^2$, initialization labels (ℓ_0, ℓ'_0), and $\mathsf{pp}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.SetUp}(\lambda)$. The public parameters pp consist of $(\mathcal{G}, \mathcal{H}, \ell_0, \ell'_0, \mathsf{pp}_{\mathsf{NIZK}})$ and are implicit arguments to all other algorithms.
- $\mathsf{KeyGen}()$:
  1. Generates $[\boldsymbol{v}] = \mathcal{H}(\ell_0)$ and $[\boldsymbol{v'}] = \mathcal{H}(\ell'_0)$.
  2. Generates $\boldsymbol{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$ and commits each of them as $([k_i], [k'_i]) = ([\boldsymbol{v}^\top] \cdot \boldsymbol{s}_i, [\boldsymbol{v'}^\top] \cdot \boldsymbol{s}_i)$ for all $i \in [n]$.
  3. The encryption keys are $\mathsf{ek}_i = \boldsymbol{s}_i$ for all $i \in [n]$, and the secret key is $\mathsf{sk} = (\boldsymbol{s}_i)_i$, the public key is $\mathsf{pk} = ([k_i], [k'_i])_i$, and the verification keys are $\mathsf{vk}_i = ([k_i], [k'_i])$ for all $i \in [n]$.
- $\mathsf{Encrypt}(\mathsf{vk}_i, \mathsf{ek}_i, x_i, \ell, m)$: Takes as input the verification key $\mathsf{vk}_i = ([k_i], [k'_i])$, the encryption key $\mathsf{ek}_i = \boldsymbol{s}_i$, the value x_i to encrypt, the label ℓ, and a binary upper bound m.
  1. It computes $[\boldsymbol{u}_\ell] = \mathcal{H}(\ell)$, and computes $[c_{\ell,i}] = [\boldsymbol{u}_\ell^\top \boldsymbol{s}_i + x_i]$.
  2. It computes a proof $\pi^i_{\mathsf{range-key}} \leftarrow \mathsf{NIZK.Prove}(([c_{\ell,i}], \mathsf{vk}_i, m), (x_i, \boldsymbol{s}_i))$.
  3. The ciphertext is $C_{\ell,i} := (\ell, [c_{\ell,i}], \pi^i_{\mathsf{range-key}})$.
- $\mathsf{DKeyGen}(\mathsf{sk}, \boldsymbol{y})$: Takes as input $\mathsf{sk} = (\boldsymbol{s}_i)_i$ and an inner-product function defined by $\boldsymbol{y}$ as $f_{\boldsymbol{y}}(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$, and outputs the functional decryption key $\mathsf{dk}_{\boldsymbol{y}} = (\mathsf{dk} = \sum_{i=1}^n \boldsymbol{s}_i \cdot y_i, \boldsymbol{y})$.
- $\mathsf{VerifyDK}(\mathsf{pk}, \mathsf{dk}_{\boldsymbol{y}}, m)$: Takes as input the public key $\mathsf{pk} = ([k_i], [k'_i])_i$ and a functional decryption key $\mathsf{dk}_{\boldsymbol{y}} = (\mathsf{dk}, \boldsymbol{y})$.
  1. It verifies that for all $i \in [n]$, each entry $y_i \in [0, 2^m - 1]$.
  2. It generates $[\boldsymbol{v}] = \mathcal{H}(\ell_0)$, $[\boldsymbol{v'}] = \mathcal{H}(\ell'_0)$.
  3. It verifies dk satisfies both

$$\sum_{i=1}^n y_i \cdot [k_i] = [\boldsymbol{v}^\top] \cdot \mathsf{dk} \qquad\qquad \sum_{i=1}^n y_i \cdot [k'_i] = [\boldsymbol{v'}^\top] \cdot \mathsf{dk}.$$

  4. If $\boldsymbol{y}$ and dk are valid, then it outputs 1. Otherwise, it outputs 0.
- $\mathsf{VerifyCT}(\mathsf{vk}_i, C_{\ell,i}, m)$: Takes as input the verification key $\mathsf{vk}_i = ([k_i], [k'_i])$, a ciphertext $C_{\ell,i} = (\ell, [c_{\ell,i}], \pi^i_{\mathsf{range-key}})$, and a binary upper bound m.
  1. It computes $[\boldsymbol{u}_\ell] = \mathcal{H}(\ell)$ and $[\boldsymbol{v}] = \mathcal{H}(\ell_0)$, $[\boldsymbol{v'}] = \mathcal{H}(\ell'_0)$.
  2. It computes $b \leftarrow (\mathsf{NIZK.Verify}(([c_{\ell,i}], \mathsf{vk}_i, m), \pi^i_{\mathsf{range-key}}) = 1)$.
  3. If $b = 1$, then it outputs 1. Otherwise, it outputs 0.
- $\mathsf{Decrypt}(\boldsymbol{C}_\ell, \mathsf{dk}_{\boldsymbol{y}})$: Takes as input an n-vector ciphertext $\boldsymbol{C}_\ell := (C_{\ell,i})_{i \in [n]}$, a functional decryption key $\mathsf{dk}_{\boldsymbol{y}}$. It computes $[\alpha] = \sum_i [c_i] \cdot y_i - [\boldsymbol{u}_\ell^\top] \cdot \mathsf{dk}_{\boldsymbol{y}}$, and eventually solves the discrete logarithm to extract and return α.

Theorem 10 (Range-Verifiable Inner-Product MCFE). *The inner-product* MCFE *scheme in the construction above has correctness and verifiability for range predicates in the random oracle. More precisely,*

$$\mathsf{Adv}^{\mathsf{verif}}_{\mathsf{VMCFE}}(t) \leq n \cdot \mathsf{Adv}^{\mathsf{snd}}_{\mathit{NIZK}_{\mathsf{range-key}}}(t),$$

where

– $\mathsf{Adv}^{\mathsf{verif}}_{\mathsf{VMCFE}}(t)$ *is the best advantage of any PPT adversary running in time* t *against the verifiability game in Definition 7;*
– n *is a polynomially bounded number of senders;*
– $\mathsf{Adv}^{\mathsf{snd}}_{\mathit{NIZK}_{\mathsf{range-key}}}(t)$ *is the best advantage of any PPT adversary running in time* t *against the soundness of* $\mathit{NIZK}_{\mathsf{range-key}}$.

Proof. Let us prove the verifiability. We first show the correctness.

Correctness. When every input x_i and every entry y_i of the inner-product function are in the small range $[0, 2^m - 1]$ with a polynomially bounded m, then solving $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ from $[\langle \boldsymbol{x}, \boldsymbol{y} \rangle]$ can be done efficiently. The correctness of $\mathsf{Decrypt}$ is then implied by that of the underlying MCFE [CDG+18]. The correctness of $\mathsf{VerifyCT}$ is implied by that of $\mathsf{NIZK}_{\mathsf{range-key}}$ in Section 4.2. The correctness of $\mathsf{VerifyDK}$ is as shown in Section 4.3.

Verifiability. To prove that the advantage of any PPT adversary $\mathcal{A}$ in the verifiability game is negligible, we assume that $\mathcal{A}$ wins the games, and then show that the soundness of $\mathsf{NIZK}_{\mathsf{range-key}}$ must be broken for at least one instance. Note that the hash function $\mathcal{H}$ is modeled as a random oracle RO onto $\mathbb{G}^2$ in the verifiability game.

– For any $\mathsf{vk}_i \in \mathbb{G}^2$ produced by $\mathcal{A}$, there exists a unique vector $\boldsymbol{s}_i$ such that $\mathsf{vk}_i = ([\boldsymbol{v}^\top] \cdot \boldsymbol{s}_i, [\boldsymbol{v'}^\top] \cdot \boldsymbol{s}_i)$. Consequently, any ciphertext $[c_{\ell,i}]$ determines a unique value x_i such that $[c_{\ell,i}] = [\boldsymbol{u}_\ell^\top \boldsymbol{s}_i + x_i]$.
– For any functional decryption key $\mathsf{dk}_{\boldsymbol{y}_j} = (\mathsf{dk}_j, \boldsymbol{y}_j)$, we have the winning condition $\mathsf{VerifyDK}(\mathsf{pk}, \mathsf{dk}_{\boldsymbol{y}_j}, m) = 1$. Note that $\mathsf{pk} = (\mathsf{vk}_i)_i$ uniquely determines the tuple $(\boldsymbol{s}_i)_i$, then the validity of $\mathsf{dk}_{\boldsymbol{y}_j}$ with respect to pk implies that $\mathsf{dk}_j = \sum_{i=1}^m y_{j,i} \cdot \boldsymbol{s}_i$ where $y_{j,i} \in [0, 2^m - 1]$ for all i and j. (Section 4.3).
– For any $C_{\ell,i} = (\ell, [c_{\ell,i}], \pi^i_{\mathsf{range-key}})$, we have $\mathsf{VerifyCT}(\mathsf{vk}_i, C_{\ell,i}, m) = 1$ (the winning condition). Then $\pi^i_{\mathsf{range-key}}$ is a $\mathsf{NIZK}_{\mathsf{range-key}}$ accepting proof for all $i \in [n]$. Suppose that for every $[c_{\ell,i}]$, there exists a unique value x_i such that $x_i \in [0, 2^m - 1]$ and $[c_{\ell,i}] = [\boldsymbol{u}_\ell^\top \boldsymbol{s}_i + x_i]$. In this case, by the correctness, the decryption process with input $([c_{\ell,i}])_i$ and $\mathsf{dk}_{\boldsymbol{y}_j}$ will output $\langle \boldsymbol{x}, \boldsymbol{y}_j \rangle$ for all j. This contradicts that $\mathcal{A}$ wins the game.
– From the contradiction, there must exist at least one $[c_{\ell,i}]$ such that $[c_{\ell,i}] \neq [\boldsymbol{u}_\ell^\top \boldsymbol{s}_i + x_i]$ for all $x_i \in [0, 2^m - 1]$. In other words, the statement $([c_{\ell,i}], \mathsf{vk}_i, m)$ is not in the language $L_{\mathsf{range-key}}$, which is defined in Section 4.1. As $\mathsf{NIZK}_{\mathsf{range-key}}$ is a zero-knowledge argument for this language, the soundness is broken for this instance.

An adversary $\mathcal{B}$ against the soundness of $\mathsf{NIZK}_{\mathsf{range-key}}$ can be constructed as follows: $\mathcal{B}$ first guesses an index $i \in [n]$, then invokes $\mathcal{A}$, and outputs the i instance $([c_{\ell,i}], \mathsf{vk}_i, \pi^i_{\mathsf{range-key}})$ from $\mathcal{A}$'s output. In the case $\mathcal{A}$ wins the game, the probability that $\mathcal{B}$ breaks the soundness of $\mathsf{NIZK}_{\mathsf{range-key}}$ is at least $\frac{1}{n}$.
Concretely,

$$\frac{1}{n} \cdot \mathsf{Adv}^{\mathsf{verif}}_{\mathsf{VMCFE}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{snd}}_{\mathsf{NIZK}_{\mathsf{range-key}}}(\mathcal{B}).$$

By choosing the adversary $\mathcal{A}$ running in t that has the best advantage in the verifiability game, we have

$$\mathsf{Adv}^{\mathsf{verif}}_{\mathsf{VMCFE}}(t) \leq n \cdot \mathsf{Adv}^{\mathsf{snd}}_{\mathsf{NIZK}_{\mathsf{range-key}}}(t).$$

As long as the number of senders n is polynomially bounded, $\mathsf{Adv}^{\mathsf{verif}}_{\mathsf{VMCFE}}(t)$ is negligible. The proof is complete.

5.2 Security Analysis

Theorem 11 (Range-Verifiable Inner-Product MCFE). *The above inner-product* MCFE *scheme (see Section 5.1) is* IND*-secure under the* DDH *assumption, in the random oracle model. More precisely, we have*

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{VMCFE}}(t, q_E) \leq \mathsf{Adv}^{\mathsf{zk}}_{\mathit{NIZK}_{\mathsf{range-key}}}(t, q_E) + \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{MCFE}}(\hat{t}, q_E + 2n),$$

where

- $\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{VMCFE}}(t, q_E)$ *is the best advantage of any PPT adversary running in time* t *with* q_E *encryption queries against the* IND*-security game in Section 3.2;*
- $\mathsf{Adv}^{\mathsf{zk}}_{\mathit{NIZK}_{\mathsf{range-key}}}(t, q_E)$ *is the best advantage of any PPT adversary running in time* t *with* q_E *encryption queries against the zero-knowledge property of* $\mathit{NIZK}_{\mathsf{range-key}}$ *(Section 4.3);*
- $\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{MCFE}}(\hat{t}, q_E + 2n)$ *is the best advantage of any PPT adversary running in time* $\hat{t}$ *(which is* t *plus a negligible time of running a zero-knowledge simulator of* $\mathit{NIZK}_{\mathsf{range-key}}$*) with* $q_E + 2n$ *encryption queries against the* IND*-security of the* MCFE *scheme in [CDG+18].*

This theorem supports both adaptive encryption queries and adaptive corruptions.

Proof. We proceeds by using hybrid games: G_0 corresponds to the game as given in the security definition and G_1 corresponds to the game where the adversary $\mathcal{A}$ has an advantage bounded by the best advantage that any PPT adversary can get in the IND-security game of MCFE. We denote by $\mathsf{Adv}_{\mathsf{index}} := |\Pr[G_{\mathsf{index}}(\mathcal{A})|b = 1] - \Pr[G_{\mathsf{index}}(\mathcal{A})|b = 0]|$, where the probability is taken over the random coins of G_{index} and $\mathcal{A}$.

Game G_0: This is the IND-security game as given in the definition (Section 3.2).

Game G_1: From the zero-knowledge property of $\mathsf{NIZK}_{\mathsf{range-key}}$, a simulator $\mathcal{S}$ is invoked to provide proofs $\pi^i_{\mathsf{range-key}}$ on input $([c_{\ell,i}], \mathsf{vk}_i, m)$ for all $i \in [n]$. Then $|\mathsf{Adv}_1 - \mathsf{Adv}_0| \leq \mathsf{Adv}^{\mathsf{zk}}(t)$. We now construct an adversary $\mathcal{B}$ against the IND-security of the underlying MCFE with oracle access to $(\mathsf{QEncrypt}, \mathsf{QDKeyGen}, \mathsf{QCorrupt})$:

- Initialization:
  - upon receiving $\mathsf{MCFE.pp} = (\mathcal{G}, \mathcal{H})$, adversary $\mathcal{B}$ chooses randomly a pair of labels (ℓ_0, ℓ'_0) and sends $\mathsf{pp} = (\mathcal{G}, \mathcal{H}, \ell_0, \ell'_0)$ to $\mathcal{A}$;
  - $\mathcal{B}$ sends MCFE encryption queries for $(i, 0, 0, \ell_0)$ and $(i, 0, 0, \ell'_0)$ for all $i \in [n]$. An important point is that an encryption of 0 under the private encryption key $\mathsf{ek}_i = \boldsymbol{s}_i$ and the label ℓ_0 (or ℓ'_0) in the MCFE scheme [CDG+18] is equal to $([\boldsymbol{v}^\top] \cdot \boldsymbol{s}_i)$ (or $[\boldsymbol{v'}^\top] \cdot \boldsymbol{s}_i)$) in our construction.
  - Upon receiving all $[k_i] = \mathsf{QEncrypt}(i, 0, 0, \ell_0)$ and $[k'_i] = \mathsf{QEncrypt}(i, 0, 0, \ell'_0)$, $\mathcal{B}$ sets $\mathsf{pk} = ([k_i], [k'_i])_i$, and $\mathsf{vk}_i = ([k_i], [k'_i])$ for all $i \in [n]$. $\mathcal{B}$ sends $(\mathsf{pk}, (\mathsf{vk}_i)_i)$ to $\mathcal{A}$.
- Encryption queries (i, x^0, x^1, ℓ) from $\mathcal{A}$ such that $x^k \in [0, 2^m - 1]$ for all $k \in \{0, 1\}$: $\mathcal{B}$ sends (i, x^0, x^1, ℓ) to the MCFE encryption oracle and receives back $\mathsf{QEncrypt}(i, x^0, x^1, \ell) = [c_{\ell,i}]$. The simulator $\mathcal{S}$ is invoked to produce $\pi^i_{\mathsf{range-key}}$. $\mathcal{B}$ sends $(\ell, [c_{\ell,i}], \pi^i_{\mathsf{range-key}})$ to $\mathcal{A}$ as the query answer.
- Functional decryption key queries for $\boldsymbol{y}$ from $\mathcal{A}$: $\mathcal{B}$ sends $\boldsymbol{y}$ and receives $\mathsf{QDKeyGen}(f) = \mathsf{dk}_{\boldsymbol{y}}$ from the MCFE functional decryption key oracle. $\mathcal{B}$ sends $\mathsf{dk}_{\boldsymbol{y}}$ to $\mathcal{A}$ as the query answer.
- Corruption queries for an index i from $\mathcal{A}$: $\mathcal{B}$ sends and receives $\mathsf{QCorrupt}(i) = \mathsf{ek}_i$ from the MCFE corruption oracle. $\mathcal{B}$ sends $\mathsf{QCorrupt}(i)$ to $\mathcal{A}$ as the query answer.
- $\mathcal{B}$ outputs what $\mathcal{A}$ outputs for the guess on the bit b.

From this construction, we have $\mathsf{Adv}_1 \leq \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{MCFE}}(\hat{t}, q_E + 2n)$, where $\hat{t}$ includes the time for $\mathcal{B}$ to call $\mathcal{S}$. Therefore,

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{VMCFE}}(t, q_E) = \mathsf{Adv}_0 \leq \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{range-key}}}(t, q_E) + \mathsf{Adv}^{\mathsf{ind}}_{\mathsf{MCFE}}(\hat{t}, q_E + 2n).$$

Since the MCFE scheme in [CDG+18] is IND-secure under the DDH assumption in the random oracle, the proof is complete.

Additional cost for verifiability. We assume that $\mathsf{NIZK}_{\mathsf{range-key}}$ is instantiated with the Schnorr-based $\mathsf{NIZK}_{\mathsf{key}}$ (Appendix B) and the Bulletproof for range [BBB+18] (the adapted version for $\mathcal{R}_{\mathsf{range}+}$ in Section 4.2). Since the scalar operation in $\mathbb{Z}_p$ is cheap compared to the exponentiation in $\mathbb{G}$, so we do not detail them here.

- Verification time: for all n cipherexts under the same label, $\mathsf{NIZK}_{\mathsf{key}}$ costs about 3 multi-exponentiations of size $3 + 2n$, Bulletproof costs a multi-exponentiation of size $2m + 3 + n(2\log_2 \lceil m \rceil + 5)$. The number of scalar operations is $\mathcal{O}(n \cdot m)$. For the functional key verification, the cost is $2n + 4$ group exponentiations.
- Encryption time: $\mathsf{NIZK}_{\mathsf{key}}$ costs 7 exponentiations, Bulletproof costs about $12m + 10$ exponentiations. The number of scalar operations is $\mathcal{O}(m)$.
- Ciphertext size: each $\pi^i_{\mathsf{range-key}}$ has the size of $2\log_2 \lceil m \rceil + 7$ elements in $\mathbb{G}$ and 10 elements in $\mathbb{Z}_p$.

5.3 Verifiable Decentralized MCFE for Inner Product

A decentralized multi-client functional encryption (DMCFE) [CDG+18] is an MCFE in which the process of functional key generation is not controlled by any authority, but completely under the control of each of the clients.

Based on the underlying MCFE scheme in [CDG+18], all the senders can generate their own secret $\boldsymbol{s}_i \in \mathbb{Z}_p^2$, and set $\mathsf{ek}_i = \mathsf{sk}_i = \boldsymbol{s}_i$. They have to additionally execute a multi-party protocol during the setup to obtain for each one a secret correlated randomness. With his own randomness, a sender can encode under a functional label $\ell_{\boldsymbol{y}}$ his partial functional key, which is the term $y_i \cdot s_i$ in $\mathsf{dk}_{\boldsymbol{y}} = \sum_i y_i \cdot s_i$, as dk_i. When the receiver collects the encodings of all senders under the same label, he can reveal $\mathsf{dk}_{\boldsymbol{y}}$ in the form of a discrete logarithm. For the functional key in this form, the decryption can then be done by using a bilinear map. In a later work, the multi-party protocol can be replaced by a decentralized sum [CDSG+20] to remove the interactivity between all senders and to allow directly revealing $\mathsf{dk}_{\boldsymbol{y}}$.

From the above framework that is instantiated with the decentralized sum, our verifiable inner-product MCFE scheme (in Section 5.1) can be automatically extended to a decentralized one. Namely, each ek_i is still committed as $\mathsf{com}_{\mathsf{ek}_i} = ([\boldsymbol{v}^\top] \cdot \boldsymbol{s}_i, [\boldsymbol{v'}^\top] \cdot \boldsymbol{s}_i)$. The verification key $\mathsf{vk}_i = \mathsf{com}_{\mathsf{ek}_i}$ and $\mathsf{pk} = (\mathsf{com}_{\mathsf{ek}_i})_i$. The algorithms $(\mathsf{Encrypt}, \mathsf{VerifyCT}, \mathsf{Decrypt})$ remain the same. The only difference is in the verification of functional keys: the receiver has to wait for all (possibly maliciously generated) encodings dk_i, and then use them to reveal a functional key $\mathsf{dk}_{\boldsymbol{y}}$. Knowing $(\boldsymbol{y}, \mathsf{dk}_{\boldsymbol{y}}, (\mathsf{com}_{\mathsf{ek}_i})_i)$, the receiver can verify $\mathsf{dk}_{\boldsymbol{y}}$ as in the $\mathsf{VerifyDK}$ algorithm. The key verification outputs 1 if $\mathsf{VerifyDK}(\mathsf{pk}, \mathsf{dk}_{\boldsymbol{y}}, m) = 1$, and otherwise. As a consequence, the verifiability of this decentralized scheme, where $\mathcal{A}$ can corrupt all senders, is implied by that of the verifiable inner-product MCFE scheme.

Extending to a verifiable DMCFE scheme as above is more efficient than verifying each functional key share dk_i. Note that the generation of dk_i takes as input a secret correlated randomness output from a multi-party protocol or from a decentralized sum, which requires running a pseudo-random function on the exchanged keys. Therefore, verifying each dk_i may be highly inefficient. A drawback of this shortcut is that a receiver cannot detect which sender sent a malicious key share in the case the combined $\mathsf{dk}_{\boldsymbol{y}}$ is not valid.

6 Verifiable MCFE with Incomplete Ciphertexts

Our Verifiable MCFE has a feature of detecting malicious users: VerifyCT takes as input each individual ciphertext of each client, and VerifyDK takes as input each functional decryption key of each key authority. Therefore, the sender of a rejected ciphertext (or key) must be malicious. We demonstrate in this section that it is possible to detect malicious users even in the case of incomplete ciphertexts.

In the IND-security game for verifiable MCFE in Section 3.2, the condition 2 defining the adversary's output β states that if one ciphertext of an honest sender is queried for a label ℓ, then all the ciphertexts under the same label of other honest senders must be queried. Intuitively, the leakage from incomplete ciphertexts is at first not considered in the security game.

In [CDSG+20], the authors introduced a primitive called All or Nothing Encapsulation (AoNE) to lift this condition for MCFE. The AoNE 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. Given an IND-secure AoNE scheme, the idea is that by adding a layer of this AoNE encryption on the underlying MCFE ciphertexts, the answers for incomplete encryptions queries can be simulated to be encapsulations of a fixed value, rather than the encapsulations of the underlying incomplete MCFE ciphertexts.

This technique can be adapted to our verifiable MCFE scheme in Section 5.1 for the same purpose. The verifiability will remain, as the ciphertexts of the underlying MCFE scheme $[c_{\ell,i}]$ have to be revealed from encapsulations and verified by VerifyCT with a proof $\pi^i_{\mathsf{range-key}}$. However, when the receiver wants to detect a malicious sender in this case, a trivial solution is that each sender has to provide a proof that his encapsulation honestly encrypts (under a committed AoNE secret key) a ciphertext $[c_{\ell,i}]$ and that $[c_{\ell,i}]$ honestly encrypts an input $x_i \in [0, 2^m - 1]$ under a committed encryption key ek_i. A NIZK proof for the last statement may be prohibitively expensive. Before proposing an efficient approach to circumvent this issue, we briefly recall the pairing-based AoNE construction in [CDSG+20], which serves as a building block.

- $\mathsf{AoNE.SetUp}(\lambda)$: Generates a pairing group $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, P_1, P_2, e) \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$, 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 $t_i \xleftarrow{\$} \mathbb{Z}_p$ 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 $r_i \xleftarrow{\$} \mathbb{Z}_p$ 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 $x_i = \mathsf{SDec}(K_{i,\ell}, c_i)$.

Detecting malicious users. Our ciphertext verification will consist of two steps when the above AoNE scheme is instantiated:

- AoNE share validation: each sender sends additionally with his encapsulation a Schnorr-based 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.
- MCFE ciphertext verification: each x_i decrypted from AoNE is parsed as an MCFE ciphertext under the label ℓ, namely $x_i = [c_{\ell,i}]$. The receiver combines $C_{\ell,i} = (\ell, [c_{\ell,i}], \pi^i_{\mathsf{range-key}})$ and computes $\mathsf{VerifyCT}(\mathsf{vk}_i, C_{\ell,i}, m)$ as in Section 5.1. If all $C_{\ell,i}$ are valid, returns 1, else returns 0.

The idea of this two-step ciphertext verification is from the fact that in the 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 designed to detect malicious senders who want to distort the symmetric key of other honest senders. The second step is designed to detect malicious senders who give maliciously generated ciphertexts on purpose or by acting maliciously during the AoNE encryption.

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

7 Conclusion

The interest in functional encryption has significantly increased in recent years, particularly in multi-user settings where data inputs are provided by different users, sometimes in competition. It is important to verify the accuracy of the inputs and detect malicious users in such scenarios. This new direction of research has garnered interest from both academic and industrial sectors [BGJS16, BGL+22], and we aim at promoting to its development. There are numerous open problems that need to be addressed, ranging from practical solutions for simple functionalities to more general frameworks. In this paper, we leave the detection of malicious users in Verifiable DMCFE, when generating their key shares, as an open problem. A more general problem is to design practical solutions for verifiability for larger classes of functions, such as quadratic functions.

Acknowledgments

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

ABM+20. M. Abdalla, F. Bourse, H. Marival, D. Pointcheval, A. Soleimanian, and H. Waldner. Multi-client inner-product functional encryption in the random-oracle model. In *SCN 20*, *LNCS* 12238, pages 525–545. Springer, Heidelberg, September 2020.

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.

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.

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`.

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.

WC22. N. Wang and S. C.-K. Chau. Flashproofs: Efficient zero-knowledge arguments of range and polynomial evaluation with transparent setup. Cryptology ePrint Archive, Paper 2022/1251, 2022. `https://eprint.iacr.org/2022/1251`.

A More Definitions

A.1 Pairing Groups

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 $[a]_1$, $[a]_2$, one can efficiently compute $[ab]_T$ using the pairing e.

A.2 Non-interactive Zero-Knowledge Proofs

Definition 12 (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 $\mu(\lambda)$:

$$\Pr\begin{bmatrix}\sigma \leftarrow \mathsf{SetUp}(\lambda), \\ (u,\pi) \leftarrow \mathcal{A}(\sigma;r) \\ w \leftarrow K(\sigma,u,\pi;r)\end{bmatrix} : \begin{matrix}\mathsf{Verify}(u,\pi)=1 \\ \wedge\, \mathcal{R}(u;w)=0\end{matrix}\Bigg] \leq \mu(\lambda).$$

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

Definition 13 (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 Verification for Inner-Product MCFE: Schnorr-based protocol

In this part, we describe the Schnorr-based protocol $\mathsf{NIZK}_{\mathsf{range-key}}$ for $\mathcal{R}_{\mathsf{key}}$ in Section 4.2. 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

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

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 scheme into non-interactive mode, one can apply the Fiat-Shamir heuristic.