Privacy-Preserving Federated Learning from Partial Decryption Verifiable Threshold Multi-Client Functional Encryption

Minjie Wang[a], Jinguang Han[a,b] and Weizhi Meng[c]

[a]School of Cyber Science and Engineering, Southeast University, Nanjing, 210096, China
[b]Wuxi Campus, Southeast University, Wuxi, 214125, China
[c]School of Computing and Communications, Lancaster University, Lancaster, United Kingdom

ARTICLE INFO

Keywords:
Multi-client functional encryption
Secure aggregation
Verifiability
Privacy-preserving federated learning

ABSTRACT

In federated learning, multiple parties can cooperate to train the model without directly exchanging their own private data, but the gradient leakage problem still threatens the privacy security and model integrity. Although the existing scheme uses threshold cryptography to mitigate the inference attack, it can not guarantee the verifiability of the aggregation results, making the system vulnerable to the threat of poisoning attack. Considering these issues, we construct a partial decryption verifiable threshold multi-client functional encryption scheme to prevent gradient leakage and inference attacks, and apply it to federated learning to implement verifiable threshold secure aggregation protocol (VTSAFL). Furthermore, to resist poisoning attacks, VTSAFL empowers clients to verify aggregation results even when up to $t-1$ aggregators collude, concurrently minimizing both computational and communication overhead. The size of the functional key and partial decryption results of the scheme are constant, which provides efficiency guarantee for large-scale deployment. The experimental results on MNIST dataset show that VTSAFL can achieve the same accuracy as existing schemes, while reducing the total training time by more than 40%, and reducing the communication overhead by up to 50%.

1. Introduction

Federated Learning (FL) is a distributed machine learning framework, which was first proposed by Google in 2016 [1] to solve the growing problem of data privacy. Its core design allows multiple participants to collaborate to train the global model under the coordination of the central aggregation server, while ensuring that the original data of all participants remains localized and never shared. In federated learning, the main role of the central server is to aggregate the model parameters of participants, rather than collect their private data. This model effectively reduces the inherent privacy risks associated with traditional centralized machine learning.

However, FL introduces new privacy vulnerabilities. Adversaries may infer private data from these uploaded updates. For instance, attackers could potentially deduce true labels of input data from leaked gradients or even employ leaked models [2], [3], [4], [5], [6], possibly using Generative Adversarial Networks (GANs) [7], to reconstruct datasets that approximate the clients' private data, resulting in serious privacy breaches. To address these privacy challenges, several secure aggregation schemes have been proposed to facilitate privacy-preserving federated learning (PPFL), employing diverse primitives. As shown by Table 1, existing PPFL approaches exhibit significant diversity in architecture, trust assumptions, and cryptographic methods. Many early schemes [8], [9], [10], [11], [12], [13] equip a single, Honest-but-Curious (HbC) aggregator with cryptographic techniques, e.g., Homomorphic Encryption (HE) or Functional Encryption (FE). While offering partial protection against gradient inference or "mix-and-match" attacks, these often face the risk of intermediate model leakage and suffer from a single point of failure. Because FE can prevent the aggregator from accessing client local models, but the aggregator can still access the intermediate model via the functional key in federated learning training. On the other hand, although HE protects the intermediate model through direct ciphertext aggregation, it is susceptible to replay attacks and suffers from a single point of failure.

Subsequent works introduced multi-server architectures, such as two-server secure multi-party computation (MPC) [14] or trust execution environment (TEE) [15], [16], often retaining the HbC assumption. More recently, the scheme TAPFed [17] addresses adversarial aggregators using multiple-aggregator systems and threshold MCFE. Under a multi-aggregator architecture, the intermediate model cannot be obtained by any independent aggregator, thereby mitigating intermediate model leakage. Furthermore, the threshold mechanism in TAPFed mitigates the single point of failure issue, as the training task can be processed by t active aggregators. This setup is also resilient to collusion attacks from up to $t-1$ aggregators. However, under the assumption of adversarial aggregators, certain aggregators may not adhere to established training protocols and could attempt to tamper with aggregation results. For instance, they might maliciously alter these results to launch poisoning attacks or directly return random values to evade substantial computational overhead. In such circumstances, previous schemes cannot guarantee the validity of the decrypted results furnished by these aggregators, and clients are subsequently unable to discern the validity of these results, thereby failing to achieve the intended training outcomes.

To overcome limitations in prior work, we propose VTSAFL, a novel verifiable threshold secure aggregation scheme for federated learning. VTSAFL ensures aggregation result confidentiality via a threshold mechanism and

provides result verifiability for all participants. The scheme is efficient in computation and communication, particularly in large-scale client scenarios.

This work's main contributions include:

1. Verifiability of the aggregation results: We construct a partial decryption verifiable threshold multi-client functional encryption scheme, and implement a new privacy preserving federated learning framework called VTSAFL. In this system, each aggregator not only calculates the partial decryption result, but also calculates the corresponding correctness proof using the DLEQ protocol [18]. Upon receiving these results and proofs, clients run a verification algorithm to check the validity of the computation performed by each aggregator. If an aggregator provides a malicious or incorrect result, the verification will fail, and its result will be discarded by the clients, thus ensuring the integrity of the global model aggregation.
2. High runtime efficiency and low communication overhead: In our framework, the threshold functional encryption scheme employed can reduce the size of both the functional key and the partial decryption results to a constant level, rather than being proportional to the number of clients. This offers a distinct advantage in large-scale privacy-preserving federated learning tasks involving a substantial number of clients. Additionally, we utilize a multi-secret sharing scheme for the distribution of functional keys, which further helps to minimize the communication overhead.
3. Comparison and Implementation: We implemented the VTSAFL framework and compared it with existing schemes. Our experiments are carried out on MNIST and CIFAR10 datasets. The final experimental results show that our VTSAFL framework has improved in training time and communication overhead, and is optimized for scenarios with a large number of participants. The framework not only achieves the same model performance, but also provides the verifiability of aggregate results.
4. Security Analysis: We present security model for our MCFE scheme and conduct a formal security proof. Subsequently, we analyze the security of the VTSAFL framework, proving its resilience against various attacks.

2. Related Work

2.1. Multi-input Function Encryption

Functional encryption (FE) [19],[20] is an example of public key cryptosystem. It enables all parties to encrypt data so that authorized entities holding specific function keys can calculate specific functions on the ciphertext. This calculation shows the output of the function $f(x)$, without disclosing any additional information about the underlying plaintext x. The setting of Fe scheme usually involves a trusted institution (TA). TA is responsible for generating master key msk and corresponding master public key mpk. mpk is provided to one or more entities that use it to encrypt their input. For the given function $f(\cdot)$, TA uses the msk derived function key dk_f. Any designated decryptor with dk_f can calculate the function f on the ciphertext $enc(x)$to obtain the result $f(x)$. It is critical that this assessment is performed directly on encrypted data to ensure the confidentiality of plaintext x throughout the process.

Although the standard FE can operate on large inputs (such as high-dimensional vectors), it essentially assumes that the entire input comes from one party. Therefore, all components of the input vector must be simultaneously provided and encrypted by one entity. For many practical applications, this model is not enough. These applications usually rely on aggregating information from multiple, different and possibly untrusted parties. To address this limitation, Goldwasser et al. [21] introduced multiple input function encryption (MIFE). MIFE extends FE to calculate functions on inputs provided by multiple parties. However, MIFE scheme is vulnerable to "hybrid matching" attack. The reason for this vulnerability is that during decryption calculation, ciphertext from different clients (or different time periods) can be combined. For example, consider a scenario where client 1 provides ciphertext for input $\{x_0, x_1\}$ and client 2 provides ciphertext for $\{y_0, y_1\}$. For all possible combinations of $(i, j) \in \{0, 1\}^2$, the calculator may be able to calculate $f(x_i, y_j)$, resulting in serious information leakage. To mitigate this attack, a multi client function encryption (MCFE) [22],[23] is proposed. MCFE enhances MIFE by introducing labels. In the MCFE scheme, each encrypted message is associated with a specific tag. The decryption process is limited, so that the ciphertext can be combined only when sharing the same token, so as to prevent accidental "mixed-and-match" calculation.

2.2. Privacy Preserving Federated Learning

In order to achieve privacy preserving federated learning (PPFL), a variety of security aggregation schemes are introduced, which are characterized by the use of different encryption primitives: (1) Differential privacy (DP) reference [24],[25]: DP is a non encrypted method, and users add noise locally to their data before sending the "randomized" version to the aggregator. This process is designed to prevent anyone (including aggregators) from recovering private data. However, a key disadvantage is that the added noise will reduce the prediction accuracy of the obtained intermediate model; (2) Multi-party computation (MPC) [14],[26]: most solutions in this domain use secret sharing. Users usually use a one-time pad to mask messages for the aggregator. Decryption requires the aggregator to collect a sufficient number of these shares. Although this method maintains accuracy, it is different from the DP based method in that it needs to increase the interaction between users and aggregators, so as to increase the communication cost and the risk of disconnection; (3) Trust execution environment (TEEs) [15],[16]: the scheme based on trusted execution environment uses Intel SGX, amd PSP and arm TrustZone to create isolated

Table 1
Comparison of Different Approaches in Privacy-preserving Federated Learning

Scheme	Aggregator		Approaches	Attack Resistance			Single point of failure
	architecture	assumption		gradient inference attacks	"mix-and-match" attack	poisoning attack †	
[8],[9]	single	HbC	HE	✓	✗	✗	✗
[10]	single	HbC	MIFE	✗	✗	✗	✗
[11]	single	HbC	2DMCFE	✓	✓	✗	✗
[12],[13]	single	HbC	DMCFE	✗	✓	✗	✗
[14]	two-server	HbC	MPC	✓	✓	✗	✗
[17]	multiple	Adversarial	TMCFE	✓	✓	✗	✓
Our work	multiple	Adversarial	VTMCFE	✓	✓	✓	✓

† The defense against poisoning attacks specifically targets those from adversarial aggregators only

safe enclaves relying on dedicated hardware. However, these solutions face practical limitations, especially the limited availability of required hardware and the limited memory space for secure computing; (4) Homomorphic encryption (HE) [8, 9]: it allows direct public calculation of encrypted data. In this example, each user encrypts its local model, and then the aggregator can perform arithmetic operations on this model. However, the based solutions are often plagued by computational constraints, making them unsuitable for large-scale security aggregation of model updates. In addition, they also have potential isolation or replay attack vulnerabilities. Other schemes explore different architectures. For example, SVFLC [27] proposes a chain aggregation framework, which combines lightweight mask technology to resist collusion attacks, and uses homomorphic hash functions to achieve verifiable aggregation for the server, but cannot resist the poison attack from malicious aggregators. In order to prevent malicious forgery of aggregation results in the cloud, FVFL [28] introduces Lagrange interpolation polynomial and secret sharing to implement an effective verification mechanism. However, this scheme still relies on a centralized server. If the server cannot follow the rules, the whole training process will be difficult to implement.

In recent years, FE based solutions have attracted more and more attention. Compared with the above technologies, the method based on multi-input function encryption (MIFE) has advantages in computational efficiency and communication efficiency. In addition, unlike DP, MIFE based solutions do not compromise model accuracy or rely on other dedicated hardware. Xu et al. [10] represents the first use of MIFE. Its purpose is to prevent a curious aggregator and colluding users from inferring private information. Nevertheless, their proposed scheme is still vulnerable to "Mix-and-Match" attacks and shows vulnerabilities related to the disclosure of intermediate models. Chang et al. [11] designed 2DMCFE and applied it to federal learning. By adding tags to the model ciphertext of the same training batch, they solved the "Mix-and-Match" attack. In addition, in each round of training, clients share a session key to reduce the risk of intermediate model leakage.

Another way to mitigate intermediate model leakage is to use multiple aggregators instead of relying on a single centralized aggregator. Xu et al. [17] First proposed the definition of threshold function encryption, and introduced a new privacy preserving federated learning framework called TAPFed, in which security aggregation can be achieved through decentralized multiple aggregators, which are independent of each other and do not need peer-to-peer communications. At the same time, the scheme also allows the existence of malicious entities. In this scheme, shamir secret sharing was employed to distribute the functional key. Each aggregator receives only a secret share and computes a partial aggregation result, thereby preventing any single aggregator from directly accessing the intermediate model. However, clients are unable to verify the validity of the aggregation results. This verification is crucial, as the scheme permits the existence of malicious aggregators, which might launch poisoning attacks to disrupt the training process or even return random values instead of valid aggregated results to evade computational costs. Concurrently, in the threshold multi-client functional encryption [17], the length of the functional key and the partial decryption results are both proportional to the number of clients. Consequently, this approach is not well-suited for large-scale training tasks that involve a substantial number of clients.

3. Preliminaries

3.1. Notions

For clarity and convenience, we summarize the main notations used throughout this paper in Table 2 including the key parameters and variables related to the system architecture, federated learning process, and the underlying cryptographic scheme.

3.2. Complexity Assumptions

Definition 1 (DDH Assumption). Let $\mathbb{G}$ be a cyclic group of prime order p with generator g. The decisional Diffie-Hellman (DDH) assumption states that no probabilistic polynomial time (PPT) adversary $\mathcal{A}$ can distinguish between the

Table 2
Notations Summary

Notation	Description
λ	The security parameter
n	Total number of clients
s	Total number of aggregators
t	Threshold number of aggregators
K	Maximum number of training rounds
p_i	The i-th client
a_j	The j-th aggregator
$\theta_{p_i}^{(k)}$	Local model of client p_i in round k
$\theta_G^{(k)}$	Aggregated global model in round k
$l^{(k)}$	A unique public label for round k
pp	Public parameters of the cryptosystem
msk	Master secret key
ek_i	Encryption key for client p_i
dk_j	Decryption key share for aggregator a_j
$ct_{p_i}^{(k)}$	Ciphertext of the local model in round k
ct'_j	A partial decryption result
Π_j	A proof of correctness for decryption
$[\beta]$	g^β in a cyclic group $\mathbb{G}$ with the generator g
PPT	Probabilistic Polynomial Time
$\langle\cdot,\cdot\rangle$	The inner product operation
$\xleftarrow{\$}$	Sampling uniformly at random

following two distributions with a non-negligible advantage:

$$D_{real} = \left\{ (g^a, g^b, g^{ab}) \mid a, b \xleftarrow{\$} \mathbb{Z}_p \right\}$$
$$D_{rand} = \left\{ (g^a, g^b, g^r) \mid a, b, r \xleftarrow{\$} \mathbb{Z}_p \right\}$$

More formally, let P_{real} denote the probability that $\mathcal{A}$ outputs 1 when given an element from D_{real}, and P_{rand} be the probability that $\mathcal{A}$ outputs 1 when given an element from D_{rand}:

$$P_{real} = \Pr_{a,b \xleftarrow{\$} \mathbb{Z}_p} [\mathcal{A}(g, g^a, g^b, g^{ab}) = 1]$$
$$P_{rand} = \Pr_{a,b,r \xleftarrow{\$} \mathbb{Z}_p} [\mathcal{A}(g, g^a, g^b, g^r) = 1]$$

The DDH assumption holds if the advantage, defined as

$$Adv_{\mathcal{A}}^{DDH} = |P_{real} - P_{rand}| \leq \epsilon(\lambda)$$

is negligible.

Definition 2 (Multi-DDH Assumption[23]). The Multi-DDH assumption, as introduced in [23], is that for PPT adversary $\mathcal{A}$ operating within t, its advantage in distinguishing the distributions $\mathcal{D}_m$ from $\mathcal{D}'_m$ is bounded. These distributions are defined over a group $\mathbb{G}$ as follows:

$$\mathcal{D}_m = \{(X, (Y_j, Z_j)) \mid X, Y_j \xleftarrow{\$} \mathbb{G}, Z_j = \mathrm{CDH}(X, Y_j)\}$$
$$\mathcal{D}'_m = \{(X, (Y_j, Z_j)) \mid X, Y_j, Z_j \xleftarrow{\$} \mathbb{G}\}$$

This advantage is formally bounded by $\mathsf{Adv}_{\mathcal{G}}^{\mathsf{ddh}}(t + 4m \times t_{\mathbb{G}})$, where $j = 1, \ldots, m$, and $t_{\mathbb{G}}$ represents the computational cost required for a single exponentiation within the group $\mathbb{G}$.

3.3. Homogeneous linear recursions

Homogeneous linear recursions (HLR) are used in multi-secret sharing [29].

Definition 3 (HLR). Let t be a positive integer, and $b_0, \ldots, b_{t-1}, z_1, \ldots, z_t \in \mathbb{R}$. A homogeneous linear recursion (HLR) of degree t is defined by:

$$\begin{cases} w_0 = b_0, w_1 = b_1, \ldots, w_{t-1} = b_{t-1} \\ w_{i+t} + z_1 w_{i+t-1} + \cdots + z_t w_i = 0, \quad i \geq 0 \end{cases}$$

The associated auxiliary equation is:

$$x^t + z_1 x^{t-1} + \cdots + z_{t-1} x + z_t = 0.$$

Suppose $\beta_1, \ldots, \beta_l$ are the distinct roots of the auxiliary equation with multiplicities $k_1, \ldots, k_l$ (where $\sum_{j=1}^{l} k_j = t$). The general solution for w_i is:

$$w_i = \sum_{j=1}^{l} p_j(i) \beta_j^i,$$

$p_j(i)$ is a polynomial function of i with degree at most $k_j - 1$.

If the auxiliary equation has a single root β with multiplicity t, the solution simplifies to $w_i = p(i)\beta^i$, where $p(i)$ is a polynomial in i of degree at most $t - 1$.

3.4. Secret Sharing

Our multi-secret sharing construction is inspired by the HLR-based PVMSS scheme proposed in [29], with modifications to suit the federated learning context. This scheme allows a dealer to distribute m secrets among s participants. It is a (t, s)-threshold scheme, where any group of t or more participants can reconstruct the secrets. The construction relies on an HLR with a single root of multiplicity t. The scheme consists of three algorithms: Setup, ShareDistribution, and Reconstruction.

Setup. The dealer first establishes the public parameters. Given a threshold t and number of participants s, the dealer chooses a large prime p and a random element $\alpha \in \mathbb{Z}_p$. Then, the dealer defines the characteristic polynomial $P(x) = (x - \alpha)^t \pmod p$ and expands it to obtain the coefficients $a_1, \ldots, a_t$. These coefficients define the HLR that will be used. The public parameters are $(p, \alpha, t, s, \{a_i\}_{i \in [t]})$.

ShareDistribution. To share m secrets $(\sigma_1, \ldots, \sigma_m) \in \mathbb{Z}_p^m$ where $m < t$, the dealer first constructs the initial state of the sequence $(w_0, \ldots, w_{t-1})$. The secrets are embedded as the first m terms, $w_i = \sigma_{i+1}$ for $i \in \{0, \ldots, m-1\}$, and the remaining $t - m$ terms are chosen randomly, $w_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in \{m, \ldots, t-1\}$. Then, using the HLR relation, the

dealer computes the subsequent terms. The share for the j-th participant is $sh_j = w_{t+j-1}$.

$$w_{i+t} + a_1 w_{i+t-1} + \cdots + a_t w_i \equiv 0 \pmod{p}$$

Reconstruction. Any group of t participants, holding a set of shares $\{sh_j\}_{j \in S'}$ where $|S'| = t$, can reconstruct the secrets. Based on Theorem 3, each sequence term satisfies $w_i = q(i) \cdot \alpha^i$ for a polynomial $q(i)$ of degree at most $t-1$. Each participant $j \in S'$ uses their share $sh_j = w_{t+j-1}$ to compute a point on this polynomial: $q(t+j-1) = sh_j / \alpha^{t+j-1}$. With t such distinct points, the group can use Lagrange interpolation to uniquely determine the polynomial $q(i)$. Finally, they recover the secrets by evaluating $q(i)$ at the first m indices: $\sigma_{i+1} = q(i) \cdot \alpha^i \pmod{p}$ for $i \in \{0, \ldots, m-1\}$.

3.5. Discrete Logarithms Equality Protocol

The Chaum-Pedersen protocol [18] is a proof of knowledge for the equality of two discrete logarithms. It allows a prover (P) to convince a verifier (V) that $y = g^x$ and $t = h^x$ share the same secret exponent $x \in \mathbb{Z}_q$, without revealing x.

The protocol operates in a cyclic group G_q of prime order q with distinct generators g and h. The proof, Π : Pok $\{(x) : y = g^x \wedge t = h^x\}$, proceeds as follows:

1. Commitment: P selects a random $r \in \mathbb{Z}_q$, computes $a_1 = g^r$ and $a_2 = h^r$, and sends (a_1, a_2) to V.
2. Challenge: V sends a random challenge $c \in \mathbb{Z}_q$ to P.
3. Response: P computes $z = r + cx \pmod{q}$ and sends z to V.
4. Verification: V accepts if $g^z = a_1 y^c$ and $h^z = a_2 t^c$.

3.6. Threat Model

We consider the following threat model:

- *Adversarial aggregator:* Unlike many existing Privacy-Preserving Federated Learning (PPFL) schemes, which assume aggregators are 'honest-but-curious' and will adhere to computational protocols, our model permits up to $t-1$ adversarial aggregators to participate and collude. This approach aligns with the assumptions in TAPFed. Furthermore, these aggregators may attempt to tamper with the aggregation results or deviate from the correct computation prescribed by the training rules to disrupt the normal training process.
- *Trusted authority:* A dependency on a trusted authority is a common feature of most cryptography-based privacy-preserving federated learning schemes, and our solution is no exception. In our framework, this authority handles the configuration of the cryptosystem and the management of keys in training.

We assume that the communications between participants occur over secure channels, thereby effectively preventing eavesdropping attacks. Our work focuses on the privacy leakage risks posed by adversarial aggregators and on providing clients with verifiability of the aggregation results to ensure they can obtain the intended training outcomes.

4. Partial Decryption Verifiable Threshold MCFE

4.1. Definition

Definition 4 (Partial Decryption Verifiable Threshold MCFE). *A t-of-s Partial Decryption Verifiable Threshold MCFE (VTMCFE) scheme is comprised of the following algorithms:*

- Setup (λ, t, s, n): Takes as input a security parameter λ, threshold t, total number of clients s, and number of encryption entities n. This algorithm outputs public parameters pp, a master secret key msk, and a set of encryption keys $\{ek_i\}_{i \in \{1,\ldots,n\}}$.
- DKeyGen $(pp, msk, \boldsymbol{y})$: Takes as input public parameters pp, master secret key msk, and a function vector $\boldsymbol{y}$. This algorithm generates and outputs a set of functional decryption keys $\{dk_j\}_{j \in \{1,\ldots,s\}}$.
- Encrypt (ek_i, x_i, l): Takes as input an encryption key ek_i (for entity i), a message x_i, and a label l. It outputs a ciphertext ct_i.
- PaDecrypt $(pp, \{ct_i\}_{i \in \{1,\ldots,n\}}, \boldsymbol{y}, dk_j, S)$: Takes as input public parameters pp, a set of ciphertexts $\{ct_i\}_{i \in \{1,\ldots,n\}}$, function vector $\boldsymbol{y}$, functional decryption key dk_j for entity j, and a subset S of decryption entities. This algorithm computes and outputs a partial decryption result (ct'_j, Π_j), where Π_j is a proof of correctness.
- Verify $(pp, \{ct'_j, \Pi_j\}_{j \in \{1,\ldots,s'\}})$: Takes as input public parameters pp and a set of s' partial decryption results $\{ct'_j\}$ with their corresponding proofs $\{\Pi_j\}$. This algorithm verifies the correctness of the partial decryptions and outputs a decision (e.g., 1 for valid, 0 otherwise).
- CombineRecovery $(pp, \{ct'_j\}_{j \in \{1,\ldots,s'\}}, l)$: Takes as input public parameters pp, a set of s' valid partial decryption results $\{ct'_j\}$ (which have passed verification), and the label l. This algorithm combines these results to recover and output the final function result $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$.

Correctness. A Partial Decryption Verifiable Threshold MCFE (VTMCFE) scheme is correct, if

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(pp, \{ct'_j, \Pi_j\}_{j \in S}) \to 1 \\ \quad \wedge \\ \mathsf{CombineRecovery}(pp, \{ct'_j\}_{j \in S}, l) \\ \to \langle \boldsymbol{x}, \boldsymbol{y} \rangle \end{array} \middle| \begin{array}{l} \mathsf{Setup}(\lambda, t, s, n) \to (pp, \\ msk, \{ek_i\}_{i \in [n]}); \\ \mathsf{DKeyGen}(pp, msk, \boldsymbol{y}) \\ \to \{dk_j\}_{j \in [s]}; \\ \mathsf{Encrypt}(ek_i, x_i, l) \to ct_i; \\ \mathsf{PaDecrypt}(pp, \{ct_i\}_{i \in [n]}, \\ \boldsymbol{y}, dk_j, S) \to (ct'_j, \Pi_j) \end{array}\right] = 1.$$

4.2. Security Model

Definition 5 (IND-Security Game). *Adapting the foundational definition from [23], we formalize the IND-Security of our scheme over n senders using the following game, which models the interaction between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.*

$QSetup(\lambda)$: The challenger $\mathcal{C}$ executes $(pp, msk, \{ek_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(\lambda)$ to generate the parameters and master secret key. $\mathcal{C}$ also samples a random challenge bit $b \leftarrow \{0,1\}$, which will be used to determine whether to encrypt the challenge message x^0 or x^1 in the subsequent $QEncrypt$ queries. The public parameters pp are then sent to the adversary $\mathcal{A}$.

$QEncrypt(i, x^0, x^1, l)$: The adversary $\mathcal{A}$ can make unlimited, adaptive queries to a Left-or-Right encryption oracle. Upon querying (i, x^0, x^1, l), the challenger $\mathcal{C}$ computes $ct_{l,i} \leftarrow \mathsf{Encrypt}(ek_i, x^b, l)$ and sends $ct_{l,i}$ to $\mathcal{A}$. Subsequent queries for an identical (l, i) pair are disregarded.

$QDKeyGen(f, j)$: $\mathcal{A}$ can adaptively select a function f and query the $\mathsf{DKeyGen}(msk, f)$. For single function f, this query is permitted up to $t-1$ times, with each successful query yielding the functional decryption key dk_f. Once the $t-1$ query limit for a specific f is reached, all future queries for that same function f are denied.

$QCorrupt(i)$: The adversary $\mathcal{A}$ is permitted to make unlimited and adaptive corruption queries. By querying an index i, $\mathcal{A}$ can obtain the corresponding encryption key ek_i for any sender i it selects.

We emphasize that the $t-1$ threshold applies strictly to aggregators (enforced via $QDKeyGen$). Conversely, the adversary can make unlimited adaptive corruption queries against clients (via $QCorrupt$), as the abort conditions in the *Finalize* phase enforce that their challenge plaintexts must be identical, preventing trivial wins.

Finalize: Ultimately, $\mathcal{A}$ submits its guess b' for the challenge bit b. The procedure then concludes by outputting β, the outcome of the security game, as defined in the subsequent analysis.

The game's final output β is determined as follows. Let CS denote the set of corrupted sender indices and let HS be the set of honest (non-corrupted) senders. By default, the output is set to $\beta \leftarrow b'$, which is the adversary's guess.

However, if any of the following three abort conditions occurs, the output is instead set randomly: $\beta \xleftarrow{\$} \{0,1\}$.

1. An encryption query $QEncrypt(i, x_i^0, x_i^1, l)$ is made for a corrupted sender $i \in CS$, but the challenge plaintexts were different (i.e., $x_i^0 \neq x_i^1$).
2. For label l, encryption query is issued for at least one honest sender $i \in HS$, but not all honest senders $j \in HS$ were queried for that same label l.
3. There exists a label l and a function f (for which dk_f is queried via $QDKeyGen$) that distinguishes the two challenge vectors $x^0 = (x_i^0)_i$ and $x^1 = (x_i^1)_i$. That is, $f(x^0) \neq f(x^1)$, where these vectors are constructed from $QEncrypt$ queries associated with label l as follows:
   - For all $i \in CS$ (corrupted senders), the inputs are identical: $x_i^0 = x_i^1$.
   - For all $i \in HS$ (honest senders), the values x_i^0 and x_i^1 are taken from the corresponding $QEncrypt(i, x_i^0, x_i^1, l)$ queries for that label.

The scheme is considered secure for encrypted data if, for any adversary $\mathcal{A}$, its advantage, defined as

$$Adv_{\mathcal{A}}^{IND} = |P[\beta = 1 \mid b = 1] - P[\beta = 1 \mid b = 0]|,$$

is negligible.

4.3. Construction

In this section, we present a detailed description of our partial decryption verifiable threshold MCFE scheme, which consists of six algorithms: Setup, DKeyGen, Encrypt, ShareDecrypt, Verify, and CombineRecover.

Setup (λ, t, s, n): Takes as input λ, threshold t, the number of partial decryption entities s, and the number of encryption entities n, and generates $\mathcal{G} = (G, p, g, h)$ using $\mathcal{BG}(1^\lambda)$. Selects hash functions $\mathcal{H}_1 : \{0,1\}^* \to G^2$, $\mathcal{H}_2 : \{0,1\}^* \to \mathbb{Z}_p$. Generates keys $\boldsymbol{s_i} \xleftarrow{\$} \mathbb{Z}_p^2$, $i \in \{1, \ldots, n\}$. Randomly chooses $\{c_{i,k}\}_{i\in\{1,\ldots,t-2\}} \xleftarrow{\$} \mathbb{Z}_p$ and computes $\{H_{i,k}\}_{i\in\{2,\ldots,t-1\}}$, where $H_{i,k} = h^{c_{i-1,k}}$. Defines public parameters pp as $(G, p, g, h, \mathcal{H}_1, \mathcal{H}_2, t, s, n, \{H_{i,k}\}_{i\in\{2,\ldots,t-1\}})$, encryption keys $ek_i = \boldsymbol{s_i}$ for $i \in \{1, \ldots, n\}$, and $msk = \{ek_i\}_{i\in[n]}$.

DKeyGen $(pp, msk, \boldsymbol{y}, k)$: Takes as input msk and a vector $\boldsymbol{y}$ defining an inner product function $f_y(x) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$, and computes function decryption keys $d = \sum_{i=1}^{n} \boldsymbol{s_i} \cdot y_i = (d_1, d_2) \in \mathbb{Z}_p^2$. Then constructs the following HLR:

$$\begin{cases} w_0 = d_1, w_1 = d_2, w_2 = c_{1,k}, \ \ldots, \ w_{t-1} = c_{t-2,k}; \\ w_{i+t} \ + \ a_1 \, w_{i+t-1} \ + \ \cdots + \ a_t \, w_i = 0 \pmod p, \quad i \geq 0 \end{cases}$$

where the auxiliary equation satisfies $(x - \alpha)^t = x^t + \sum_{i=1}^{t} a_i \, x^{t-i} \pmod p$. Using the above recursion relation, computes w_{t+j-1} for $j \in \{1, \ldots, s\}$, $H_{0,k} = h^{d_1}$ and $H_{1,k} = h^{d_2}$. The algorithm outputs the decryption key $dk_j = w_{t+j-1}, j \in \{1, \ldots, s\}$ for every partial decryption entity and publishes $H_{0,k}, H_{1,k}$.

Encrypt (pp, ek_i, x_i, l): Takes as input pp, encryption keys ek_i, a message x_i, and a label l. This algorithm computes $(g^{u_{l,0}}, g^{u_{l,1}}) = \mathcal{H}_1(l) \in G^2$, then outputs the ciphertext as follows:

$$ct_i = g^{u_l^\top \boldsymbol{s_i}} \cdot g^{x_i} = g^{u_l^\top \boldsymbol{s_i} + x_i} \in G.$$

PaDecrypt $(pp, \{ct_i\}_{i\in\{1,\ldots,n\}}, \boldsymbol{y}, dk_j, S, k)$: Takes as input $\{ct_i\}_{i\in\{1,\ldots,n\}}$, pp, a function $\boldsymbol{y} = \{y_i\}_{i\in\{1,\ldots,n\}}$, and a set of partial decryption entities S. The partial decryption participant $d_j \in S$ computes the partial decryption result as follows:

$$ct'_j = (ct'_{j,0}, ct'_{j,1}, ct'_{j,2}),$$

where:

$$ct'_{j,0} = \prod_{i\in\{1,\ldots,n\}} ct_i^{y_i},$$

$$ct'_{j,1} = g^{u_{l,1}\cdot\frac{w_{t+j-1}}{\alpha^{t+j-1}}\cdot\lambda_{j,1}\cdot\alpha^0},$$

$$ct'_{j,2} = g^{u_{l,2}\cdot\frac{w_{t+j-1}}{\alpha^{t+j-1}}\cdot\lambda_{j,2}\cdot\alpha^1}.$$

Here, $\lambda_{j,1} = \prod_{j'\in s, j'\neq j} \frac{0-(j'+t-1)}{j-j'}$ $\lambda_{j,2} = \prod_{j'\in s, j'\neq j} \frac{1-(j'+t-1)}{j-j'}$ is the Lagrange basis polynomial. Then it computes the proof for $ct'_{j,1}$ and $ct'_{j,2}$ using the DLEQ protocol: let $h_{j,1} = g^{u_{l,1}\cdot\lambda_{j,1}\cdot\alpha^{1-t-j}}$, $h_{j,2} = g^{u_{l,2}\cdot\lambda_{j,2}\cdot\alpha^{2-t-j}}$

The proof is as follows:

$$\Pi : \mathrm{Pok}(w_{t+j-1}) : \quad H_{t+j-1} = h^{w_{t+j-1}} \wedge ct'_{j,1} = h_{j,1}^{w_{t+j-1}} \wedge ct'_{j,2} = h_{j,2}^{w_{t+j-1}}$$

The algorithm outputs the partial decryption result (ct'_j, Π_j).

Verify $(pp, \{ct'_j\}_{j\in\{1,\ldots,s'\}}, k)$: Takes as input t partial decryption results $\{ct'_j\}_{j\in\{1,\ldots,s'\}}$. For each $ct'_{j,0} = \prod_{i=1}^{n} g^{(u_l^\top s_i + x_i)\cdot y_i} \in \{ct'_j\}_{j\in\{1,\ldots,s'\}}$, the algorithm first verifies if they are equal. Then, the algorithm computes $\{H_{t+j-1}\}_{j\in\{1,\ldots,s'\}}$, where $H_{t+j-1} = H_{t+j-2}^{-a_1} \cdot \ldots \cdot H_{j-1}^{-a_t} = h^{-a_1 w_{t+j-2} - \cdots - a_t w_{j-1}} = h^{w_{t+j-1}}$ and using the recursion relation .Verifies the proof of $\{ct'_{j,1}, ct'_{j,2}\}_{j\in\{1,\ldots,s'\}}$ using non-interactive DLEQ protocol. If the verification passes, the input t partial decryption ciphertexts are a valid set of partial decryption ciphertexts.

CombineRecover $(pp, \{ct'_j\}_{j\in\{1,\ldots,s'\}}, l)$: The algorithm takes as input a set of valid partial decryption results $\{ct'_j\}_{j\in\{1,\ldots,s'\}}$, and outputs the recovery result:

$$[\beta] = \frac{ct'_{j,0}}{\left(\prod_{j\in\{1,\ldots,s'\}} ct'_{j,1}\right)\cdot\left(\prod_{j\in\{1,\ldots,s'\}} ct'_{j,2}\right)},$$

and ultimately solves the discrete logarithm problem to obtain the function value $\beta = \langle \boldsymbol{x}, \boldsymbol{y}\rangle$.

Correctness.

$$\begin{aligned}
[\beta] &= \frac{ct'_{j,0}}{\left(\prod_{j\in\{1,\ldots,s'\}} ct'_{j,1}\right)\cdot\left(\prod_{j\in\{1,\ldots,s'\}} ct'_{j,2}\right)} \\
&= \frac{g^{\sum_{i=1}^n \boldsymbol{u}_l^\top \boldsymbol{s}_i y_i + \sum_{i=1}^n x_i y_i}}{g^{u_{l,1}\cdot\alpha^0\cdot\sum_j \frac{w_{t+j-1}}{\alpha^{t+j-1}}\lambda_{j,1}} \cdot g^{u_{l,2}\cdot\alpha^1\cdot\sum_j \frac{w_{t+j-1}}{\alpha^{t+j-1}}\lambda_{j,2}}} \\
&= \frac{g^{\sum_{i=1}^n \boldsymbol{u}_l^\top \boldsymbol{s}_i y_i + \sum_{i=1}^n x_i y_i}}{g^{u_{l,1}\cdot\alpha^0\cdot p(0)} \cdot g^{u_{l,2}\cdot\alpha^1\cdot p(1)}} \\
&= \frac{g^{\sum_{i=1}^n \boldsymbol{u}_l^\top \boldsymbol{s}_i y_i + \sum_{i=1}^n x_i y_i}}{g^{u_{l,1}\cdot w_0} \cdot g^{u_{l,2}\cdot w_1}} \\
&= \frac{g^{\sum_{i=1}^n \boldsymbol{u}_l^\top \boldsymbol{s}_i y_i + \sum_{i=1}^n x_i y_i}}{g^{\boldsymbol{u}_l^\top \boldsymbol{d}}} \\
&= \frac{g^{\sum_{i=1}^n \boldsymbol{u}_l^\top \boldsymbol{s}_i y_i + \sum_{i=1}^n x_i y_i}}{g^{\boldsymbol{u}_l^\top \sum_{i=1}^n y_i \boldsymbol{s}_i}} = g^{\sum_{i=1}^n x_i y_i}.
\end{aligned}$$

4.4. Security Analysis

In order to prove the security of our scheme, we designed a series of hybrid games. Under the DDH assumption, the real security game is gradually transformed into an ideal game in which the opponent's advantage can be ignored.

Theorem 1 (IND-Security). *We assert that the presented MCFE protocol achieves IND-security within the random oracle model, contingent upon the DDH assumption. Specifically, for a PPT adversary $\mathcal{A}$, the advantage is bounded as follows:*

$$\mathsf{Adv}^{IND}_{\mathcal{G},\mathcal{A}} \leq 2M \cdot \mathsf{Adv}^{ddh}_{\mathcal{G}}(t) + \mathsf{Adv}^{ddh}_{\mathcal{G}}(t + 4M \times t_{\mathbb{G}}) + \frac{2M}{p},$$

where M represents the queries number made to the random oracle $\mathcal{H}$, and $t_{\mathbb{G}}$ denotes the computational cost of a single exponentiation in the group $\mathbb{G}$.

Proof. As shown in Fig. 1, our proof is carried out through mixed games. We suppose that $\mathcal{A}$ is a probability polynomial time (PPT) adversary. For each game G_{index} in the sequence, we define the adversary's advantage as $\mathsf{Adv}_{\text{index}} = \left|\Pr[G_{\text{index}}(\mathcal{A})|b=1] - \Pr[G_{\text{index}}(\mathcal{A})|b=0]\right|$, with the probability space defined by the random coins of both the game G_{index} and the adversary $\mathcal{A}$. Furthermore, we use the notation $G_{\text{index}}(\mathcal{A})$ (or simply G_{index} when the context is clear) to represent the event where the Finalize procedure (as per Definition 5) outputs $\beta = 1$, based on the adversary's final guess b' during the interaction.

Game G_0 This initial game, G_0, corresponds to IND-security experiment outlined in Definition 2. In this game, hash function $\mathcal{H}$ is defined as a random oracle (RO), and its output is mapped to $\mathbb{G}^2$. Its primary role is to generate $g^{\boldsymbol{u}_\ell}$ by computing $\mathcal{H}(\ell)$.

Game G_1 In G_1, we modify the simulation. Answers to all new queries to the random oracle are now generated as uniformly random pairs from $\mathbb{G}^2$. This change is purely conceptual, as the oracle was already random; thus, the simulation is perfect. Consequently, the adversary's advantage remains unchanged: $\mathsf{Adv}_0 = \mathsf{Adv}_1$.

Game G_2 We introduce another modification. RO-queries are now answered with a random pair drawn from the span of $g^{\boldsymbol{a}}$, where $\boldsymbol{a}$ is defined as $\binom{1}{a}$ for a randomly chosen $a \leftarrow \mathbb{Z}_p$. This transition depends on the Multi-DDH problem. The difference in advantage is bounded by: $\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq \mathsf{Adv}^{ddh}_{\mathcal{G}}(t + 4M \times t_{\mathbb{G}})$, where M represents again RO-queries $t_{\mathbb{G}}$ is the cost of an exponential operation.

Game G_3 We alter QEncrypt simulation. All encryption queries are now processed using the message x_i^0, regardless of the challenge b. Concurrently, we revert simulation of the RO queries back to G_1's method, answering them with uniformly random pairs from $\mathbb{G}^2$.

It is evident that in this final game, the adversary's advantage becomes exactly 0, as the challenge bit b is

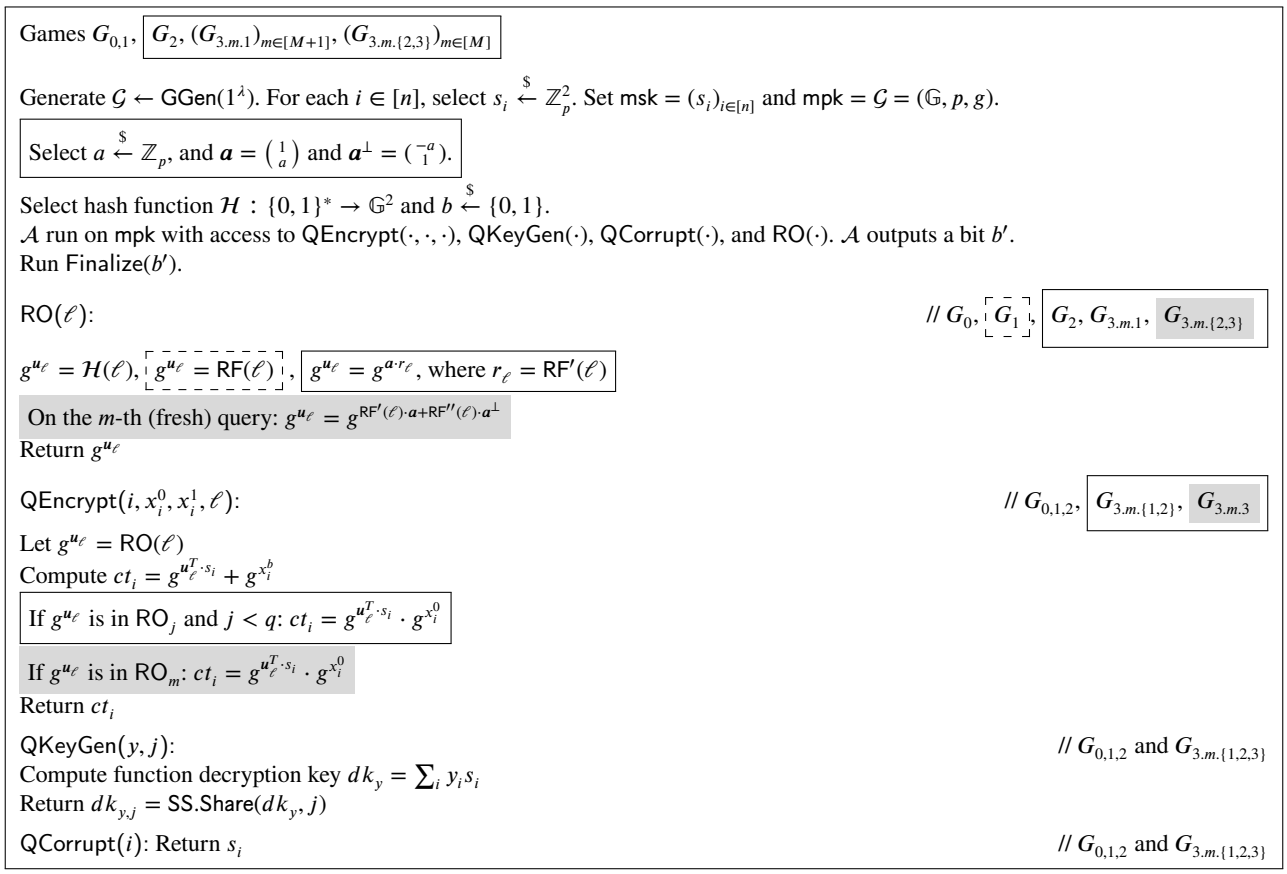


Fig. 1: Games $G_{0,1,2}$ and $G_{3.m.\{1,2,3\}}$ for the proof.

completely absent from the simulation. We use a hybrid argument applied sequentially to the RO-queries to trans from G_2 to G_3. We introduce a series of games indexed by m, for $m = 1, \ldots, M$. Note that this counts only the distinct queries to the RO, as any repeated query receives its original answer. We provide this proof in detail due to the importance of the technique.

$G_{3.1.1}$: This game is defined to be identical to G_2. Consequently, their advantages are equal: $\mathsf{Adv}_{G_2} = \mathsf{Adv}_{G_{3.1.1}}$.

$G_{3.m.1} \rightsquigarrow G_{3.m.2}$: For the m-th RO-query, we first alter the distribution of its response. Leveraging the DDH assumption, the output distribution is modified, transitioning from a uniform distribution over the subspace spanned by $g^{\boldsymbol{a}}$ to a uniform distribution across the entire group $\mathbb{G}^2$.

To analyze this change, we express a uniformly random vector from $\mathbb{Z}_p^2$ using the basis $\left(\binom{1}{a}, \binom{-a}{1}\right)$. A vector can thus be written as $u_1 \cdot \boldsymbol{a} + u_2 \cdot \boldsymbol{a}^\perp$, where both u_1, u_2 are randomly sampled from $\mathbb{Z}_p$.

Finally, we transition to a slightly different distribution, $u_1 \cdot \boldsymbol{a} + u_2 \cdot \boldsymbol{a}^\perp$, where u_1 is still drawn from $\mathbb{Z}_p$, but u_2 is drawn from $\mathbb{Z}_p^*$ (i.e., $u_2 \neq 0$). The maximum impact of this final change on the adversary's view is a distance of $1/p$ in static. This leads to the bound: $\mathsf{Adv}_{3.m.1} - \mathsf{Adv}_{3.m.2} \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{G}}(t) + 1/p$. The requirement $u_2 \in \mathbb{Z}_p^*$ is a crucial detail needed to ensure that $\boldsymbol{u}_\ell^\top \cdot \boldsymbol{a}^\perp \neq 0$.

$G_{3.m.2} \rightsquigarrow G_{3.m.3}$: The ciphertext generation process is modified from $ct_i = g^{\boldsymbol{u}_\ell{}^\top\cdot s_i} \cdot g^{x_i^b}$ to $ct_i = g^{\boldsymbol{u}_\ell{}^\top\cdot s_i} \cdot g^{x_i^0}$, where $g^{\boldsymbol{u}_\ell}$ represents the m-th RO-query. It will now be demonstrated that this modification is imperceptible to the adversary.

When there is none QEncrypt-queries use RO_m, games $G_{3.m.\{2,3\}}$ are trivially identical. But it shows that $\mathsf{Adv}_{3.m.2} = \mathsf{Adv}_{3.m.3}$ holds even in the case where this RO-query output is indeed utilized by QEncrypt-queries. We establish this assertion using a two-step argument.

Step 1. First, we demonstrate a PPT adversary $\mathcal{B}^*$ that plays against selective variants of the games, $G^*_{3.m.2}$ and $G^*_{3.m.3}$ (defined in Fig. 2). In these selective games, QCorrupt queries must be issued before the initialization phase. We show that the advantages are related as follows for $t = 2, 3$:

$$\mathsf{Adv}_{3.m.t} = (p^2+1)^n \cdot \mathsf{Adv}^*_{3.m.t}(\mathcal{B}^*).$$

Step 2. Second, we prove that for any PPT adversary $\mathcal{B}^*$, its advantages in these selective games are equal: $\mathsf{Adv}^*_{3.m.2}(\mathcal{B}^*) = \mathsf{Adv}^*_{3.m.3}(\mathcal{B}^*)$. The completion of these two steps concludes the proof.

Details of Step 1. We construct the adversary $\mathcal{B}^*$ to play against $G^*_{3.m.t}$ ($t = 2, 3$) such that the aforementioned advantage relationship holds.

$\mathcal{B}^*$ initiates by guessing, for all $i \in [n]$, a value $z_i \leftarrow \mathbb{Z}_p^2 \cup \{\perp\}$. It sends this complete set of guesses to its

Games $(G^*_{3.m.2}, G^*_{3.m.3})_{m\in[M]}$:

$(\mathsf{state}, (z_i \in \mathbb{Z}_p^2 \cup \{\perp\})_{i\in[n]}) \leftarrow \mathcal{A}(1^\lambda, 1^n)$

Generate $\mathcal{G} \leftarrow \mathsf{GGen}(1^\lambda)$. For each $i \in [n]$, sample $s_i \xleftarrow{\$} \mathbb{Z}_p^2$. Set $\mathsf{msk} = (s_i)_{i\in[n]}$ and $\mathsf{mpk} = \mathcal{G} = (\mathbb{G}, p, g)$.

Sample $a \xleftarrow{\$} \mathbb{Z}_p$, and define $\boldsymbol{a} = \binom{1}{a}$ and $\boldsymbol{a}^\perp = \binom{-a}{1}$.

Sample a bit $b \xleftarrow{\$} \{0,1\}$

The adversary $\mathcal{A}$ is run on mpk with access to oracles $\mathsf{QEncrypt}(\cdot,\cdot,\cdot)$, $\mathsf{QKeyGen}(\cdot)$, $\mathsf{QCorrupt}(\cdot)$, and $\mathsf{RO}(\cdot)$. $\mathcal{A}$ outputs a bit b'.

Run $\mathsf{Finalize}(b')$.

$\mathsf{RO}(\ell)$: // $G^*_{3.m.\{2,3\}}$

$g^{\boldsymbol{u}_\ell} = g^{\boldsymbol{a}\cdot r_\ell}$, where $r_\ell = \mathsf{RF}'(\ell)$

On the m-th (fresh) query: $g^{\boldsymbol{u}_\ell} = g^{\mathsf{RF}'(\ell)\cdot\boldsymbol{a} + \mathsf{RF}''(\ell)\cdot\boldsymbol{a}^\perp}$

Return $g^{\boldsymbol{u}_\ell}$

$\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$: // $\boxed{G^*_{3.m.2}}$, $G^*_{3.m.3}$

Let $g^{\boldsymbol{u}_\ell} = \mathsf{RO}(\ell)$

Compute $ct_i = g^{\boldsymbol{u}_\ell{}^\top \cdot s_i} \cdot g^{x_i^b}$

If $g^{\boldsymbol{u}_\ell}$ is in RO_j and $j < q$: $ct_i = g^{\boldsymbol{u}_\ell{}^\top \cdot s_i} \cdot g^{x_i^b}$

If $g^{\boldsymbol{u}_\ell}$ is in RO_m:

- If $(x_i^0, x_i^1) \neq z_i$, the game ends and returns $\beta \xleftarrow{\$} \{0,1\}$
- Otherwise, set $ct_i = g^{\boldsymbol{u}_\ell{}^\top \cdot s_i} \boxed{\cdot g^{x_i^b}} \, \cdot g^{x_i^0}$, $S = S \cup \{i\}$

Return ct_i

$\mathsf{QKeyGen}(y, j)$: // $G^*_{3.m.\{2,3\}}$

Compute function decryption key $dk_y = \sum_i y_i s_i$

Return $dk_{y,j} = \mathsf{SS.Share}(dk_y, j)$

$\mathsf{QCorrupt}(i)$: // $G^*_{3.m.\{2,3\}}$

If $z_i = (x_i^0, x_i^1)$ with $x_i^0 \neq x_i^1$, the game ends and returns $\beta \xleftarrow{\$} \{0,1\}$

Return s_i

Fig. 2: Games $G^*_{3.m.\{2,3\}}$, $m \in [M]$ for the proof.

selective game $G^*_{3,m,t}$. Each z_i represents $\mathcal{B}^*$'s guess for the $\mathsf{QEncrypt}$ query: either a pair (x_i^0, x_i^1) or $\perp$ (signifying no query for index i). $\mathcal{B}^*$ then simulates the view for the original adversary $\mathcal{A}$ using its own oracles.

Let E be the event that all of $\mathcal{B}^*$'s guesses are correct. If E occurs, $\mathcal{B}^*$ perfectly simulates $\mathcal{A}$'s view as in $G_{3.m.t}$. If E does not occur (the guess was incorrect), $\mathcal{B}^*$ aborts the simulation and outputs a random bit β.

The probability of this success event E is $P_E = (p^2 + 1)^{-n}$, and E is independent of $\mathcal{A}$'s view. We define $P(b, E) = \Pr[G_{3.m.t}|b = b, E]$. The advantage of $\mathcal{B}^*$ in its selective game, $\mathsf{Adv}_{G^*_{3.m.t}}(\mathcal{B}^*)$, is calculated as:

$$
\begin{aligned}
&\left| \left(P(0,E)\cdot P_E + \frac{1-P_E}{2} \right) \right. \\
&\qquad \left. - \left(P(1,E)\cdot P_E + \frac{1-P_E}{2} \right) \right| \\
&= \left| P(0,E)\cdot P_E - P(1,E)\cdot P_E \right| \\
&= P_E \cdot |P(0,E) - P(1,E)| \\
&= P_E \cdot \mathsf{Adv}_{3.m.t} \\
&= (p^2+1)^{-n} \cdot \mathsf{Adv}_{3.m.t}.
\end{aligned}
$$

Rearranging this equation gives the relation claimed in Step 1.

Details of Step 2. Conditioned on event E' (consistent guesses) occurring, we will now demonstrate that the distributions of games $G^*_{3.m.2}$ and $G^*_{3.m.3}$ are identical. This proof relies on the statistical indistinguishability of the following two distributions, which holds for any choice of γ:

$$
(s_i)_{\substack{i\in[n] \\ z_i=(x_i^0,x_i^1)}} \quad \text{and} \quad \left(s_i + \boldsymbol{a}^\perp \cdot \gamma(x_i^b - x_i^0)\right)_{\substack{i\in[n] \\ z_i=(x_i^0,x_i^1)}}
$$

Here, $\boldsymbol{a}^\perp$ is defined as $\binom{-a}{1} \in \mathbb{Z}_p^2$, and each s_i is drawn uniformly at random from $\mathbb{Z}_p^2$. The identical nature of the distributions holds because the s_i values are independent of the z_i guesses. It is crucial to note that this independence is a consequence of the selective security setting; it would not be guaranteed with adaptive $\mathsf{QEncrypt}$-queries. Consequently, we can substitute s_i with $s_i + \boldsymbol{a}^\perp \cdot \gamma(x_i^b - x_i^0)$ without altering the game's overall distribution.

We trace where the term $\boldsymbol{a}^\perp \cdot \gamma(x_i^b - x_i^0)$ impacts the adversary's view:

- $\mathsf{QCorrupt}$: The term has no effect. We are conditioned on E', which guarantees that for any i with $z_i \neq \perp$, i is not corrupted or $x_i^1 = x_i^0$.

- QKeyGen(y): The term could potentially modify the key to:

$$\mathsf{dk}_y = \sum_{i\in[n]} s_i \cdot y_i + \boldsymbol{a}^{\perp} \cdot \gamma \sum_{i\in[n], z_i=(x_i^0,x_i^1)} y_i(x_i^b - x_i^0).$$

This additional component vanishes. The conditions of E' ensure the summation $\sum_{i:z_i=(x_i^0,x_i^1)} y_i(x_i^b - x_i^0)$ is always zero, as $z_i \neq \perp$ implies $x_i^1 = x_i^0$ for all relevant indices i.

- QEncrypt: The term appears only in QEncrypt-queries using the m-th RO-query, $g^{\boldsymbol{u}_\ell}$. For all other RO-queries, $g^{\boldsymbol{u}_\ell}$ is in the span of $[a]$, and $\boldsymbol{a}^\top \boldsymbol{a}^{\perp} = 0$. For the m-th query, the ciphertext becomes:

$$ct_i = g^{\boldsymbol{u}_\ell^\top \cdot s_i} \cdot g^{\boldsymbol{u}_\ell^\top \cdot \boldsymbol{a}^{\perp} \gamma (x_i^b - x_i^0)} \cdot g^{x_i^b}.$$

Since $\boldsymbol{u}_\ell^\top \boldsymbol{a}^{\perp} \neq 0$ (by setup), we set $\gamma \leftarrow -1/(\boldsymbol{u}_\ell^\top \boldsymbol{a}^{\perp}) \bmod p$. This constant γ is independent of i and transforms the ciphertext to $ct_i = g^{\boldsymbol{u}_\ell^\top \cdot s_i} \cdot g^{x_i^0}$, simultaneously changing all x_i^b encryptions to x_i^0.

By reversing this statistically perfect substitution, we confirm that the original ct_i (in $G^*_{3.m.2}$) is identically distributed to $g^{\boldsymbol{u}_\ell^\top \cdot s_i} \cdot g^{x_i^0}$, which is precisely the form of the ciphertext in game $G^*_{3.m.3}$.

The games are identical conditioned on E' (consistent guesses). If $\neg E$ (incorrect guess) occurs, $\mathcal{B}^*$ aborts with a random bit in both games. Therefore, $\mathsf{Adv}^*_{3.m.2}(\mathcal{B}^*) = \mathsf{Adv}^*_{3.m.3}(\mathcal{B}^*)$, implying $\mathsf{Adv}_{3.m.2} = \mathsf{Adv}_{3.m.3}$.

$G_{3.m.3} \rightsquigarrow G_{3.m+1.1}$: This step reverses the $G_{3.m.1} \rightsquigarrow G_{3.m.2}$ transition. By the DDH assumption, we revert the m-th RO-query $g^{\boldsymbol{u}_\ell}$ from uniformly random in $\mathbb{G}^2$ (where $\boldsymbol{u}_\ell^\top \boldsymbol{a}^{\perp} \neq 0$) to uniformly random in the span of $\boldsymbol{a}$.

$$\mathsf{Adv}_{3.m.3} - \mathsf{Adv}_{3.m+1.1} \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{G}}(t) + 1/p.$$

In summary, by iterating this process through all M RO-queries, we find that $G_{3.M+1.1}$ is equivalent to G_3. This yields the bound:

$$\mathsf{Adv}_2 - \mathsf{Adv}_3 \leq 2M(\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{G}}(t) + 1/p).$$

Furthermore, since the advantage in game G_3 is zero ($\mathsf{Adv}_3 = 0$), the proof is complete.

5. Federated Learning Secure Aggregation

In this section, we present VTSAFL, a new verifiable threshold secure aggregation protocol for privacy-preserving federated learning, which is based on our novel verifiable threshold MCFE scheme. We will detail the system framework, which consists of clients, distributed aggregators, and a trusted authority (TA), as well as the distributed training process. A security analysis will follow, examining privacy risks and attacks from malicious aggregators.

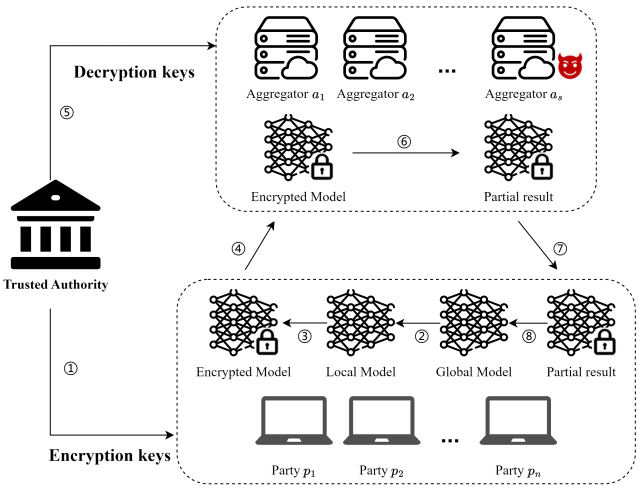


Fig. 3: Our Privacy-Preserving Federated Learning Framework

5.1. System Architecture

The VTSAFL framework, illustrated in Fig. 3, consists of three primary entities. A trusted authority (TA) initializes the cryptosystem and distributes keys. A set of clients uses local data to train and subsequently encrypt their models. Finally, a group of distributed aggregators collects these encrypted models, decrypts them, and computes an accompanying proof of decryption. This allows all clients to verify the correctness of the aggregation result.

It is important to note that, unlike previous solutions based on functional encryption, we assume a more robust threat model wherein aggregators can be malicious. These aggregators may collude to breach user privacy or alter results to disrupt training, an assumption that aligns with the TAPFed framework [17].

By virtue of the system's threshold nature, participation from the entire set of aggregators is not necessary in each round. The aggregators do not communicate with or even need to be aware of each other. This feature significantly enhances both the stability and the operational efficiency of the system.

5.2. Training Process

The distributed training process of the VTSAFL framework is based on the traditional federated learning workflow and is primarily divided into five stages: initialization, local model training and encryption, aggregation and partial decryption by aggregators, verification of decrypted results by clients, and finally, global model updates. The entire process is detailed in Algorithm 1.

Initialization: The Trusted Authority (TA) runs the VTMCFE.Setup algorithm to generate public parameters pp and a master secret key msk. The public parameters are broadcast to all participants, while the TA securely sends the respective encryption keys ek_i to each client p_i.

Local Model Training and Encryption: Each client p_i first initializes its local model. Then, for each training round, it trains its local model using its own private dataset D_i.

Algorithm 1: Federated Learning Framework

Input : The security parameter λ, party set S_P and each party $p_i \in S_P$ has its dataset D_i, aggregator set S_A, trusted authority T, maximum training round K, hyperparameters hp, encryption label for each training round $\{l^{(k)}\}_{k\in\{1,...,K\}}$

Output: Global model θ_G

Initialize:
T runs $(pp, msk) \leftarrow \mathsf{VTMCFE.Setup}(\lambda, t, s, n)$ and broadcasts pp;
T sends ek_i to p_i for $i \in \{1, ..., n\}$;
Training:
do
  Local model training:
  if $k == 1$ **then** p_i initializes local model $\theta_{p_i}^{(0)}$;
  $\theta_{p_i}^{(k)} \leftarrow p_i$ trains model with $(D_i, \theta_{p_i}^{(0)}, hp)$;
  p_i runs $ct_{p_i}^{(k)} \leftarrow \mathsf{VTMCFE.Encrypt}(ek_i, \theta_{p_i}^{(k)}, l^{(k)})$;
  p_i sends $ct_{p_i}^{(k)}$ to $a_j \in S_A$;
  FedAvg model aggreation:
  a_j obtains all $ct_{p_i}^{(k)}$ from $p_i \in S_P$;
  a_j prepares fusion weight y^{a_j} based on $\{ct_{p_i}^{(k)}\}$;
  a_j requests dk_j from T with (y^{a_j});
  T checks and validates all requests collected from $a_j \in S_A$;
  T chooses a valid request y and runs $\{dk_j\} \leftarrow \mathsf{VTMCFE.DKeyGen}(pp, msk, y)$;
  T sends dk_j to $a_j \in S_A$;
  a_j runs $(ct_j^{'(k)}, \Pi_j) \leftarrow \mathsf{VTMCFE.Decrypt}(pp, \{ct_{p_i}^{(k)}\}, y, dk_j)$;
  a_j sends $(ct_j^{'(k)}, \Pi_j)$ to $p_i \in S_P$;
  Global model updating:
  p_i runs $\mathsf{VTMCFE.Verify}(pp, \{ct'_j, \Pi_j\}_{j\in\{1,...,s'\}})$;
  if $\mathsf{VTMCFE.Verify}(pp, \{ct'_j, \Pi_j\}_{j\in\{1,...,s'\}}) == 1$ **then**
    p_i runs $\theta_A^{(k)} \leftarrow \mathsf{VTMCFE.CombineRecovery}(pp, \{ct'_j\}_{j\in\{1,...,s'\}}, l)$;
    p_i updates $\theta_G^{(k)} \leftarrow \theta_A^{(k)}$;
while $k \leq K$;

After training, the client encrypts its updated local model $\theta_{p_i}^{(k)}$ using its encryption key and the current round's label $l^{(k)}$, producing a ciphertext $ct_{p_i}^{(k)}$. This encrypted model is then sent to the aggregators.

Aggregation and Partial Decryption: After receiving the encrypted models from all clients, each aggregator a_j prepares a fusion weight vector y and requests the corresponding decryption key share from the TA. The TA validates these requests, generates the decryption key shares $\{dk_j\}$ using VTMCFE.DKeyGen, and distributes them to the respective aggregators. Each aggregator a_j then uses its key share dk_j to compute a partial decryption result of the aggregated model and a corresponding proof of correctness $(ct_j^{(k)}, \Pi_j)$ by running the VTMCFE.Decrypt algorithm. This partial result and proof are then sent back to the clients.

Verification: Upon receiving the partial results and proofs from a threshold number of aggregators, each client p_i runs the VTMCFE.Verify algorithm. This step allows clients to check the validity of the computation performed by each aggregator, thus protecting against malicious aggregators who might return incorrect results to disrupt the training process.

Global Model Update: If the verification is successful, the clients proceed to the final step. They run the VTMCFE.CombineRecovery algorithm to combine the valid partial results and recover the final aggregated global model $\theta_G^{(k)}$. This new global model is then used as the starting point for the next round of local training. This iterative process continues for a predetermined number of rounds K.

5.3. Practical considerations

To bridge the gap between the VTMCFE scheme and the actual requirements of federated learning tasks, it is necessary to clarify some implementation details regarding data encoding, range management, and cryptographic optimization.

Model weight quantization and encoding: The underlying VTMCFE scheme operates on elements of a finite field $\mathbb{Z}_p$, while neural network weights (θ) are typically represented as 32-bit floating-point numbers. To ensure compatibility, we employ a quantization mechanism. Given a scaling factor 10^k (e.g., $k = 4$) and precision requirement, each weight $w \in \mathbb{R}$ is converted to an integer $\overline{w} \in \mathbb{Z}_p$.

Efficient discrete logarithm solving: In the VTMCFE scheme, the CombineRecovery algorithm generates the result at an exponential position, i.e., $G = g^{\beta}$. Therefore, recovering β requires solving the discrete logarithm problem (DLP). While DLP is generally difficult to solve, the aggregated weights in federated learning are restricted to a relatively small range B compared to p. For typical model updates, the scaled range of β is limited to $[-10^{12}, 10^{12}]$. To solve the discrete logarithm problem, we implemented the Baby-Step Giant-Step (BSGS) algorithm, which reduces the time complexity to $O(\sqrt{B})$. To further optimize the training process, we leveraged the fact that model updates between consecutive rounds are usually small, allowing us to use the result from the previous round as a hint for the search starting point.

5.4. Security Analysis of VTSAFL

In this section, we analyze the security properties of the VTSAFL framework. Our threat model assumes a malicious setting where up to $t - 1$ out of s aggregators may collude.

Table 3
Experimental Setting

Dataset	Samples(train|test)/Party	Model Parameters	Training Rounds	Local Epochs	Local Batch
MNIST	1,000|200	110,170	20	10	50
CIFAR10	10,000|2,000	129,242	20	5	50

These adversarial aggregators may deviate from the protocol to compromise integrity or attempt to infer sensitive information.

Privacy Guarantees (Defense against Inference Attacks). VTSAFL ensures the confidentiality of local gradients and prevents unauthorized access to the global model updates through two layers:

- *Individual Privacy:* In VTSAFL, clients encrypt their local models $\theta_{p_i}^{(k)}$ using the VTMCFE.Encrypt algorithm. Due to the semantic security of the underlying Multi-Client Functional Encryption (MCFE) primitive, no individual aggregator can inspect or derive any sensitive information from a client's individual ciphertext $ct_{p_i}^{(k)}$.
- *Intermediate Model Privacy:* To prevent unauthorized decryption of the aggregated result, we employ a (t, s)-threshold architecture. The functional decryption key is distributed such that any collusion of up to $t-1$ adversarial aggregators is mathematically insufficient to reconstruct the functional key or decrypt the intermediate model.

Integrity Guarantees (Defense against Poisoning Attacks). Unlike 'honest-but-curious' models, VTSAFL withstands active malicious behaviors during aggregation:

- *Aggregation Poisoning Defense:* Adversarial aggregators may return tampered results to disrupt training. VTSAFL counters this via *verifiability*. Each aggregator a_j must generate a cryptographic proof Π_j for its partial decryption ct'_j. Clients execute VTMCFE.Verify to validate these proofs; any incorrect result will be discarded. This ensures the correctness and integrity of the global model updates.
- *Anti-Replay Mechanism:* To prevent adversaries from replaying ciphertexts from previous rounds (e.g., using $ct^{(k-1)}$ in round k), each encryption is bound to a unique round label $l^{(k)}$. The MCFE primitive ensures that only ciphertexts with the same label can be combined, rendering outdated ciphertexts incompatible with the current round.

Resistance to Collusion. The security of our framework holds as long as the number of colluding malicious aggregators is less than the threshold t. The use of a verifiable multi-secret sharing scheme ensures that even if $t-1$ nodes share their internal states, they cannot bypass the functional encryption layer or forge a valid proof of correctness for a tampered result.

6. Performance Evaluation

In this section, we evaluate the proposed VTSAFL framework against the state-of-the-art TAPFed scheme [17], focusing on model quality, computational efficiency, and communication overhead. Experimental results demonstrate that VTSAFL matches TAPFed's model accuracy while significantly reducing both running time and communication costs. Furthermore, our framework provides the added security benefit of verifiable aggregation.

6.1. Experimental Setup

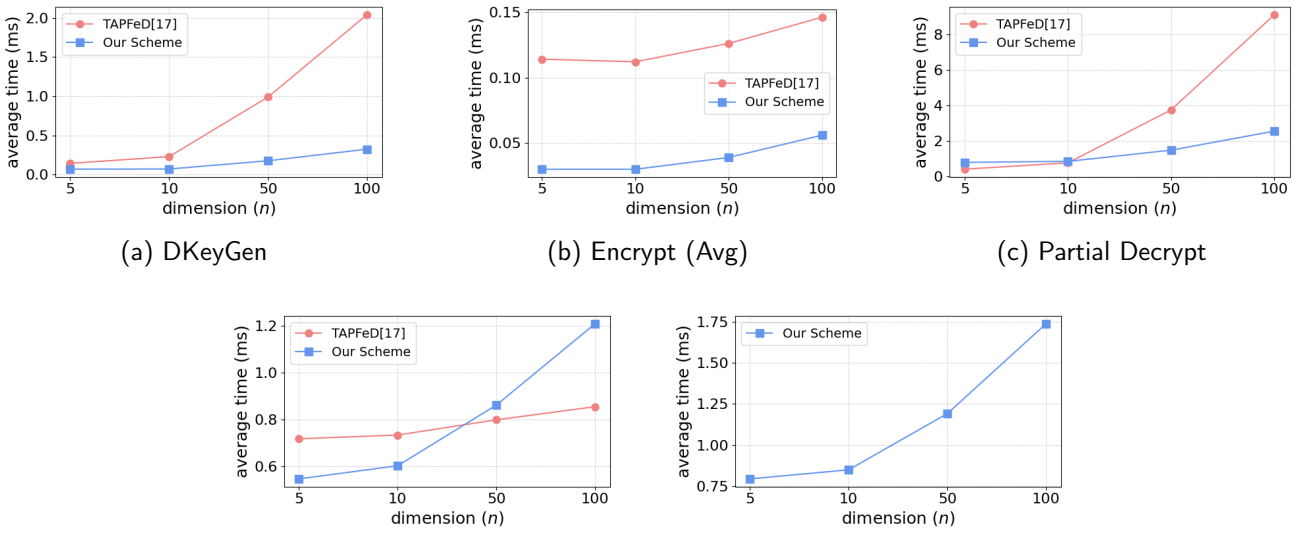
Implementation. We utilized the TensorFlow library for training the federated learning models and implement our framework in python. For a fair comparison, the core cryptographic components of the TAPFed scheme [17] were re-implemented precisely as described in the original publication. All experiments were performed on a local simulated system. The hardware configuration comprised an Intel(R) Core(TM) i5-13400F CPU, 32GB of DDR4 RAM, and an NVIDIA GeForce RTX 4070s GPU with 12GB of memory.

Federated Learning Setting. Model performance was evaluated using the public MNIST and CIFAR10 datasets. A Keras-based Convolutional Neural Network (CNN) architecture was employed. The model configured for MNIST contained 110,170 parameters, while the CIFAR10 model comprised 129,242 parameters. During each training round, clients first train models on their local datasets, after which the resulting model updates are encrypted and uploaded. A comparative analysis was conducted against the TAPFed scheme [17], focusing on model accuracy, training time, and communication overhead. This analysis involved varying both the number of participating clients and the dimensionality of the model vectors.

Our experimental evaluation is structured in two main phases. The initial phase focused on assessing model performance. Here, both our proposed scheme and TAPFed were trained for 20 rounds on the MNIST and CIFAR10 datasets to compare their test accuracies. Relevant hyperparameters and experimental settings are detailed in Table 3. The second phase was designed to evaluate the training time and communication overhead of both schemes as the number of clients varied. Experiments were conducted with the number of clients set to 5, 10, 20, 30, 40, and 50, respectively.

6.2. Experimental Results

Analysis of Cryptographic Primitives. To provide a more detailed view, we analyzed the performance of the core cryptographic primitive, as shown in the Fig. 4. Our scheme shows significant efficiency improvement in the DKeyGen



(a) DKeyGen (b) Encrypt (Avg) (c) Partial Decrypt

(d) Combine Decrypt (e) Verify

Fig. 4: Performance Comparison of Core Cryptographic Primitives

and Partial Decrypt stages, which is the main factor to reduce the overall training time. Although our Combine Decrypt is equivalent to TAPFed [17], our framework introduces a key Verify step. The time cost of this verification is reasonable, and it provides the necessary guarantee for the correctness of the aggregation results, which is the missing feature of TAPFed [17].

Model Quality and Performance. We first evaluate the model quality of VTSAFL relative to the baseline. The experimental results show that, compared with standard federated learning, the proposed security aggregation protocol will not adversely affect the key performance indicators, namely learning convergence speed, training loss and test accuracy. As shown in Fig. 5, during the whole 20 rounds of federal learning and training, the performance curve of test accuracy and training loss generated by our scheme is very close to that of TAPFed. This result is attributed to the design of VTSAFL, which uses pure password mechanism to protect the exchange of model parameters. By avoiding introducing noise, our method achieves the same model quality as other password based security aggregation baselines.

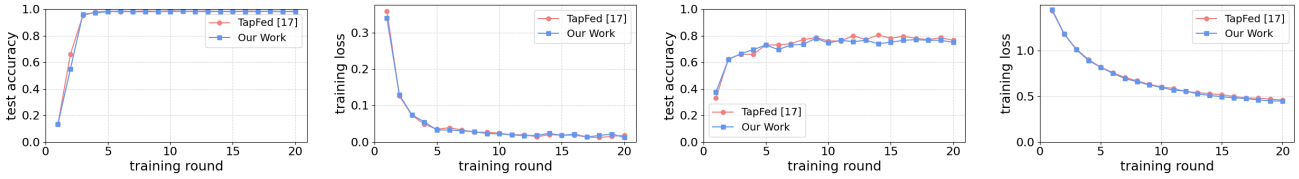
Training Time. We studied the effect of security aggregation method on training time and compared it with TAPFed. As shown in Fig. 6 (a) and Fig. 6 (c), the total running time of VTSAFL is significantly lower than that of TAPFed, and the performance gap will expand with the increase of the number of clients. In 50 client scenarios, our framework achieved more than 40% time reduction on both datasets. Specifically, the running time for MNIST is reduced by 40.6%, and for the more complex CIFAR10 dataset, the reduction is 44.4%. This highlights the superior time performance of our scheme in a practical, large-scale FL setting, which directly benefits from the highly efficient DKeyGen and Decrypt primitives analyzed previously.

Table 4
Comparison of Communication Cost per Round

	DKeyGen	Encrypt	ShareDecrypt
TAPFed[17]	$(n+1)\|\mathbb{Z}_p\|$	$2\|G\|$	$(n+2)\|G\|$
our work	$2\|G\|+\|\mathbb{Z}_p\|$	$\|G\|$	$5\|G\|+2\|\mathbb{Z}_p\|$

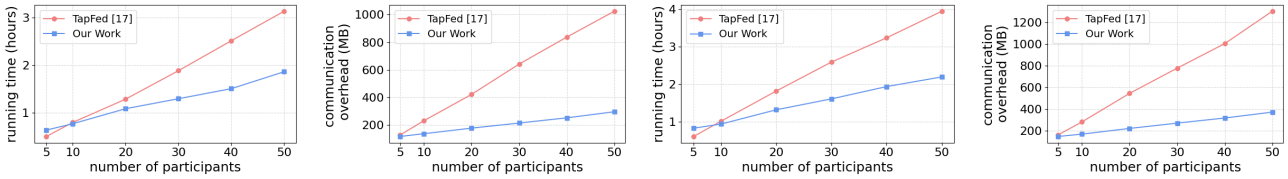
Communication Overhead. In order to evaluate the communication efficiency of our scheme, we measured the transmission overhead in each round of training through experimental comparison. As indicated in Table 4 and illustrated in Fig. 6 (b) and Fig. 6 (d), VTSAFL achieves a substantially lower transmission payload. For the client-side upload, our framework reduces the payload by 50.1% for MNIST and 50.8% for CIFAR10 in the 50-client scenario. While this client-upload cost scales linearly with n for both schemes, VTSAFL's scalability is far superior. Furthermore, this advantage is compounded by other protocol phases. As detailed in Table 4, the communication costs for key phases in TAPFed (i.e., DKeyGen and ShareDecrypt) are also proportional to n. In sharp contrast, the corresponding communication phases in VTSAFL (e.g., DS-to-Aggregator) have a constant cost that is independent of n.

This performance advantage underscores the superior scalability of our framework. As shown in Fig. 6, both the total running time and communication overhead of VTSAFL scale much more efficiently compared to TAPFed [17]. This confirms that our framework is exceptionally well-suited for large-scale federated learning tasks, offering a practical path to high efficiency and strong security at scale.



(a) (b) (c) (d)

Fig. 5: Comparison of test accuracy and training loss on MNIST (a, b) and CIFAR-10 (c, d).



(a) (b) (c) (d)

Fig. 6: Comparison of running time and communication overhead on MNIST (a, b) and CIFAR-10 (c, d).

7. Conclusion

In this paper, we proposed a new privacy preserving federated learning security aggregation framework VTSAFL, to address the security vulnerabilities of existing solutions in the face of malicious aggregators. Based on our partial decryption verifiable threshold multi client function encryption scheme, the framework of VTSAFL was constructed. It can not only resist the gradient reasoning and data leakage attacks in the existing schemes, but also enable each client to verify the correctness of the aggregation results. This effectively prevents poisoning attacks from malicious aggregators and ensures the integrity of model training.

At the same time, in order to apply it into the scenario of more constrained devices in the internet of things environment and avoid the limitation of the computing power of edge devices, we are committed to improving the computing efficiency and communication overhead of existing solutions. Experimental results show that VTSAFL has significant advantages in computational performance and communication efficiency compared with existing schemes, and achieves the same model performance. For internet of things scenarios with a large number of devices, the scheme has good scalability.

For future work, we will extend the application of VTSAFL to a broader cross device joint learning environment to verify its practicability and performance under different network and hardware conditions.

CRediT authorship contribution statement

Minjie Wang: Conceptualization, Methodology, Software, Investigation, Validation, Formal analysis, Writing - Original Draft, Writing - Review & Editing. **Jinguang Han**: Conceptualization, Writing - Review & Editing, Supervision, Funding acquisition. **Weizhi Meng**: Writing - Review & Editing.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grant 62372103, the Natural Science Foundation of Jiangsu Province under Grant BK20231149 and the Jiangsu Provincial Scientific Research Center of Applied Mathematics under Grant BK20233002.

References

[1] Brendan McMahan, Ewan Moore, Daniel Ramage, Seth Smathon, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proc. Int. Conf. Artif. Intell. Statist.*, volume 54, pages 1273–1282, 2017.

[2] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *Proc. IEEE Symp. Secure. Privacy*, pages 3–18, 2017.

[3] Karan Ganju, Qi Wang, Wei Yang, Carl A. Gunter, and Nikita Borisov. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proc. ACM SIGSAC Conf. Comput. Commun. Secure.*, pages 619–633, 2018.

[4] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *Proc. IEEE Symp. Secure. Privacy*, pages 739–753, 2019.

[5] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 14774–14784, 2019.

[6] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients—how easy is it to break privacy in federated learning? In *Proc. Adv. Neural Inf. Process. Syst.*, pages 16937–16947, 2020.

[7] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the GAN: Information leakage from collaborative deep learning. In *Proc. ACM SIGSAC Conf. Comput. Commun. Secure.*, pages 603–618, 2017.

[8] Chengliang Zhang, Sijia Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. BatchCrypt: Efficient homomorphic encryption for cross-silo federated learning. In *Proc. USENIX Annu. Tech. Conf.*, pages 493–506, 2020.

[9] Guowen Xu, Hongwei Li, Yuan Zhang, Shouhuai Xu, Jianting Ning, and Robert H. Deng. Privacy-preserving federated deep learning with irregular users. *IEEE Trans. Depend. Secure Comput.*, 19(2):1364–1381, 2022.

[10] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. HybridAlpha: An efficient approach for privacy-preserving federated learning. In *Proc. ACM Workshop Artif. Intell. Secure.*, pages 13–23, 2019.

[11] Yan Chang, Kejia Zhang, Junbin Gong, and Haifeng Qian. Privacy-preserving federated learning via functional encryption, revisited. *IEEE Trans. Inf. Forensics Secure.*, 18:1855–1869, 2023.

[12] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, Swanand Kadhe, and Heiko Ludwig. DeTrust-FL: Privacy-preserving federated learning in decentralized trust setting. In *Proc. IEEE Int. Conf. Cloud Comput.*, pages 417–426, 2022.

[13] Xiaoning Qian, Hongwei Li, Menghao Hao, Guowen Xu, Haomiao Wang, and Yuguang Fang. Decentralized multi-client functional encryption for inner product with applications to federated learning. *IEEE Trans. Depend. Secure Comput.*, 21(6):5781–5796, 2024.

[14] Mayank Rathee, Cong Shen, Sameer Wagh, and Raluca Ada Popa. ELSA: Secure aggregation for federated learning with malicious actors. In *Proc. IEEE Symp. Secure. Privacy*, pages 1961–1979, 2023.

[15] Pau-Chen Cheng, Kevin Eykholt, Zhongshu Gu, Hani Jamjoom, K. R. Jayaram, Enriquillo Valdez, and Abhishek Verma. Separation of powers in federated learning (poster paper). In *Proc. 1st Workshop Syst. Challenges Rel. Secure Federated Learn.*, pages 16–18, 2021.

[16] Yipeng Zhang, Zhibo Wang, Jianping Cao, Rui Hou, and Dan Meng. ShuffleFL: Gradient-preserving federated learning using trusted execution environment. In *Proc. ACM Int. Conf. Comput. Frontiers*, pages 161–168, 2021.

[17] Runhua Xu, Boyang Li, Chao Li, James B. D. Joshi, Shuo Ma, and Jianghe Li. TAPFed: Threshold secure aggregation for privacy-preserving federated learning. *IEEE Trans. Depend. Secure Comput.*, 21(5):4309–4323, 2024.

[18] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Proc. CRYPTO*, pages 89–105, 1993.

[19] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Proc. 8th Theory Cryptogr. Conf.*, pages 253–273, 2011.

[20] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Proc. IEEE Symp. Found. Comput. Sci.*, pages 40–49, 2013.

[21] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Proc. EUROCRYPT*, pages 578–602, 2014.

[22] Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In *Proc. CRYPTO*, pages 597–627, 2018.

[23] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In *Proc. ASIACRYPT*, pages 703–732, 2018.

[24] Ximeng Liu, Hongwei Li, Guowen Xu, Sencun nations Liu, Zhili Liu, and Rongxing Lu. PADL: Privacy-aware and asynchronous deep learning for IoT applications. *IEEE Internet Things J.*, 7(8):6955–6969, 2020.

[25] Nan Wu, Farhad Farokhi, David Smith, and Mohamed Ali Kaafar. The value of collaboration in convex machine learning with differential privacy. In *Proc. IEEE Symp. Secure. Privacy*, pages 304–317, 2020.

[26] Xiaofeng Guo, Zheli Liu, Jin Li, Junbin Gao, Bingzhe Hou, Changyu Dong, and Thar Baker. VeriFL: Communication-efficient and fast verifiable aggregation for federated learning. *IEEE Trans. Inf. Forensics Secure.*, 16:1736–1751, 2020.

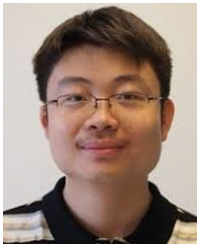
[27] Ning Li, Meng Zhou, Han Yu, Yenwei Chen, and Zhe Yang. SVFLC: Secure and verifiable federated learning with chain aggregation. *IEEE Internet Things J.*, 11(8):13125–13136, 2024.

[28] Gang Wang, Lei Zhou, Qi Li, Xiaojuan Yan, Ximeng Liu, and Yongdong Wu. FVFL: A flexible and verifiable privacy-preserving federated learning scheme. *IEEE Internet Things J.*, 11(13):23268–23281, 2024.

[29] Sedigheh Mashahdi, Behzad Bagherpour, and Abbas Zaghian. A non-interactive (t, n)-publicly verifiable multi-secret sharing scheme. *Des. Codes Cryptogr.*, 90(8):1761–1782, 2022.

Minjie Wang received the B.E. degree from School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China, in 2023. He is currently working toward the the master's degree with the School of Cyber Science and Engineering, Southeast University, Nanjing, China. His main research directions include functional encryption and privacy computing.

Jinguang Han received the PhD degree from the University of Wollongong, Australia, in 2013. He is a professor with the School of Cyber Science and Engineering, Southeast University, China. His research focuses on access control, cryptography, cloud computing, and privacy-preserving systems. He served as a Program Committee Co-Chair for ProvSec'2016, FCS'2019, and SPNCE'2020; and a ProgramCommitteeMemberforseveralconferences, including SecureCom'2023, ISC'2022, PST'2021, ESORICS'2020, and ICICS'2019.



Weizhi Meng(Senior Member, IEEE) received the Ph.D. degree in computer science from the City University of Hong Kong. He is currently a Full Professor with the School of Computing and Communications, Lancaster University, U.K., and an Adjunct Faculty Member with the Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark. His primary research interests include blockchain technology, cyber security, and artificial intelligence in security, including intrusion detection, blockchain applications, smartphone security, biometric authentication, and the IoT security. He was a recipient of Hong Kong Institution of Engineers (HKIE) Outstanding Paper Award for Young Engineers/Researchers in both 2014 and 2017. He also received the IEEE ComSoc Best Young Researcher Award for Europe, Middle East, and Africa Region (EMEA) in 2020, and the IEEE ComSoc Communications and Information Security (CISTC) Early Career Award in 2023.