

# Efficient Full-Stack Private Federated Deep Learning with Post-Quantum Security

Yiwei Zhang, Rouzbeh Behnia, Attila A. Yavuz, Reza Ebrahimi, Elisa Bertino

**Abstract**—Federated learning (FL) enables collaborative model training while preserving user data privacy by keeping data local. Despite these advantages, FL remains vulnerable to privacy attacks on user updates and model parameters during training and deployment. Secure aggregation protocols have been proposed to protect user updates by encrypting them, but these methods often incur high computational costs and are not resistant to quantum computers. Additionally, differential privacy (DP) has been used to mitigate privacy leakages, but existing methods focus on secure aggregation or DP, neglecting their potential synergies. To address these gaps, we introduce **Beskar**, a novel framework that provides post-quantum secure aggregation, optimizes computational overhead for FL settings, and defines a comprehensive threat model that accounts for a wide spectrum of adversaries. We also integrate DP into different stages of FL training to enhance privacy protection in diverse scenarios. Our framework provides a detailed analysis of the trade-offs between security, performance, and model accuracy, representing the first thorough examination of secure aggregation protocols combined with various DP approaches for post-quantum secure FL. **Beskar** aims to address the pressing privacy and security issues FL while ensuring quantum-safety and robust performance.

**Index Terms**—Privacy-preserving AI, post-quantum security, differential privacy, secure aggregation, deep learning.

## I. INTRODUCTION

Federated learning (FL) enables collaborative learning of a shared model between distributed parties while keeping the data local, mitigating data privacy and collection challenges common in traditional centralized learning. In large-scale FL, clients with limited computational resources, such as mobile devices, can contribute to training a global model with the assistance of a central server. In each iteration, the central server distributes an *intermediate model* to all clients, who then train the model using their local data to compute *local updates* (i.e., user gradients). The server aggregates the local updates from all users, refines the intermediate model, and sends it back to the clients. This iterative process continues until the model achieves a satisfactory level of performance, resulting in a *final model* to be deployed in production. FL contributes to data privacy by keeping the user data local. However, recent attacks have demonstrated that deploying a plain FL paradigm is insufficient to protect the privacy of the participating users' data [1], [2], [3]. More specifically,

these attacks can undermine the privacy of the training data by having access only to the user updates or the model at any stage (training and/or deployment).

A well-known solution to protect the user updates during the training phase is secure aggregation [4], where the server can compute the global model without knowledge of any individual user update. This is achieved by masking/encrypting the updates so that the masking factors cancel out during aggregation. Secure aggregation can be achieved with different techniques, such as secure distributed computation [5], [6], [7] or Homomorphic Encryption (HE) [8], [9]. However, existing secure aggregation protocols often incur high communication and/or computation overhead.

Additionally, to our knowledge, except for a few (e.g., [10]), the existing methods are primarily based on conventional cryptographic tools. However, such tools are not resistant to quantum computers, which are on the verge of becoming a reality. Therefore a pressing requirement is that FL protocols (and other distributed protocols for machine learning) must be post-quantum secure. Given the directives by the NSA and the White House [11], [12], [13], NIST has suggested a series of post-quantum (PQ) secure cryptographic schemes. While one might consider the direct adoption of the recent NIST PQ cryptographic standards [14], these schemes and their subsequent extensions (e.g., [15], [16]), despite their elegant designs, might not be suitable for highly distributed settings (e.g., FL) with resource- and bandwidth-constrained devices (e.g., mobile phones). Finally, existing methods do not protect the intermediate model distributed by the server in each iteration, making it vulnerable to privacy attacks by adversaries disguised as clients.

Differential privacy (DP) [17], as a popular statistical tool, can mitigate these privacy leakages effectively. This is achieved by injecting a controlled noise to the model to distort the effects of individual data points on model parameters. Abadi et al. [18] introduced the concept of DP in deep learning by proposing DP-SGD, a privacy-preserving version of the well-known SGD algorithm. In the FL setting, DP can be applied independently or in conjunction with secure aggregation at various stages of the training process (e.g., [19], [20], [21], [22], [23], [24], [25]), taking into account different performance impacts and adversarial models. Accordingly, implementing these privacy-preserving techniques often depends on thoroughly understanding the adversarial and threat models involved. Existing approaches have primarily been designed focusing on individual privacy-preserving methods (i.e., secure aggregation or DP); therefore, there remains a significant gap regarding a comprehensive threat model tailored for privacy-

Yiwei Zhang and Elisa Bertino are with Purdue University. E-mail: yiweizhang, bertino@purdue.edu.

Rouzbeh Behnia, Attila A. Yavuz, Reza Ebrahimi are with the University of South Florida. E-mail: behnia, attilayavuz, ebrahimim@usf.edu.

preserving FL. We stress that establishing a comprehensive threat model and evaluating the effectiveness of recommended privacy-preserving methods for each threat scenario would be critical for organizations striving to address diverse privacy requirements and comply with regulations such as HIPAA.

TABLE I: High-level Comparison with State-of-the-Art

Protocol	Rd.	Dropout Resilience	Model Privacy	PQ
[7]	6	Low	✗	✗
[26]	3	Moderate	✗	✗
[27]	3	Low	✗	✗
[10]	3	Low	✗	✓
[6]	1	High	✗	✗
Ours	1	High	✓	✓

### A. Our Contribution

In response to the above requirements, we propose *Beskar*. To our knowledge, *Beskar* is the first to introduce a comprehensive threat model for FL settings by considering adversaries with different capabilities. As shown in Table I, *Beskar* is the only solution offering high dropout resilience with only one communication round, while simultaneously ensuring post-quantum (PQ) security and privacy of user data during and after training. *Beskar fills the critical gaps in existing protocols by its balance of privacy, minimal communication requirements, and robustness against quantum attacks, making it a holistic solution for FL in the post-quantum era.* We detail the contribution of our work in what follows.

- **Efficient Post-Quantum Secure Aggregation.** We propose a new secure aggregation framework with post-quantum security by leveraging NIST post-quantum standards. Despite their elegant design, most of the suggested standards (e.g., [28], [29]) are not optimized for resource-constrained environments, such as mobile devices, where minimizing computational overhead is critical. While prior works (e.g., [30]) have demonstrated the effectiveness of pre-computation techniques for minimizing the overhead for mobile devices, applying these methods to their post-quantum counterparts is not straightforward due to their inherent design requirements (e.g., rejection sampling). Other methods (e.g., [31]) rely on precomputed tables that might not be suitable for mobile devices due to their significant storage overhead. To address these challenges, we develop two lightweight yet efficient pre-computation strategies that explicitly account for the rejection sampling of post-quantum methods and eliminate the need for large lookup tables. Our methods are specifically designed for low-end devices (e.g., mobile devices) with limited computational and storage resources, ensuring practical deployment in resource-constrained environments. Both optimizations leverage the characteristic that FL operates over a relatively small number of iterations, typically in the order of hundreds. Our first optimization algorithm improves the signature generation of Dilithium [32] by pre-computing message-independent tokens, thereby minimizing the signature generation overhead. This results in a 30% faster signing process compared to the standard Dilithium algorithm. Generating the masking terms to

hide each element of user gradient is one of the dominant costs in secure aggregation protocols designed for larger deep learning models. Our second optimization algorithm significantly improves this process by achieving favorable computation and storage trade-offs and pre-computing a masking table for each client. Our experiments demonstrate that the two optimizations yield efficiency improvements of 134x, 1.1x, and 1233x in the aggregation phase with 1,000 clients.

- **Comprehensive Threat Models for FL.** Existing approaches [26], [27], [10], [33], [34] typically assume a single type of adversary, whereas FL applications face diverse threats and have to meet varying security requirements depending on their specific use cases. We thus propose the first comprehensive security framework for FL settings by considering three types of adversaries with varying capabilities and access levels. We define three distinct threat models corresponding to these adversaries, each posing a unique threat to the privacy of local updates, intermediate models, and final models, respectively. This comprehensive approach enables developers and organizations to identify the threat model most pertinent to their privacy needs or mandated by regulatory frameworks such as HIPAA. Thus, a customized security solution is provided that aligns with specific compliance obligations and protection needs and guarantees robust and targeted privacy safeguards.

- **Various Compositions of Secure Aggregation with DP.** Existing approaches [7], [10], [35] primarily focus on enhancing secure aggregation using various strategies (e.g., MPC, HE) or applying DP methods separately. However, these approaches do not offer comprehensive privacy protections for FL, particularly against the threat models discussed above. To address this gap, we integrate different DP applications within FL to provide tailored privacy protection against various threat models. Our approach involves applying DP at different stages of FL training, effectively defending against the adversaries defined in our threat models. Additionally, we conduct a thorough performance analysis of our approach, emphasizing the trade-offs between security, performance, and accuracy. To our knowledge, this is the first work to analyze the performance of secure aggregation protocols using different DP approaches, offering valuable insights into their effectiveness across various scenarios.

## II. PRELIMINARY

Vectors are denoted by bold letters (e.g.,  $\mathbf{a}$ ). We define a pseudorandom function  $\text{PRF} : \{0, 1\}^* \rightarrow \mathbf{a}$  where  $\mathbf{a}$  shares the same dimension as the global model. The number of elements in a list  $\mathcal{L}$  is represented as  $|\mathcal{L}|$ . Given two vectors  $\mathbf{a}$  and  $\mathbf{b}$  with the same dimension,  $\mathbf{a} + \mathbf{b}$  denotes the element-wise addition.  $[j]$  denotes the set  $\{1, \dots, j\}$ . In the following, we define the cryptographic building blocks used in our protocol.

**Definition 1** (Key-encapsulation [29]). *A key-encapsulation mechanism  $\mathcal{E} = \{\text{KeyGen}, \text{Encaps}, \text{Decaps}\}$  with key space  $\mathcal{K}$  is defined as follows.*

- $(\text{pk}_{\mathcal{E}}, \text{sk}_{\mathcal{E}}) \leftarrow \text{KeyGen}(1^{\kappa})$ : *On the input of the security parameter  $\kappa$ , it returns a pair consisting of a public and private key  $(\text{pk}_{\mathcal{E}}, \text{sk}_{\mathcal{E}})$ .*

- $(c_x, x) \leftarrow \text{Encaps}(\text{pk}_\mathcal{E})$ : On the input of the public key, it returns a key  $x \in \mathcal{K}$  and its ciphertext  $c_x$ .
- $\{\perp, x\} \leftarrow \text{Decaps}(\text{sk}_\mathcal{E}, c_x)$ : On the input of the secret key and ciphertext, it either outputs the key  $x \in \mathcal{K}$  or  $\perp$ , indicating rejection.

A key encapsulation algorithm is  $(1-\beta)$ -correct if  $\Pr(c_x \leftarrow \mathcal{E}.\text{Decaps}(\text{sk}_\mathcal{E}, c_x) : (c_x, x) \leftarrow \mathcal{E}.\text{Encaps}(\text{pk}_\mathcal{E})) \geq 1-\beta$  where probability is taken over  $\mathcal{E}.\text{KeyGen}(\cdot)$  and  $\mathcal{E}.\text{Encaps}(\cdot)$ . The standard security notion for a key-encapsulation algorithm is indistinguishably under a chosen-ciphertext attack (IND-CCA) where the adversary has access to a  $\mathcal{E}.\text{Decaps}(\cdot)$  oracle.

**Definition 2** (Digital Signatures). A digital signature scheme for a messages space  $\mathcal{M}_\Pi$  is defined by  $\Pi: (\text{KeyGen}, \text{Sign}, \text{Verify})$ :

- $(\text{pk}_\Pi, \text{sk}_\Pi) \leftarrow \text{KeyGen}(1^\kappa)$ : On the input of security parameter  $\kappa$ , it returns a pair consisting of a public and private key  $(\text{pk}_\Pi, \text{sk}_\Pi)$ .
- $\sigma \leftarrow \text{Sign}(\text{sk}_\Pi, m)$ : On the input of a secret private key  $\text{sk}_\Pi$  and a message  $m \in \mathcal{M}_\Pi$ , it outputs a signature  $\sigma$ .
- $\{0, 1\} \leftarrow \text{Verify}(\text{pk}_\Pi, m, \sigma)$ : On the input of a public key  $\text{pk}_\Pi$ , a message  $m \in \mathcal{M}_\Pi$ , and an alleged signature  $\sigma$ , it outputs 1 if  $\sigma$  is a valid signature under  $\text{pk}_\Pi$  for message  $m$ . Otherwise, it outputs 0.

A digital signature scheme is correct if for any  $m \in \mathcal{M}_\Pi$ ,  $\text{Verify}(\text{pk}_\Pi, m, \sigma) = 1$ , where  $(\text{pk}_\Pi, \text{sk}_\Pi) \leftarrow \text{KeyGen}(1^\kappa)$  and  $\sigma \leftarrow \text{Sign}(\text{sk}_\Pi, m)$ . The standard security notion for a digital signature scheme is existential unforgeability against adaptive chosen message attacks (EU-CMA), defined in Appendix B.

Following Bell et al. [36], we utilize the following  $\alpha$ -summation ideal functionality to prove the privacy of our protocol. The  $\alpha$ -summation ideal functionality necessity for a sufficient proportion of honest clients is to ensure that the aggregated model in the secure aggregation setting does not leak any information about each user update.

**Definition 3** ( $\alpha$ -summation ideal functionality [36]). Given  $p, n, d$  as integers, we let  $L \subseteq [n]$  and  $\mathcal{X}_L := \{\mathbf{x}_i\}_{i \in L}$  where  $\mathbf{x}_i \in \mathbb{Z}_p^d$ . Now, given a  $0 \leq \alpha \leq 1$  and  $Q_L$  as the set of partitions of  $L$  and a set of pairwise disjoint subsets  $\{L_1, \dots, L_l\} \in Q_L$ , the  $\alpha$ -summation ideal functionality  $\mathcal{F}_{\mathbf{x}, \alpha}(\cdot)$  computes  $\mathcal{F}_{\mathbf{x}, \alpha}(\{L_i\}_{i \in [1, \dots, l]}) \rightarrow \{\mathbf{s}_i\}_{i \in [1, \dots, l]}$  where

$$\forall j \in [1, \dots, l], \mathbf{s}_j = \begin{cases} \sum_{j \in Q_L} \mathbf{x}_j & \text{if } |Q_L| \geq \alpha |L| \\ \perp & \text{else.} \end{cases}$$

**Definition 4** (The MSIS Problem [28]). With an algorithm  $\mathcal{A}$ , we associate the advantage function  $\text{Adv}_{m, k, \gamma}^{\text{MSIS}}$  to solve the  $\text{MSIS}_{m, k, \gamma}$  problem over the ring  $R_q$  as

$$\text{Adv}_{m, k, \gamma}^{\text{MSIS}}(\mathcal{A}) := \Pr(0 < \|y\|_\infty \leq \gamma \wedge (\mathcal{I}|\mathcal{A}) \cdot y = 0 \mid \mathcal{A} \leftarrow R_q^{m \times k}; y \leftarrow A(\mathcal{A}))$$

**Definition 5** (Differential Privacy [37]). A randomized mechanism  $\mathcal{M}: \mathcal{X} \rightarrow \mathcal{Y}$  satisfies  $(\epsilon, \delta)$ -DP, if for any two adjacent datasets  $X, X' \in \mathcal{X}$  that differ in only a single data element

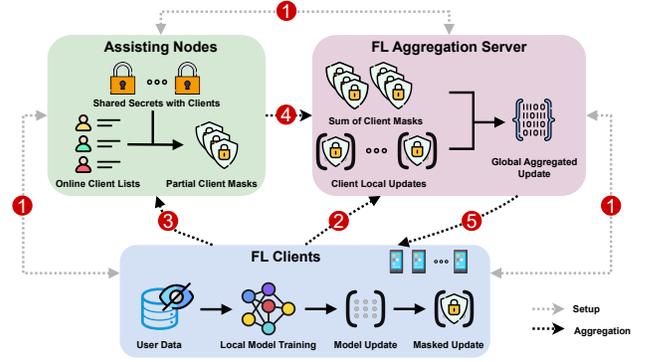


Fig. 1: Beskar's System Model

and for any subset of output  $Y \subseteq \mathcal{Y}$ ,  $\Pr(M(X) \in Y) \leq \exp(\epsilon) \cdot \Pr(M(X') \in Y) + \delta$  holds.

The parameter  $(\epsilon, \delta)$  is often called the *privacy budget*. Specifically,  $\epsilon$  represents the privacy guarantee: a lower  $\epsilon$  corresponds to a higher level of privacy; and  $\delta$  indicates the probability that the upper-bound does not hold.

### III. MODELS

In this section, we first introduce the system model for Beskar, then define its threat and security models.

#### A. System Model

We define a system model similar to models defined by previous work [6], [26] (see Fig. 1). Our system consists of three types of participants: an aggregation server, a set of  $n$  users with their local dataset collaborating to train a central model, and  $k$  assisting nodes, which assist the aggregation server in unmasking the final model without leaking any individual gradient. Beskar offers a one-time setup (1) for  $T$  training iterations. After the setup, each client trains the model on their local data, masks the local updates, and sends the masked updates to the aggregation server (2) with a simple participation message broadcast to all  $k$  assisting nodes (3). After receiving all the masked updates, the server receives the aggregated masking terms from the  $k$  assisting nodes (4). It then aggregates all the provided information to obtain the unmasked global model (5). To address the different privacy requirements, Beskar seamlessly adopts DP in different stages of training, depending on the target privacy requirements. We discuss each variation in Section IV.

#### B. Threat Model

Attacks on FL systems predominantly aim to compromise the integrity and confidentiality of these systems [4], [7], [26]. These include traditional Man-in-the-middle (MITM) attacks and emerging attacks targeting the model to compromise the privacy of the training data. Following [26], we consider a malicious MITM attacker. This is the strongest adversary in the context of FL [26], [27], [7]. More specifically, aside

from the ability to analyze communication, such an adversary can actively force users to drop out, and drop or replace messages, compromising up to  $(k - 1)$  assisting nodes. The adversary is assumed to have access to quantum computers capable of breaking conventional cryptographic problems. Consistent with [26], we consider a secure and authenticated channel between the entities (e.g., via [38]). For simplicity and following prior privacy-preserving approaches for the FL setting [26], [27], [6], we assume that the compromised parties will follow the protocol.

Attacks targeting the model to compromise data privacy, irrespective of the adversarial method utilized—whether black-box or white-box—can be classified, based on severity, into membership inference, model inversion, and training data extraction attacks. Such adversaries can attack user gradients and intermediate models during training and the final model after deployment [1] by compromising different entities involved in the protocol. Unlike traditional centralized training, where an adversary typically has access only to the final model, FL involves multiple participants who may have different objectives and varying levels of trust. Since training occurs in a distributed setting, privacy concerns are not limited to the deployed model but also include the entire training process. As highlighted by [39], it is crucial to protect user data both during training and after deployment. To this end, we propose a full-stack threat model that addresses privacy risks at every stage of the training pipeline and after deployment. This includes attacks on user updates sent to the server, the intermediate global models shared with clients, and the final deployed model. Note that extensive research has been conducted on model correctness and poisoning attacks [6], which can be utilized alongside our method to enhance client data privacy protection, hence outside the scope of our work. Therefore, we define three threat models, categorized based on the adversary’s capabilities and access to the model in different stages (i.e., user gradients, intermediate or final models). Following the above attack categories, *the adversary succeeds if it can infer any information about the training data* (i.e., the membership inference attack).

- **Threat Model 1:** TM1 considers an adversary targeting user gradients during the training phase to undermine the privacy of the honest users’ data. In TM1, the adversary captures a compromised aggregation server that aims to undermine the clients’ data privacy through its access to their masked gradient. A real-world example is a cloud provider running the federated learning server which, despite performing aggregation as expected, attempts to reconstruct sensitive information (e.g., handwritten digits or medical conditions) from user-submitted updates using gradient inversion attacks.
- **Threat Model 2:** TM2 considers an adversary targeting the intermediate model during the training phase. This threat model accounts for a compromised aggregation server or a subset of the clients (or both) that aim to undermine the clients’ data privacy through their access to the intermediate model computed and distributed by the aggregation server at the end of each iteration. A

practical scenario is a group of colluding clients in a cross-silo FL setup (e.g., hospitals sharing models for disease prediction) using model update differences across rounds to infer training data from other participants.

- **Threat Model 3:** TM3 considers privacy attacks after the training phase, once the model is deployed. As stated above, these attacks can occur via white-box and black-box access. A real-world instance is a deployed language model accessed via an API (black-box) or downloaded in its entirety (white-box), where adversaries attempt membership inference or data reconstruction attacks to determine whether specific records (e.g., patient names or user queries) were part of the training data.

A protocol is considered to provide *full-stack* privacy if it effectively safeguards user data across all the above threat models, ensuring comprehensive protection throughout the entire lifecycle of the model.

### C. Security Models

In the following, we define the security models for our primitives. The standard security notion for a digital signature scheme is existential unforgeability against adaptive chosen message attacks (EU-CMA), defined below.

**Definition 6** (EU-CMA). *Existential Unforgeability under Chosen Message Attack (EU-CMA) experiment  $Exp_{\Pi, A}^{EU-CMA}$  for a signature scheme  $\Pi: (\text{KeyGen}, \text{Sign}, \text{Verify})$  with an adversary  $A$  is defined as follows.*

- $(\text{sk}, \text{pk}) \leftarrow \Pi.\text{KeyGen}(1^\kappa)$
- $(m^*, \sigma^*) \leftarrow A^{\Pi.\text{Sign}(\cdot)}(\text{pk})$
- If  $1 \leftarrow \Pi.\text{Verify}(\cdot)$  and  $m^*$  was never queried to  $\text{Verify}(\cdot)$ , return ‘success’ otherwise, output ‘ $\perp$ ’.

The main security notion for a key encapsulation mechanism is the indistinguishability of the chosen ciphertext attack (IND-CCA), which enables the adversary to access the decapsulation mechanism.

**Definition 7** (IND-CCA). *IND-CCA property of a key encapsulation scheme  $\mathcal{E} = \{\text{KeyGen}, \text{Encaps}, \text{Decaps}\}$  with key space  $\mathcal{K}$  under chosen ciphertext attack (IND-CCA) experiment  $Exp_{\mathcal{E}, A}^{IND-CCA}$  with an adversary  $A$  is defined as follows.*

- $(\text{sk}, \text{pk}) \leftarrow \Pi.\text{KeyGen}(1^\kappa)$
- $b \leftarrow \{0, 1\}, (c_x, x_0) \leftarrow \Pi.\text{Encaps}(\text{pk})$
- $x_1 \leftarrow \mathcal{K}$
- $b' \leftarrow A^{\Pi.\text{Encaps}(\cdot), \Pi.\text{Decaps}(\cdot)}(\text{pk}, c_x, x_b)$

*The advantage of the adversary in the above experiment is defined as  $\Pr[b = b'] \leq \frac{1}{2} + \varepsilon$  for a negligible  $\varepsilon$ .*

## IV. POST-QUANTUM FEDERATED LEARNING

Given a secure public key infrastructure, `Beskar` only requires a single setup round to train a central model (via multiple rounds). `Beskar` provides security against quantum adversaries while incurring the least computational overhead compared to its counterparts with conventional (non-quantum) security. Below, we present the high-level ideas of `Beskar`, followed by a detailed protocol.

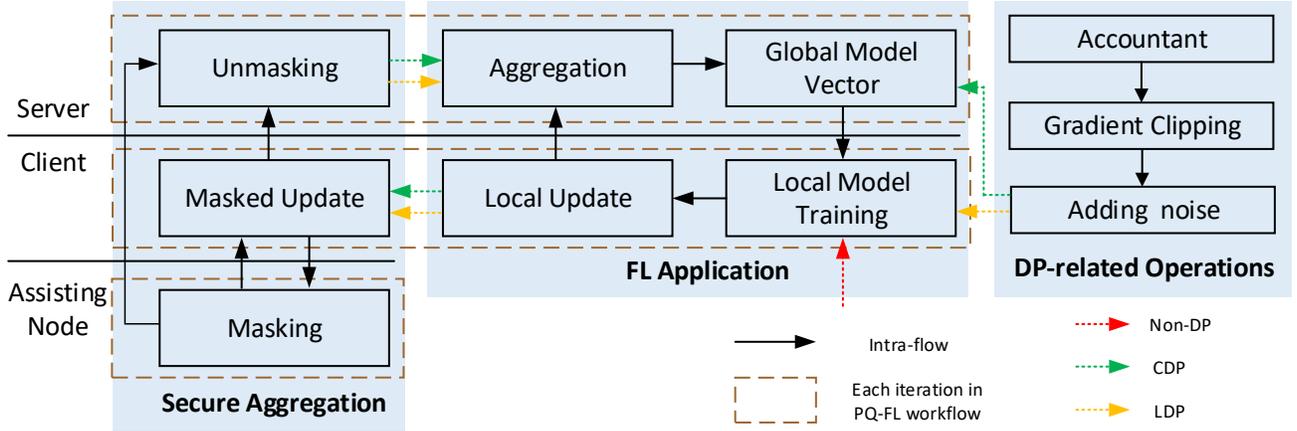


Fig. 2: An overview of Beskar’s architecture and how it fits in the existing FL workflow.

### A. High-level Idea

The design of Beskar is based on two main observations: (1) The main overhead in privacy-preserving FL arises from the underlying cryptographic operations. This is further exacerbated when post-quantum security is considered. (2) The training takes place over a small number of iterations, typically between 20 and 50. With these insights, we design a new, highly efficient privacy-preserving FL framework by devising a series of precomputation methods to reduce the cryptographic overhead while attaining post-quantum security. Following the recent development in practical secure aggregation protocols (e.g., [26], [6]), Beskar assumes a set of assisting nodes to assist the aggregation server in unmasking the intermediate/final model. In addition, to ensure privacy against different threat models, we integrate Beskar with different DP methods. Fig. 2 shows the high-level architecture of Beskar and its integration with the FL training process.

### B. FL Secure Aggregation with Post Quantum Security.

1) *Setup Phase*: The setup phase is a one-time process that can be performed, at least partially, offline. This phase consists of a single round, namely, *KeyGen and Advertise*. During this round, all users  $P_1, \dots, P_n$  and assisting nodes  $A_1, \dots, A_k$  are initialized with the two pairs of public keys, i.e.,  $(sk_{\mathcal{E}}, pk_{\mathcal{E}})$  for the key exchange scheme and  $(sk_{\Pi}, pk_{\Pi})$  for the digital signature scheme. Then, a public-key exchange procedure is performed as follows: (1) the users receive copies of the key exchanging public keys from each assisting node  $pk_{\mathcal{E}}^{A_j}$ ; (2) the assisting nodes receive copies of the user signing public keys  $pk_{\Pi}^{P_i}$ ; and (3) the aggregation server receives copies of the signing public keys from the assisting nodes  $pk_{\Pi}^{A_j}$  and the users  $pk_{\Pi}^{P_i}$ . After that, each user computes a shared secret  $x_{P_i}^{A_j}$  for each assisting node  $A_j$  with  $sk_{\mathcal{E}}^{P_i}$  and  $pk_{\mathcal{E}}^{A_j}$  via the key exchange scheme  $\mathcal{E}.Encaps(\cdot)$  and securely sends it to the corresponding assisting node  $A_j$ . The assisting node  $A_j$  can recover the shared secret using its private key  $sk_{\mathcal{E}}^{A_j}$  via  $\mathcal{E}.Decaps(\cdot)$ .

2) *Aggregation Phase*: The aggregation phase (Algorithm 2) comprises two rounds: *Masking Updates* and *Aggregate Updates*. The aggregation phase is repeated at run-time with several iterations based on the FL training requirements.

Specifically, at  $t$ -th iteration, in the first round (*Masking Updates*), each user computes a masking vector  $a_t^{P_i}$  using the shared secrets  $x_{P_i}^{A_j}$  for  $j = 1, \dots, k$  to mask its user gradient  $w_t^{P_i}$ . The user then sends the masked gradient  $y_t^{P_i}$  and a participation message (e.g., the iteration number) to the aggregation server and the  $k$  assisting nodes, respectively. All the messages are signed using  $\Pi.\text{Sign}(\cdot)$ .

In the second round (*Aggregate Updates*), upon receiving (and verifying) the participation message (and signature), each  $A_j$  adds the user to the list  $\mathcal{L}_{j,t}$ . For all users in  $\mathcal{L}_{j,t}$ ,  $A_j$  computes the aggregation of their masking vectors  $a_t^{A_j}$  using the shared keys  $x_{P_i}^{A_j}$  and sends  $a_t^{A_j}$  to the aggregation server, in which the messages are signed by assisting nodes using  $\Pi.\text{Sign}(\cdot)$ . Next, when receiving (and verifying) the participation message (and signature), the aggregation server  $\mathcal{S}$  adds the user to the list  $\mathcal{L}_{s,t}$  and then verifies that all user lists from the assisting nodes and the aggregation server are identical. For all users in  $\mathcal{L}_{s,t}$ , the server uses the masked updates ( $y_t^{P_i}$  and the aggregated masking terms  $a_t^{A_j}$ ) provided by the assisting nodes to efficiently compute the aggregated intermediate model.

3) *Precomputed Masks*: To avoid plain gradient transmission, Beskar generates a mask for each client local update and only transmits the masked update. On the user side, a mask is generated with the shared secret of the user and each assisting node  $x_{P_i}^{A_j}$  and the iteration number  $t$ . As the shared secrets are calculated in the setup phase and the iteration number is iterated in order, there is no any runtime information (i.e., local updates) involved. Therefore, we can precompute  $T$  masks for  $T$  iterations before the runtime aggregation.

### C. Resiliency Against User and Assisting Node Dropouts

The design of Beskar offers strong resilience against both user and assisting node dropouts. Specifically, Beskar is a

---

**Algorithm 1** *Beskar* Setup

**Input:** All parties are provided with the security parameter  $\kappa$ , the number of assisting nodes  $k$ , a key encapsulation protocol  $\mathcal{E} = \{\text{KeyGen}, \text{Encaps}, \text{Decaps}\}$ , a digital signature scheme with precomputation  $\Pi$ :  $(\text{KeyGen}, \text{Precomp}, \text{PSgn}, \text{Verify})$ , instantiated using the security parameter  $\kappa$ .

**Output:** All the entities generate their key exchange, signature key pairs, as well as precomputed signing parameter list  $\mathcal{LS}$ . All the users receive the public keys of all the assisting nodes  $((\text{pk}_{\Pi}^{A_1}, \dots, \text{pk}_{\Pi}^{A_k}), (\text{pk}_{\mathcal{E}}^{A_1}, \dots, \text{pk}_{\mathcal{E}}^{A_k}))$ , and shared secret seed  $\mathbf{x}_{A_j}^{P_i}$  for each assisting node  $A_j$ . All the assisting nodes receive the public keys of all the participating users  $(\text{pk}_{\Pi}^{P_1}, \dots, \text{pk}_{\Pi}^{P_n})$ , and a shared secret seed  $\mathbf{x}_{P_i}^{A_j}$  with each user  $P_i$ . The aggregation server  $S$  receives the public keys of all the users and assisting nodes  $((\text{pk}_{\Pi}^{P_1}, \dots, \text{pk}_{\Pi}^{P_n}), (\text{pk}_{\Pi}^{A_1}, \dots, \text{pk}_{\Pi}^{A_k}))$ . Lastly, all the users and assisting nodes generate the precomputed mask lists  $\mathcal{MS}$ .

---

**Phase 1 (KeyGen and Advertise)**

All the communications below are conducted via an authenticated channel (similar to [4], [27])

- 1: Each assisting node  $A_j$  generates its key pair(s)  $(\text{sk}_{\Pi}^{A_j}, \text{pk}_{\Pi}^{A_j}) \leftarrow \Pi.\text{KeyGen}(\kappa)$  and  $(\text{sk}_{\mathcal{E}}^{A_j}, \text{pk}_{\mathcal{E}}^{A_j}) \leftarrow \mathcal{E}.\text{KeyGen}(\kappa)$  and sends  $(\text{pk}_{\Pi}^{A_j}, \text{pk}_{\mathcal{E}}^{A_j})$  to the  $n$  users and sends  $\text{pk}_{\Pi}^{A_j}$  the aggregation server.
  - 2: Upon receiving  $\text{pk}_{\mathcal{E}}^{A_j}$  from  $A_j$ , each user  $P_i$  computes and encapsulate a shared secret by  $(\mathbf{x}_{A_j}^{P_i}, c_{A_j}^{P_i}) \leftarrow \mathcal{E}.\text{Encaps}(\text{pk}_{\mathcal{E}}^{A_j})$ . It then generates its key pair(s)  $(\text{sk}_{\Pi}^{P_i}, \text{pk}_{\Pi}^{P_i}) \leftarrow \Pi.\text{KeyGen}(\kappa)$  and sends  $\text{pk}_{\Pi}^{P_i}$  and  $c_{A_j}^{P_i}$  to  $A_j$  to the  $k$  assisting nodes and  $\text{pk}_{\Pi}^{P_i}$  to the aggregation server.
  - 3: Upon receiving  $c_{A_j}^{P_i}$  from the the user  $P_i$ , the assisting node  $A_j$  computes and retrieves the shared secret  $\mathbf{x}_{A_j}^{P_i} \leftarrow \mathcal{E}.\text{Decaps}(c_{A_j}^{P_i}, \text{sk}_{\mathcal{E}}^{A_j})$
  - 4: The aggregation server  $S$  generates a key pair  $(\text{sk}_{\Pi}^S, \text{pk}_{\Pi}^S) \leftarrow \Pi.\text{KeyGen}(1^\kappa)$  and sends  $\text{pk}_{\Pi}^S$  to the users.
  - 5: All the entities perform the precomputation to get a list  $\mathcal{LS}$  with  $N$  groups of pre-computed parameters for optimizing further signing operations:  $\mathcal{LS}_{A_j} \leftarrow \text{Precomp}(\text{sk}_{A_j}, N)$ ,  $\mathcal{LS}_{P_i} \leftarrow \text{Precomp}(\text{sk}_{P_i}, N)$ ,  $\mathcal{LS}_S \leftarrow \text{Precomp}(\text{sk}_S, N)$ .
  - 6: Each user  $P_i$  computes a list  $\mathcal{MS}_{P_i}^{A_j}$  for each assisting node  $A_j$ , with  $T$  groups of masks:  $\mathcal{MS}_{P_i}^{A_j}[t] = \text{PRF}(\mathbf{x}_{P_i}^{A_j}, t)$  for  $t \in [1, \dots, T]$ . Similarly, each assisting node  $A_j$  computes a list  $\mathcal{MS}_{A_j}^{P_i}$ :  $\mathcal{MS}_{A_j}^{P_i}[t] = \text{PRF}(\mathbf{x}_{A_j}^{P_i}, t)$  for  $t \in [1, \dots, T]$ .
- 

**Algorithm 2** *Beskar* Aggregation

**Input:** The iteration  $t$ , a user calculated model update  $\mathbf{w}_t^{P_i}$ , a keyed pseudorandom function PRF, a digital signature scheme  $\Pi$ , the secret seeds  $\mathbf{x}$  (shared between the users and the nodes), the signature key pair of the users and the assisting nodes, as well as the precomputed signing parameter list  $\mathcal{LS}$  of all entities and the precomputed mask list  $\mathcal{MS}$  of the users and the assisting nodes.

**Output:** The final model update  $\mathbf{w}_t \in \mathcal{M}^d$ .

---

**Phase 1 (Masking Updates)**

- 1: The user compute the local update  $\mathbf{w}_t^{P_i}$  for iteration  $t$
  - 2: The user  $P_i$  first gets the mask  $\mathcal{MS}_{P_i}^{A_j}[t]$  for iteration  $t$ , and then computes  $\mathbf{a}_t^{P_i} = \sum_{j=1}^k \mathcal{MS}_{P_i}^{A_j}[t]$  and the masked update  $\mathbf{y}_t^{P_i} = \mathbf{w}_t^{P_i} + \mathbf{a}_t^{P_i}$ .
  - 3: The user sets  $m = (t, \mathbf{y}_t^{P_i})$  and  $m' = (t)$ , computes signatures  $\sigma_S^{P_i} \leftarrow \Pi.\text{PSgn}(\text{sk}_{\Pi}^{P_i}, m, \mathcal{LS}_{P_i})$  and  $\sigma_{A_j}^{P_i} \leftarrow \Pi.\text{PSgn}(\text{sk}_{\Pi}^{P_i}, m', \mathcal{LS}_{P_i})$  and sends  $(m, \sigma_S^{P_i})$  and  $(m', \sigma_{A_j}^{P_i})$  to the aggregation server and the assisting node  $A_j$  (for  $j \in [1, \dots, k]$ ), respectively.
- 

**Phase 2 (Aggregate Updates)**

- 1: Upon receiving  $(m', \sigma_{A_j}^{P_i})$  from all the users in the system,  $A_j$  checks if  $\Pi.\text{Verify}(\text{pk}_{\Pi}^{A_j}, m', \sigma_{A_j}^{P_i}) \stackrel{?}{=} 1$  holds, it adds the user to its user list  $\mathcal{L}_{j,t}$ .
  - 2: Each assisting node checks if  $|\mathcal{L}_{j,t}| \geq \alpha \mathcal{P}_H$ , it gets  $\mathcal{MS}_{A_j}^{P_i}[t]$ , computes  $\mathbf{a}_t^{A_j} = \sum_{i=1}^{|\mathcal{L}_{j,t}|} \mathcal{MS}_{A_j}^{P_i}[t]$ , sets  $m'' = (t, |\mathcal{L}_{j,t}|, \mathbf{a}_t^{A_j})$  and  $\sigma_S^{A_j} \leftarrow \Pi.\text{PSgn}(\text{sk}_{\Pi}^{A_j}, m'', \mathcal{LS}_{A_j})$  and sends  $(m'', \sigma_S^{A_j})$  to the aggregation server  $S$ .
  - 3: Upon receiving  $(m, \sigma_S^{P_i})$ ,  $S$  first checks if  $\Pi.\text{Verify}(\text{pk}_{\Pi}^{P_i}, m, \sigma_S^{P_i}) \stackrel{?}{=} 1$  holds, it adds  $P_i$  to its user list  $\mathcal{L}_{S,t}$ .
  - 4: Next,  $S$  checks if  $\Pi.\text{Verify}(\text{pk}_{\Pi}^{A_j}, m'', \sigma_S^{A_j}) \stackrel{?}{=} 1$  holds for  $j \in [1, \dots, k]$ . It then checks if  $|\mathcal{L}_{S,t}| = |\mathcal{L}_{1,t}| = \dots = |\mathcal{L}_{k,t}|$ , does not hold it aborts.
  - 5:  $S$  computes and broadcasts the final update as  $\mathbf{w}_t = \sum_{i=1}^{|\mathcal{L}_{S,t}|} \mathbf{y}_t^{P_i} - \sum_{j=1}^k \mathbf{a}_t^{A_j}$ .
- 

one-round protocol. Unlike existing approaches [4], [7], [27], user dropouts do not introduce any additional overhead for the remaining participants. As in standard non-private FL, user dropouts may affect model accuracy but do not impact the overall protocol execution. Therefore, *Beskar* does not introduce any additional constraints in this regard. Moreover, the novel design of *Beskar* enables the use of a rotating set of assisting nodes. In scenarios where an assisting node has an unreliable connection or is at risk of dropping out, a simple secret sharing scheme can be used to distribute the node's secret among the other assisting nodes [40]. This schema allows the online assisting parties to collaboratively

reconstruct the required masking terms without imposing any additional burden on the regular users.

#### D. Integrating Differential Privacy

During and after the FL aggregation, attackers under different threat models could exploit the client gradients, intermediate models, and final models to conduct black- and white-box-based model inversion attacks, as outlined in Section III-B. These attacks could allow one to infer information about or reconstruct the model's training data, making secure aggregation alone insufficient to fully protect user training data.

**Algorithm 3** Precomputation Algorithm for Dilithium

---

 $\mathcal{LS} \leftarrow \text{Precomp}(\text{sk}_\Pi, N)$ 


---

```

1:  $\mathcal{LS} := \emptyset$ 
2: while Length of  $\mathcal{LS} < N$  do
3:    $\mathbf{y} \leftarrow \mathbb{S}_{\gamma_1-1}^\ell$ 
4:    $\mathbf{w}_1 := \text{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$ 
5:   Add  $(\mathbf{y}, \mathbf{w}_1)$  into  $\mathcal{LS}$ 
6: return  $\mathcal{LS}$ 

```

---

 $\sigma \leftarrow \text{PSgn}(\text{sk}_\Pi, m, \mathcal{LS})$ 


---

```

1: for each  $(\mathbf{y}, \mathbf{w}_1)$  in  $\mathcal{LS}$ 
2:    $c \in \mathbb{B}_{60} := \mathcal{H}(m \parallel \mathbf{w}_1)$ 
3:    $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$ 
4:   if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ 
5:     or  $\|\text{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta,$ 
6:     then Remove  $(\mathbf{y}, \mathbf{w}_1)$  from  $\mathcal{LS}$ 
7:     return  $\sigma = (\mathbf{z}, c)$ 
8: return  $\text{Sign}(\text{sk}_\Pi, m)$ 

```

---

To tackle that issue, *Beskar* integrates DP techniques with secure aggregation, adding noises to the final, intermediate, and local models during training. Depending on the threat models, in which adversaries with different abilities are involved, we employ two DP methods—**Local Differential Privacy (LDP)** and **Central Differential Privacy (CDP)**—individually or in combination to safeguard the client gradient privacy, intermediate model privacy, and final model privacy. Specifically, LDP is used to protect client gradient updates. Each client applies DP noise during local model training at every iteration, preventing user training data from being exposed to a malicious server or other clients. Note that clients may use different privacy budgets ( $\epsilon$ ), with the overall privacy guarantee determined by the largest  $\epsilon$  among all clients. CDP protects against model inversion attacks on aggregated models. The CDP server adds DP noise to the aggregated model at each training iteration, ensuring that malicious entities cannot infer the individual data of honest clients from the aggregated models.

*Beskar* adopts tailored DP strategies with the two DP methods based on the specific threat model. Under **TM1**, *Beskar* applies LDP (as well as the secure aggregation) to protect the client gradients. By adding noise to client gradients during local client training, a compromised server will be unable to infer the exact training data after receiving the client gradients. Under **TM2** and **TM3**, *Beskar* utilize LDP and CDP to protect the intermediate models and final models against compromised clients and compromised server. Noise is added to the client gradients on the client side and to the aggregated model on the server side, ensuring robust protection under these threat models.

### E. Efficient Post-Quantum Signature

Digital signatures are essential for ensuring the authenticity and integrity of transmitted messages. While post-

quantum signatures promise long-term security against quantum adversaries, they are often more resource-intensive than their classical counterparts. This overhead becomes significant when these schemes are deployed on resource and battery-constrained devices like cellular phones. Several algorithms have been proposed in the NIST post-quantum standards, such as Dilithium [32] and SPHINCS+ [41]. We select Dilithium as the most efficient option [42], and optimize it for deployment in *Beskar*. In the following, we describe Dilithium and introduce a precomputation method to significantly improve its computational overhead in the FL settings.

Dilithium is based on Fiat-Shamir with abort paradigm, and its security relies on the hardness of the modulo-SIS and modulo-LWE problems. In the standard Dilithium signature without any optimization (as Algorithm 4 in Appendix A), key generation involves selecting matrix  $\mathbf{A}$  of dimension  $k \times l$  and computing an MLWE public key  $\mathbf{t} = \mathbf{A}s_1 + s_2$  where  $s_1$  and  $s_2$  are sampled from the key space  $\mathcal{S}_\eta$  with small coefficient of size at most  $\eta$ . The  $\text{HighBits}(\cdot)$  and  $\text{LowBits}(\cdot)$  methods simply select the high-order and low-order bits of the coefficients in their vector, respectively.

Given the Fiat-Shamir paradigm, in lattice-base settings, since we are solving for a variant of the SIS problem, the signature  $\mathbf{z}$  should be small. To ensure this, an eligible signature should meet two conditions: (i) For security, it is crucial that the masking term  $\mathbf{y}$  is completely hiding  $c\mathbf{s}_1$ ; the first rejection condition (Step 7) addresses this concern. (ii) The second rejection condition is required for both the scheme’s correctness and security [43] and ensures that the same  $c$  is recovered in the verification algorithm. In practice, this could require repeating the signing algorithm 7 times before finding an eligible signature [32], [43]. This can have a significant resource and computation overhead for constrained devices.

To address such an issue, we propose a precomputation algorithm for Dilithium by leveraging the fact that FL takes a small number of training iterations (in the order of hundreds). The high-level idea of the precomputation algorithm, presented in Algorithm 3, is to store a list of masking values  $\mathbf{y}$  and their corresponding commitment values  $\mathbf{A}\mathbf{y}$ . Therefore, if the selected masking term does not pass the rejection conditions during the signing algorithm, another value and its corresponding commitment can be selected from the list. Following the one-time nature of the masking term, it will be removed from the list once an eligible masking term is selected. Detailed algorithms are provided in Appendix A.

### F. Security Analysis

**Theorem 1.** *The *Beskar* protocol presented in Algorithms 1 and 2, running with  $n$  parties  $\{P_1, \dots, P_n\}$ ,  $k$  assisting nodes  $\{A_1, \dots, A_k\}$ , and an aggregation server  $S$  provides post-quantum privacy against a malicious adversary  $A \in \mathbf{A1}, \mathbf{A2}, \mathbf{A3}$  which controls  $S$  and  $1 - \alpha$  fraction of users and  $k - 1$  assisting nodes with the offline rate  $(1 - \delta) \geq \alpha$ .*

*Proof.* Following [6], we prove the above theorem by the standard hybrid argument. The proof relies on the assumption

of the simulator  $\text{Sim}$  that controls the environment.  $\mathcal{P}_h$  and  $\mathcal{P}_C$  define the set of honest and corrupt users, respectively.

*Setup phase:*

- 1) Each honest user  $P_i$  and assisting node  $A_j$  follows the protocol in Algorithm 1.
- 2) For each corrupt user  $P_{i'} \in \mathcal{P}_C$  and honest assisting node  $A_j \in \mathcal{A}_H$ , compute and store  $(x_{A_j}^{P_{i'}}, c_{A_j}^{P_{i'}}) \leftarrow \mathcal{E}.\text{Encaps}(\text{pk}_{\mathcal{E}}^{A_j})$ .
- 3) For each honest user  $P_i \in \mathcal{P}_H$  and corrupt assisting node  $A_{j'} \in \mathcal{A}_C$ , compute and store  $(x_{A_{j'}}^{P_i}, c_{A_{j'}}^{P_i}) \leftarrow \mathcal{E}.\text{Encaps}(\text{pk}_{\mathcal{E}}^{A_{j'}})$ .
- 4) For each honest pair of user  $P_i$  and honest assisting node  $A_j$ ,  $\text{Sim}$  picks  $x_r \xleftarrow{\$} \mathcal{K}_\Sigma$  and sets  $x_{A_j}^{P_i} = x_r$ .

*Aggregation phase:*

- 1) In each iteration  $t$  of the protocol, each honest user picks a random  $\mathbf{y}_t^{P_i}$ . It then computes two signatures  $\sigma_S^{P_i} \leftarrow \Pi.\text{Sign}(\text{sk}_{\Pi}^{P_i}, m)$  and  $\sigma_{A_j}^{P_i} \leftarrow \Pi.\text{Sign}(\text{sk}_{\Pi}^{P_i}, m')$  and sends  $(m, \sigma_S^{P_i})$  and  $(m', \sigma_{A_j}^{P_i})$  to the aggregation server and the assisting node  $A_j$  (for  $j \in [1, \dots, k]$ ), respectively.
- 2) The aggregation server  $S$  first adds the user  $P_i$  to the list  $\mathcal{L}_{S,t}$  and then calls the  $\alpha$ -summation ideal functionality  $\mathcal{F}_{x,\alpha}(\mathcal{L}_{S,t} \setminus \mathcal{P}_C)$  (where  $\mathcal{P}_C$  is the set of corrupt users) to get  $\mathbf{w}_t$ .
- 3) Next, the simulator samples  $\mathbf{w}'_t \xleftarrow{\$} \mathcal{M}^d$  for all  $P_i \in \mathcal{L}_{S,t} \setminus \mathcal{P}_C$  such that  $\mathbf{w}_t = \sum_{i \in \mathcal{P}_C} \mathbf{w}'_t$  and computes  $\mathbf{a}'_t = \mathbf{y}_t^{P_i} - \mathbf{w}'_t$ .  $\text{Sim}$  sets  $\{\mathbf{a}'_{A_j,t}\}_{A_j \in \mathcal{A}_H} = \{\text{RO}(x_{A_j}^{P_i}, t)\}_{A_j \in \mathcal{A}_H}$  such that  $\mathbf{a}'_{A_j,t} = \sum_{A_j \in \mathcal{A}_H} \{\mathbf{a}'_{A_j,t}\}$ , and for each  $A_j \in \mathcal{A}_H$ , it computes  $\mathbf{a}'_{P_i,t} = \sum_{P_i \in \mathcal{P}_H} \mathbf{a}'_{j,t}$ .

The hybrids are provided below. We note that each hybrid represents a view of the system seen by the adversary. The proof relies on the indistinguishability of each hybrid. The hybrids are constructed by the simulator  $\text{Sim}$ .

**Hyb0** The random variable is identical to the real execution of the protocol (i.e.,  $\text{REAL}$ ).

**Hyb1** Now,  $\text{Sim}$  that knows the secrets of all honest entities are introduced in this hybrid. The distribution of this hybrid remains identical to the one above.

**Hyb2** This hybrid replaces the shared keys between the users and assisting nodes with a random shared secret key sampled from  $\mathcal{K}_\Sigma$ . The indistinguishability of this hybrid is delivered by the security of the key encapsulation mechanism (Definition 9). Our protocol delivers this by the security of the CRYSTALS-Kyber algorithm [29], which is based on the Module-LWE problem which is post-quantum secure.

**Hyb3** This hybrid starts with each honest user picking a random vector  $\mathbf{y}_t^{P_i}$ . The user then computes a signature  $\sigma_S^{P_i} \leftarrow \Pi.\text{Sign}(\text{sk}_{\Pi}^{P_i}, \langle \mathbf{y}_t^{P_i}, t \rangle)$  and sends  $(\mathbf{y}_t^{P_i}, \sigma_S^{P_i})$  to the server. The indistinguishability of this hybrid is due to the following. 1) Given we need at least one assisting node to remain honest, and since  $A$  does not have the secret of honest entities, in  $\text{REAL}$ , the masked gradient vector will have the same distribution as  $\mathbf{y}'_t$  and therefore indistinguishable. 2) The indistinguishability of

$\sigma_S^{P_i}$  is delivered via the security requirement (Definition 8) of the underlying signature scheme.

**Hyb4** The ideal functionality  $\mathcal{F}_{x,\alpha}(\mathcal{L}_{S,t} \setminus \mathcal{P}_C)$  and random oracles are used to substitute the aggregated mask outputted by the honest assisting nodes  $A_i \in \mathcal{A}_H$  with  $\mathbf{a}'_{P_i,t}$  (computed in Step 3 of the simulated Aggregation phase presented above). This hybrid outputs  $(\mathbf{a}'_{P_i,t}, \sigma_S^{A_i})$  where  $\sigma_S^{A_i} \leftarrow \text{Sign}(\text{sk}_{\Pi}^{A_j}, m'')$ . Given the ideal functionality, random oracles, and digital signatures, and since  $A$  does not have the secret of honest entities, the view of this hybrid is indistinguishable with the previous hybrid.

**Hyb5** In the final hybrid,  $\text{Sim}$  outputs the output of the ideal functionality (Step 2 in the simulated Aggregation phase) as the global update  $\mathbf{w}_t$ . Given the assumption of the fraction of honest users/nodes, the ideal functionality will not output  $\perp$  with an overwhelming probability. Thus, this hybrid is indistinguishable from the previous one.

In the above, we have shown that the view of all the corrupted parties controlled by  $A$  is computationally indistinguishable. This ensures that the masked gradients are essentially random values, providing full privacy and thereby protecting the privacy of the clients' dataset in the sense of  $\text{TM1}$ .  $\square$

**Corollary 1.** *The protocol presented in Algorithm 2 protects privacy against a malicious adversary capable of controlling communication and arbitrarily dropping users.*

*Proof.* The privacy guarantee of  $\text{Beskar}$  against user dropouts is provided by the  $\alpha$ -summation ideal functionality (Definition 3) and enforced by a condition check during the Aggregation phase (Algorithm 2). More specifically, to prevent privacy leakages, in the protocol each assisting node checks if at least  $\alpha \mathcal{P}_H$  users are participating in the training phase. This ensures that the aggregation process continues only when a sufficient number of users are contributing, preventing the adversary from inferring individual data through selective dropouts.  $\square$

**Corollary 2.** *Given a malicious user, the protocol presented in Algorithm 2 combined with the CDP method provides  $(\epsilon_c, \delta_c)$ -privacy in the context of  $\text{TM2}$ , and  $\text{TM3}$ .*

*Proof.* Following the proof of Theorem 1,  $\text{Beskar}$  protects the privacy of user gradients and allows the aggregation server to compute the intermediate (or final) model without revealing any information about the gradients. The Central Differential Privacy (CDP) method is applied on the server side, ensuring that the intermediate (or final) model is  $(\epsilon_c, \delta_c)$ -protected against malicious clients during training and deployment.

However, it is essential to note that since CDP is implemented on the server, a malicious server can access the plain intermediate (or final) model before CDP is applied. Consequently, while CDP provides  $(\epsilon_c, \delta_c)$ -privacy from malicious clients during training and deployment, it still assumes an honest aggregation server.  $\epsilon_c$  and  $\delta_c$  define the central privacy budget and the failure probability computed by a privacy accountant on the server side.  $\square$

**Corollary 3.** *Given a malicious aggregation server, the protocol presented in Algorithm 2 combined with the LDP method provides  $(\epsilon_l, \delta_l)$ -privacy in the context of TM1, TM2, and TM3.*

*Proof.* Compared to the previous one, the key distinction in this corollary lies in the use of Local Differential Privacy (LDP) alongside `Beskar` and the removal of the trusted server assumption. Specifically, in the LDP setting, obscuring user gradients occurs on the client side, ensuring that the intermediate and final models computed by the aggregation server maintain  $(\epsilon_l, \delta_l)$ -privacy at all times. It is important to note that `Beskar` fully conceals user gradients. Here,  $\epsilon_l$  and  $\delta_l$  represent the largest privacy budget and the corresponding probability of failure among the participating clients.  $\square$

## V. PERFORMANCE EVALUATION

### A. Analytical Evaluation

We present a comparative analysis of the computational performance of `Beskar` and several state-of-the-art protocols in Table II.

1) *Setup Phase:* The setup phase of `Beskar` involves two runs of the key generation protocol and  $k$  runs of the key agreement protocol for the user, or  $|U|$  runs for the assisting node. Additionally, each user and assisting node perform  $T \times k$  and  $T \times |U|$  PRF invocations, respectively, to precompute  $T$  groups of masks, along with  $N$  `Precomp` operations to precompute  $N$  groups of signing parameters.

`Beskar` offers improved computational efficiency over `Flamingo`, as it requires fewer participating decryptors— $k$  can be as low as 2—compared to a minimum of 64 in `Flamingo` [26]. Moreover, `Beskar` is more efficient than the maliciously-secure version of `MicroFedML` [27], which involves a linear number of operations, including signature verification, key agreement, secret sharing, and symmetric encryption/decryption, leading to higher computational costs. While `Beskar` incurs additional overhead in precomputing signing parameters and masks compared to `e-SeaFL`, this cost is offset by the increased efficiency in the subsequent aggregation phase, with minimal impact on the offline setup phase. Additionally, in contrast to the protocol by Bell et al. [7], which requires the setup phase to be repeated in every aggregation round due to the inability to reuse key material, `Beskar` avoids this repetitive overhead, further enhancing its overall efficiency.

2) *Aggregation Phase:* During the aggregation phase, `Beskar` further reduces computational overhead. The user only needs to perform  $k + 1$  summations and two optimized signature generation operations to achieve security against malicious adversaries. Notably, PRF computations are not required during aggregation since the masks are precomputed in the Setup phase. Given that  $k$  is much smaller than the number of participating users or decryptors, `Beskar` offers significantly higher efficiency compared to alternative protocols.

### B. Implementation

We implement `Beskar` with 2.5k lines of code, involving two components, namely the secure aggregation for gradient

updates and the model training with multiple DP protections. Our code is available at <https://github.com/kydahe/Beskar>.

**Secure Aggregation.** `Beskar` builds on `ABIDES` [44], a discrete event simulation framework commonly used in FL research [26], [27], enabling the simulation of multi-iterative aggregation protocols. We incorporated two post-quantum algorithms, i.e., the Kyber key encapsulation algorithm for negotiating shared secrets between clients and assisting nodes, and Dilithium signing algorithm for generating and verifying digital signatures during aggregation phase. These algorithms are implemented based on existing libraries [45], [46], with Dilithium modified to support signing with precomputation in C. For the Pseudo-Random Function (PRF) used in mask generation, we employed `ASCON` [47], known for its lightweight and efficient cryptography, making it suitable for resource-constrained client devices.

**Model Training with Multiple DP Protections.** We adopt the `Flower FL` framework [48] and integrate our DP protections using `PyTorch Opacus` [49]. In particular, `Opacus` determines the required noise multiplier through an iterative procedure that balances privacy and utility. It first converts the training configuration into a total number of steps (via the number of iterations and client fraction fits), then uses a DP accountant (e.g., Rényi differential privacy [50]) to assess  $\epsilon$  for a given noise multiplier. The algorithm locates an appropriate noise multiplier by expanding an upper bound until  $\epsilon$  is satisfied, followed by a binary search for finer precision. We implement two different DP methods—tailored to distinct threat models—alongside a baseline FL model without DP protection. Each model is trained five times, and the highest accuracy is reported, accounting for random variations in initialization and privacy mechanisms.

### C. Experimental Environment

The secure aggregation experiments were conducted on an `x86_64` Linux machine with AMD Ryzen Threadripper PRO 5965WX 24-Cores and 256 GB RAM. The models (with DP schemes) were trained using a `NVIDIA RTX 6000 Ada Generation GPU` on another `x86_64` Linux machine with `Intel(R) Xeon(R) w7-2475X` and 256 GB RAM.

### D. Experimental Setup and Evaluation Metrics

We systematically evaluated `Beskar` with respect to two key dimensions: efficiency and performance. Efficiency was measured by empirical time complexity during the setup and aggregation phases, along with corresponding bandwidth usage. Performance was assessed based on accuracy using five standard vision benchmarks that grow in difficulty and size: `MNIST` [51], `EMNIST` [52], `CIFAR-10` [53], `CIFAR-100` [53], and `CHMNIST` [54]. We pair each dataset with a representative architecture of ascending capacity: a lightweight MLP for `MNIST` and `EMNIST`, `ResNet-18` for `CIFAR` datasets, and a larger `AlexNet`-style network for `CHMNIST`. In the following, we describe the three empirical metrics employed in our experiments.

**Metrics for Efficiency:** We use two metrics, i.e., computation time and communication bandwidth, for efficiency measurement. Comparative analysis was conducted using `Flamingo`

TABLE II: Analytical performance of the secure aggregation of Beskar and its counterparts

Scheme	Setup Phase			Aggregate Phase		
	User	Assisting/Helper Nodes	Server	User	Assisting/Helper Nodes	Server
MicroFedML [27]	$1_{K_g} + 1_{S_{gn}} +  U' - 1 (1_{V_{fy}} + 1_{K_a} + 1_{ssh_l} + 1_{A_{Enc}} + 1_{A_{Dec}})$	N/A	$ U _{V_{fy}}$	$( w  + 1)_{ex} + 1_{S_{gn}} +  U - 1 (1_{su} + 1_{V_{fy}}) + 1_{su}$	N/A	$1_{R_{ssh_e}} +  w _{ex} + 1_{mu}$
Flamingo [26]	$ D' _{V_{fy}}$	$2_{Pl_l} + 3_{S_{gn}} + 2_{ssh_l} +  D (3 D'  + 1)_{V_{fy}} + ( D'  \cdot  D )_{mu}$	$( D'  \cdot  D )_{V_{fy}}$	$2_{PRG} +  D' _{(1_{ex} + 1_{Hash} + 1_{PRF} + 1_{su} + 1_{A_{Enc}} + 1_{S_{gn}} + 1_{SEnc})} + 1_{ssh_l}$	$1_{S_{gn}} +  U' _{S_{Dec}} + ( D  +  U' )_{V_{fy}}$	$ D _{Pl_l} +  U' _{(1_{su} + 1_{mu} + 1_{R_{ssh}})} + 2_{PRG} + ( U'  + 1)_{su}$
e-SeaFL [6]	$2_{K_g} + k_{K_a}$	$2_{K_g} + k_{K_a}$	$1_{K_g}$	$k_{PRF} +  k + 1 _{su} + 2_{S_{gn}}$	$ U' _{(1_{V_{fy}} + 1_{su} + 1_{PRF})} + 1_{S_{gn}}$	$( U'  +  k )_{(1_{V_{fy}} + 1_{su})}$
Beskar	$2_{K_g} + k_{Encp} + N_{Precomp} + (T \times k)_{PRF}$	$2_{K_g} +  U _{Decp} + (T \times  U )_{PRF} + N_{Precomp}$	$1_{K_g}$	$ k + 1 _{su} + 2_{PS_{gn}}$	$ U' _{(1_{V_{fy}} + 1_{su})} + 1_{PS_{gn}}$	$( U'  +  k )_{(1_{V_{fy}} + 1_{su})}$

$K_g$ ,  $S_{gn}$ ,  $V_{fy}$ , and  $Hash$  denote key generation, signature generation, verification, and hash function, respectively.  $K_a$  denotes the shared key computation in the key agreement protocol.  $|D|$  and  $|D'|$  denote the number of all decryptors and the threshold of participating decryptors [26], respectively.  $Pl_l$  denotes polynomial interpolation of length  $L$ .  $ssh_l$ ,  $R_{ssh}$ , and  $R_{ssh_e}$  denote secret sharing operation to  $l$  shares, share reconstruction, and share reconstruction in the exponent, respectively.  $mu$ ,  $su$  and  $ex$  denote multiplication, summation and exponentiation operations, respectively.  $PRG$  and  $PRF$  denote pseudorandom generators and pseudorandom functions, respectively.  $A_{Enc}$ ,  $A_{Dec}$ ,  $S_{Enc}$ , and  $S_{Dec}$  denote asymmetric encryption, decryption, symmetric encryption, and decryption, respectively.  $|w|$  denotes the number of model parameters.  $k$  is the set of assisting nodes in Beskar.

[26], PQSA [10], e-SeaFL [6], and MicroFedML [27], where MicroFedML includes two variants, MicroFedML1 and MicroFedML2.

- **Computation Time.** To assess the efficiency of our secure aggregation method with post-quantum protection, we measured the time required for computational operations such as key encapsulation, mask generation, signing, and verification. The number of clients ( $n$ ) was varied from 200 to 1000, with three assisting nodes ( $k = 3$ ) and one aggregation server. We also tested performance with a gradient vector size of 16,000 to simulate high-dimensional data.
- **Communication Bandwidth.** We recorded the bandwidth, i.e., total message size exchanged between clients, assisting nodes, and the server, to evaluate communication costs. The reason why we use message size instead of transmission time is because message transmission time highly depends on current network conditions. Variations in hardware and network quality can greatly affect latency.

**Metric for Performance:** We assessed the performance of our DP methods by evaluating the *noise multiplier* and the *accuracy* of the final models. Client data allocation adheres to standard protocols [35], [26]. Unless specified otherwise, training proceeds with 5 clients, 50 communication rounds, and a client-sampling fraction of 1.0—in other words, each client contributes to every round. Pilot runs showed that fifty rounds are already enough for the models to converge under DP noise; extending training beyond that point improves accuracy only marginally while increasing computation. Using a sampling fraction of one eliminates variability due to partial participation, allowing us to isolate the sole effect of injected noise. Starting from this baseline, we varied the privacy budget  $\epsilon$  values across  $\{5, 10, 15, 20\}$  to evaluate the privacy–utility trade-off. We then fixed  $\epsilon$  at 10 and varied the two factors independently. First, we shortened training to as few as a single iteration and gradually extended it through  $\{1, 5, 10, 15, 20\}$  to reveal how DP noise hinders convergence when updates are scarce. Second, with 10 clients in total, we adjusted sampling fraction over  $\{0.3, 0.5, 0.7, 0.9\}$  so that different number of clients participate in each iteration, to evaluate the impact of the size of the actively participating subset on both noise levels and final accuracy.

## VI. EVALUATION RESULTS

In this section, we present the experimental results evaluating the efficiency and performance of Beskar. Our findings address the following key research questions:

- **Section VI-A (Efficiency Measurement):** How efficient is post-quantum enhanced secure aggregation in FL? What gains in efficiency does Beskar achieve through the use of pre-computation?
- **Section VI-C (Model Performance Measurement):** How does Beskar’s differential privacy module perform under different threat models? What is the impact of DP-enhanced methods on FL performance across different training datasets?

### A. Efficiency Measurement

1) **Computational Cost:** We systematically compare the computational cost of Beskar in the Setup phase and Aggregation phase with four state-of-the-art approaches, i.e., Flamingo, PQSA, e-SeaFL, and MicroFedML.

**Setup Phase.** Fig. 3 shows the computation costs of each method during the Setup phase. Beskar stands out for its nearly constant computation time on the server (Fig. 3 (a)) and client (Fig 3 (b)) side, even as the number of clients increases. This is because the operations Beskar performs on the client side are primarily dependent on the number of assisting nodes, which typically remains constant in general FL settings, and the fact that server-side operations have a constant computational cost. In contrast, the computation time for assisting nodes increases as the number of clients grows (Fig. 3 (c)). This increase results from tasks performed by assisting nodes, such as shared secret generation, which become more intensive as the number of clients rises.

Our results also show that, in the setup phase, Beskar incurs higher computation times compared to Flamingo, e-SeaFL and MicroFedML across all entities. The reason is that Beskar incorporates an additional precomputation step in the setup phase, where several digital signature parameters are precomputed for subsequent runtime signing (see Section IV-E). PQSA is excluded from this comparison as it only distributes several public parameters without performing significant computations. It is also worth noting that Flamingo does not involve any client-side operations, as it relies on a trusted third party to generate and distribute signing keys for each entity.

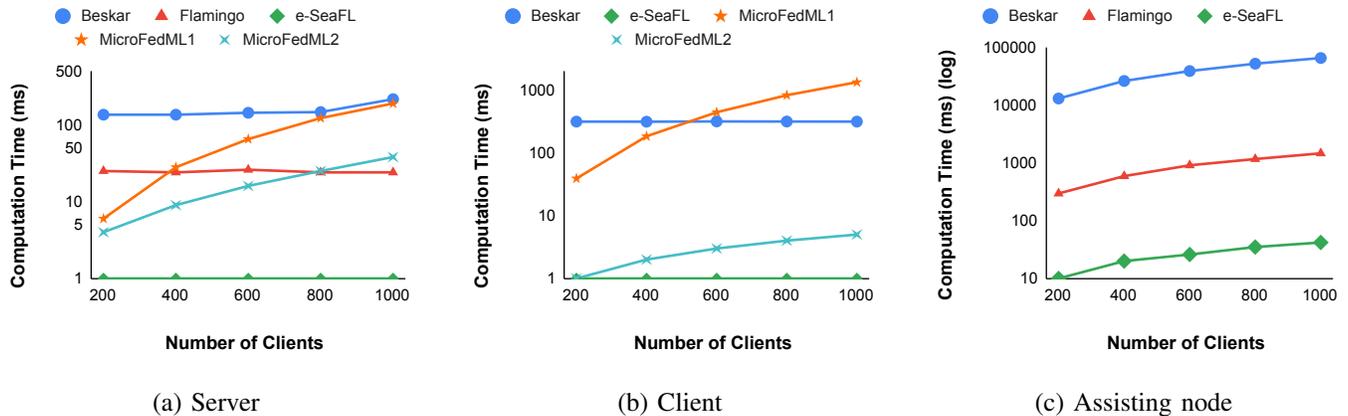


Fig. 3: Computation Time of the Setup phase (The dimension of the weight list is set to 16K.)

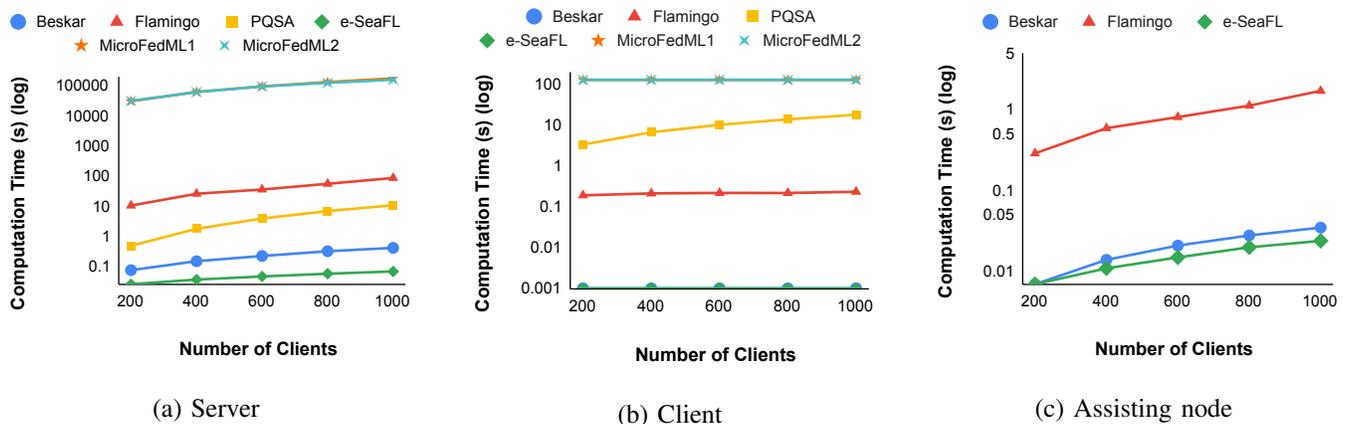


Fig. 4: Computation Time of the Aggregation phase (The dimension of the weight list is set to 16K.)

**Aggregation Phase.** Fig. 4 shows the computation time of *Beskar* in the aggregation phase compared with that of *Flamingo*, *PQSA*, *e-SeaFL*, and *MicroFedML*. A key advantage of *Beskar* is that client-side computation time remains constant (Fig. 4 (b)), even as the number of clients grows. This stability minimizes client-side resource demands, making *Beskar* particularly suitable for settings with resource-constrained FL clients. Moreover, the computation time increases for the server (Fig. 4 (a)) and assisting nodes (Fig. 4 (c)) as the number of clients grows. This increase is primarily due to the need for assisting nodes and the server to verify the signatures of messages sent by clients, making computation time proportional to the number of clients.

The results demonstrate that all entities in *Beskar* incur minimal computational overhead, with computation times significantly lower than those of *Flamingo*, *PQSA*, and *MicroFedML*, with the exception of *e-SeaFL*. Specifically, *Beskar* has a similar computation time to *e-SeaFL* on the client side, but a higher computation time on the assisting nodes and server. This difference arises because both *Beskar* and *e-SeaFL* clients perform similar operations, including gradient masking and a few signing operations. However, the assisting nodes and server in *Beskar* must conduct a large number of signature verifications, which scale with the

number of clients. The post-quantum signing algorithm (i.e., Dilithium) used by *Beskar* is originally slower than the traditional algorithm (i.e., ECDSA) used in *e-SeaFL*, leading to higher computation times as the number of clients increases.

*Beskar* outperforms *Flamingo*, *PQSA*, and *MicroFedML* in terms of computation time across clients, assisting nodes, and the server. *Flamingo* requires additional steps to remove masks for dropout clients, necessitating extra information (e.g., shares of secrets used to generate the masks) from clients and decryptors to allow the server to recover the gradient vectors of those offline clients. *PQSA* uses SHPRG-based encryption for gradient masking, which involves an extra step between clients and the server to decrypt and recover the masks and calculate the aggregated result. In *MicroFedML*, each client must additionally compute  $H(k)^{X_i}$  and  $H(k)^{R_i}$ , and the server has to recover the gradient vector by calculating the discrete logarithm of  $H(k)^{X_i} - H(k)^{R_i}$ . Such discrete logarithm calculation is computationally very intensive, especially as the vector dimensions increase.

Overall, compared with the existing approaches, *Beskar* employs a straightforward strategy for dropout clients by ignoring the masked vectors of dropped clients and utilizes a post-quantum algorithm with precomputation for optimization. This approach reduces the number of messages, eliminates un-

necessary calculations, and significantly reduces time overhead while maintaining quantum security.

2) *Communication Cost*: We analyze the communication cost, specifically the outbound bandwidth, of *Beskar* during the setup and aggregation phases in comparison to existing approaches.

**Setup Phase.** Fig. 5 presents the outbound bandwidth costs of all entities in *Beskar* during the setup phase. In general, the outbound bandwidth of *Beskar* remains constant on the server (Fig. 5 (a)) and client (Fig. 5 (b)) sides while increases on the assisting nodes (Fig. 5 (c)) as the number of clients grows. This is due to the fact that during the setup phase, the server only needs to broadcast its public key for signing, and each client sends the public keys for key exchange and signing to the assisting nodes, whose number keeps constant in our experimental setting. However, each assisting node must send these public keys to all clients, making its outbound bandwidth dependent on the number of clients.

*Beskar* incurs less outbound bandwidth than *Fleming* and *MicroFedML* but it is generally comparable to *e-SeaFL* across all three entities. Specifically, during the setup phase, *Beskar* and *e-SeaFL* perform the same public key exchange operations, resulting in similar outbound bandwidths, both of which depend on the size of the public keys. In contrast, in *Fleming*, the server and decryptors participate in DKG protocols, where their outbound bandwidth depends on the number and size of the messages in DKG protocols and is proportional to the number of decryptors, but constant relative to the number of clients. This results in greater outbound bandwidth for the server and decryptors in *Fleming* compared to *Beskar*, which only involves public key exchanges. As for *MicroFedML*, the server has to forward all the communications, i.e., the public keys and secret shares generated by Shamir’s secret sharing scheme, from one client to each of the others, making its bandwidth proportional to the number of clients and higher than that of *Beskar*. Each client in *MicroFedML* should send its public key and  $n-1$  mask shares to all other clients, making its bandwidth increase proportional to the number of clients, which is significantly higher than that of *Beskar*.

Additionally, *PQSA* and the clients of *Fleming* are excluded from this comparison for the same reasons previously mentioned.

**Aggregation Phase.** We present the outbound bandwidth cost of all entities in *Beskar* during the runtime aggregation phase in Fig. 6. Generally, the outbound bandwidth of *Beskar* remains constant across all three entities, i.e., the server (Fig. 6 (a)), the clients (Fig. 6 (b)), and the assisting nodes (Fig. 6 (c)) as the number of clients increases. This stability occurs because the outbound messages for each entity do not depend on the number of clients. Specifically, each client’s outbound messages depend only on the number of assisting nodes and the server, which remain constant in our experimental setup. Assisting nodes send a single message (containing the sum of partial masks) to the server, and the server broadcasts the aggregated results, keeping their outbound bandwidth unchanged.

TABLE III: Results of Precomputation Improvement (Computation Time in the Aggregation Phase) (ms): P - Precomputation

N	Client		Server		Assisting Node	
	w/o P	w/ P	w/o P	w/ P	w/o P	w/ P
200	135	1	77	76	8767	7
400	134	1	163	151	17703	14
600	135	1	227	225	27008	21
800	134	1	323	302	35281	28
1000	134	2	417	377	43144	35

When compared with existing approaches, *Beskar* demonstrates lower outbound bandwidth costs across all entities compared to *Fleming*, *PQSA*, *e-SeaFL*, and *MicroFedML*. Specifically, in *Fleming*, the server is involved in forwarding the messages between the clients and decryptors, leading to a higher outbound bandwidth that is roughly proportional to the number of clients. Clients have to send secret shares to decryptors and neighbors (i.e., clients in the same group), in addition to masked updates to the server, which increases their outbound bandwidth proportional to the number of decryptors and clients (i.e., neighbors), which is significantly higher than *Beskar*. Decryptors also transmit decrypted secret shares of each client to the server, resulting in higher bandwidth proportional to the number of clients, compared to *Beskar*. For *PQSA*, the server performs an additional communication step with clients to decrypt and recover the masks, resulting in a higher bandwidth that is proportional to the number of clients. Similarly, clients have to additionally send the decrypted secrets, except for the masked updates, leading to a higher bandwidth, which is proportional to the number of clients. While *e-SeaFL* performs the similar operations to *Beskar* across all entities, the difference in outbound bandwidth is primarily due to the size of each message, especially the size of signatures. As for *MicroFedML*, similar to *Fleming*, the server has to forward the signature of each client to other clients, incurring a higher outbound bandwidth than *Beskar*, which is proportional to the number of clients. Moreover, clients in *MicroFedML* send masks of online clients along with the masked updates to the server, resulting in bandwidth higher than in *Beskar*.

3) *Precomputation Improvement*: To assess the impact of precomputation strategies on *Beskar*, we conducted a comparative experiment implementing *Beskar* without precomputation in the Dilithium signing algorithms and mask generation. The results, shown in Table III, indicate that precomputation significantly enhances efficiency for both assisting nodes and clients as the number of clients increases.

Without precomputation, an assisting node must perform additional mask generation operations for  $n$  clients and one signing operation, while a client must perform additional mask generation for  $k$  assisting nodes and  $k+1$  signing operations. Consequently, precomputation notably reduces the computational burden on these entities.

On the server side, the benefits of precomputation are less pronounced. The reason is that the server performs relatively fewer computational operations, limited to only one signing

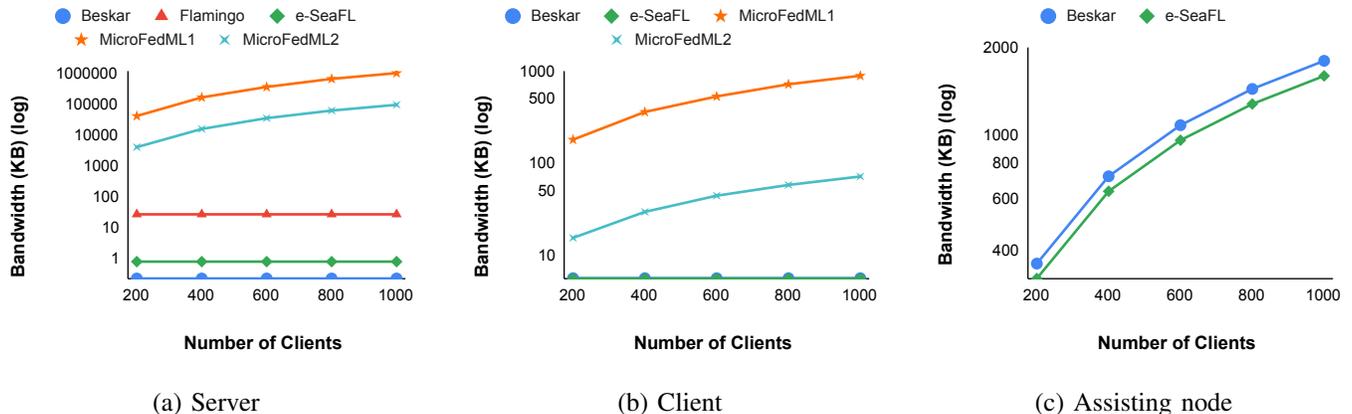


Fig. 5: Bandwidth in the Setup Phase (The dimension of the weight list is set to 16K.)

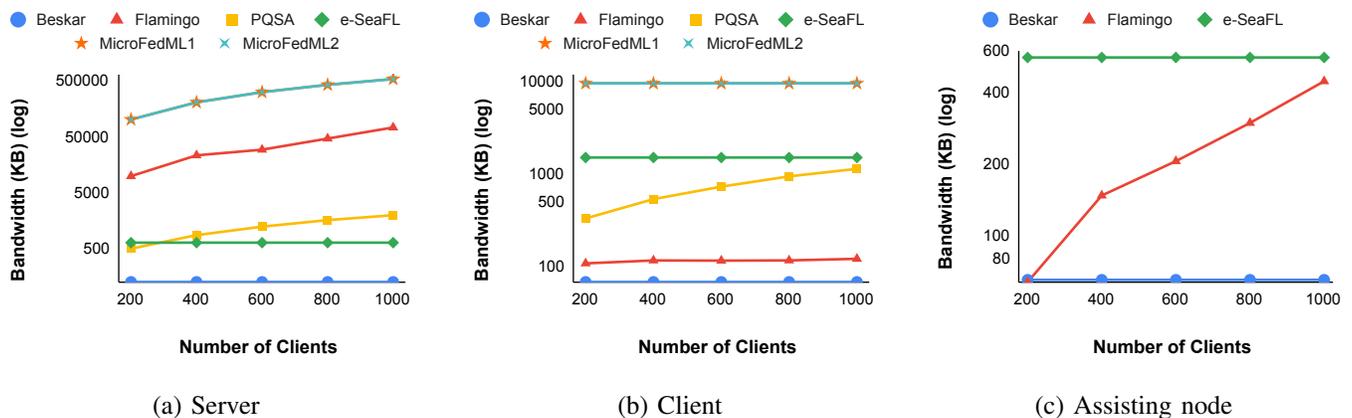


Fig. 6: Bandwidth in the Aggregation phase (The dimension of the weight list is set to 16K.)

operation, compared to the assisting nodes and clients.

### B. Feasibility of Employing Assisting Nodes

Following our computational and communication evaluation of Beskar and its counterpart, we emphasize that a assisting node incurs only minimal additional overhead—specifically, about 6 ms more computation time than a regular client (with a total of 200 clients), and approximately 2 KB less in communication. Given this lightweight cost, a simple distributed ledger-based method can be used to select a set of clients in each iteration to serve as assisting nodes. This is not a strong assumption since sharing metadata about the model and training parameters is common in FL. Deploying assisting nodes significantly reduces the trust assumption because the privacy of the protocol holds as long as at least one assisting node is honest. At the same time, it minimizes the overhead on participating users and encourages broader participation in the learning process.

### C. Model Performance

Table IV presents accuracy results on MNIST, EMNIST, CIFAR-10, CIFAR-100, and CHMNIST under Non-Differential Privacy (NDP), Central Differential Privacy

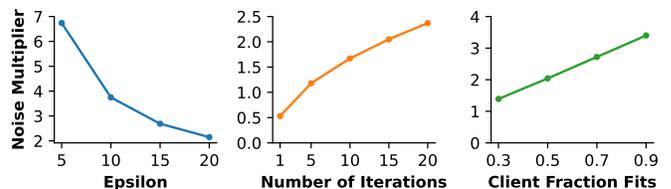


Fig. 7: Effect of Epsilon, Iterations, and Clients on Noise.

(CDP), and Local Differential Privacy (LDP) across varying privacy budgets  $\epsilon$ , training iterations, and the fraction of participating clients (fraction fit). Fig. 7 shows the corresponding noise multipliers under these configurations. Below, we discuss how  $\epsilon$ , the number of training iterations, and the fraction fit affect both model accuracy and noise.

1) *Effect of Privacy Budget ( $\epsilon$ ):* We first examine how varying privacy budget  $\epsilon$  impacts both the noise multiplier and the model accuracy. As expected, NDP, which injects no noise, yields the highest accuracy across all datasets. Once privacy is enforced, both CDP and LDP see reduced accuracy relative to NDP, with CDP generally outperforming LDP.

**Noise Multiplier vs.  $\epsilon$ .** Fig. 7 underscores the inverse relationship between  $\epsilon$  and the noise multiplier. Smaller  $\epsilon$  values offer

TABLE IV: Results of Model Performance under Different Differential Privacy Parameters.

$\epsilon$	#Iters	#Cli	MNIST			EMNIST			CIFAR-10			CIFAR-100			CHMNIST		
			NDP	CDP	LDP	NDP	CDP	LDP	NDP	CDP	LDP	NDP	CDP	LDP	NDP	CDP	LDP
20	50	5/5	0.99	0.98	0.86	0.99	0.98	0.87	0.83	0.81	0.45	0.76	0.63	0.02	0.84	0.83	0.27
15	50	5/5	0.99	0.97	0.84	0.99	0.98	0.85	0.83	0.80	0.23	0.76	0.56	0.02	0.84	0.81	0.26
10	50	5/5	0.99	0.87	0.78	0.99	0.90	0.82	0.83	0.77	0.20	0.76	0.23	0.02	0.84	0.76	0.23
5	50	5/5	0.99	0.77	0.71	0.99	0.80	0.77	0.83	0.57	0.17	0.76	0.02	0.01	0.84	0.52	0.22
10	1	5/5	0.95	0.09	0.09	0.94	0.10	0.09	0.38	0.28	0.10	0.02	0.02	0.01	0.40	0.42	0.11
10	5	5/5	0.96	0.54	0.53	0.94	0.41	0.71	0.68	0.60	0.20	0.64	0.18	0.01	0.64	0.65	0.13
10	10	5/5	0.98	0.91	0.72	0.95	0.91	0.77	0.78	0.68	0.20	0.73	0.25	0.02	0.73	0.68	0.23
10	15	5/5	0.98	0.92	0.77	0.97	0.93	0.80	0.78	0.72	0.10	0.75	0.30	0.02	0.79	0.71	0.26
10	20	5/5	0.98	0.93	0.79	0.98	0.94	0.83	0.79	0.73	0.24	0.76	0.33	0.02	0.81	0.72	0.27
10	50	3/10	0.99	0.48	0.78	0.99	0.56	0.82	0.83	0.73	0.23	0.72	0.43	0.02	0.84	0.72	0.22
10	50	5/10	0.99	0.87	0.78	0.99	0.90	0.82	0.83	0.77	0.20	0.74	0.44	0.02	0.84	0.76	0.23
10	50	7/10	0.99	0.95	0.77	0.99	0.96	0.81	0.81	0.79	0.15	0.75	0.45	0.02	0.84	0.81	0.27
10	50	9/10	0.99	0.97	0.75	0.99	0.97	0.80	0.80	0.79	0.13	0.75	0.45	0.01	0.84	0.81	0.27

stronger privacy guarantees but require to increase the noise multiplier, which injects more noise at each training iteration. This additional noise hampers the model’s ability to learn, explaining the sharper accuracy decline observed for lower  $\epsilon$ .

**Model Accuracy vs.  $\epsilon$ .** Table IV shows that accuracy under both CDP and LDP improves as  $\epsilon$  increases. At a relatively high privacy budget (e.g.,  $\epsilon$ ), the noise multiplier is moderate, and the MNIST model under CDP achieves an accuracy of 0.98—close to the NDP baseline of 0.99—while LDP reaches 0.86. However, tightening privacy (e.g.,  $\epsilon = 5$ ) forces a much larger noise multiplier, causing significant accuracy drops and widening the gap between the DP methods and NDP.

2) *Sensitivity Analysis:* We next explore how other key hyperparameters—the number of training iterations and the fraction fit of clients—further influence the trade-off between model accuracy and noise levels. For these experiments, we fix  $\epsilon$  while varying one parameter at a time.

**Number of Training Iterations.** Increasing the number of training iterations substantially improves model convergence, even under noisy conditions. When training for only one iteration under LDP on MNIST with  $\epsilon = 10$ , accuracy can be just under 10%. In contrast, extending the training to 50 iterations under the same  $\epsilon$  raises accuracy to above 70%. This indicates that additional iterations provide the model more opportunity to learn from the data, thereby offsetting much of the performance degradation caused by noise. Although longer training may require additional noise injections (i.e., larger noise multiplier), the results show that the convergence gains generally outweigh the potential downside of increased noise, particularly when training has not yet converged.

**Fraction fit of Clients.** The fraction of clients selected to participate in each training iteration—rather than merely the total pool of clients—strongly influences both the noise multiplier and model accuracy. Under CDP, noise is injected centrally after aggregating client updates. When a larger fraction of the total clients participate, the training process benefits from more data per iteration, typically boosting accuracy. However, this increase in participating clients can also raise the noise multiplier, since a larger collective update requires proportionally more noise to preserve the privacy budget. Table IV shows

that as this fraction increases, more data is included in each iteration, which improves convergence and often offsets any added noise, ultimately leading to higher accuracy for CDP. By contrast, in LDP, each client adds noise directly to its own data or gradients. Hence, a larger fraction of participating clients produces a greater cumulative amount of noise in the aggregated update, which can reduce accuracy gains from using more data. As a result, we observed in Table IV, for LDP, larger client fractions can lead to lower accuracy.

In summary, both central differential privacy (CDP) and local differential privacy (LDP) incur some loss of accuracy relative to the non-private baseline (NDP); however, for tasks of moderate complexity this reduction remains modest, allowing both schemes to provide competitive performance while markedly strengthening data protection. Therefore, one can steer the privacy–utility trade-off by adjusting the privacy budget  $\epsilon$ , the number of training rounds, and the fraction of participating clients, thereby meeting application-specific privacy requirements without incurring unnecessary accuracy loss.

On more demanding benchmarks such as CIFAR-100, the strong noise injected under LDP hampers fine-grained feature learning and produces a noticeably sharper accuracy decline. Although recent study [35] reports substantially higher CIFAR-100 accuracy with LDP, we took a closer examination which reveals that the reported gains hinge on non-standard model architectures, and the publicly released code, when adapted to our standardized experimental setup, fails to reproduce the claimed performance.

## VII. RELATED WORK

Research on FL’s security and privacy focuses on two main streams: addressing attacks targeting the privacy of the training data ([55], [4], [7]) and preventing client-side attacks undermining the model’s reliability ([56], [57]). Privacy-preserving methods can be categorized further into those that protect privacy during training and those that protect privacy post-training. Techniques safeguarding privacy during training focus on protecting user updates during the training phase [58].

They often leverage multi-party computation (MPC) or homomorphic encryption to allow the central server to compute the model in each iteration without accessing individual user updates. However, the direct application of these methods often fails to efficiently account for user dropouts, an inherent challenge in FL, where mobile participants unexpectedly exit the training process due to factors such as low battery or unstable network connections. A series of follow-up works have focused on designing more resilient, secure aggregation methods capable of handling user dropouts [55], [59], [26].

Post-training privacy mechanisms rely on DP to address the deployed model’s privacy leakage. These methods often focus on computing a carefully calibrated noise to be injected either at the user updates or the intermediate model to prevent the leakage of sensitive information from the deployed model.

The other stream of research focuses on ensuring the reliability of the final model. Since the global model is derived from user updates, an adversary can compromise its reliability by injecting malicious updates. To mitigate these attacks, various input validation mechanisms have been proposed with the aim of ensuring that only legitimate updates are used to compute the final model. This can be done by leveraging various methods such as enforcing norm bounds [60], the use of trusted hardware [61], [62], etc.

While existing approaches typically treat secure aggregation and DP as separate solutions, *Beskar* is the first to integrate both within a unified framework guided by a comprehensive threat model for FL. To the best of our knowledge, it is also the first protocol to address the emerging challenge of post-quantum security while preserving practical efficiency and model performance, particularly in resource-constrained mobile environments.

### VIII. CONCLUSION

In this paper, we propose *Beskar*, a secure framework to enhance privacy protection in FL by integrating post-quantum secure aggregation with differential privacy. We introduce novel precomputation techniques to optimize the efficiency of post-quantum signing algorithms and mask calculation for secure aggregation. Additionally, we present a comprehensive threat model for FL, addressing various adversarial scenarios and systematically applying differential privacy strategies to offer tailored defenses against diverse privacy threats. Through detailed efficiency measurements and performance analysis, we demonstrate that *Beskar* achieves a balanced trade-off between security, computational efficiency, and model accuracy. Our contributions represent a significant advancement in making FL both secure and practical in a post-quantum world, addressing contemporary privacy challenges while adhering to regulatory requirements.

### ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to the editor and reviewers for their guidance, insightful comments, and constructive feedback, which have significantly improved the quality of this paper. This work reported in this paper has been supported by the National Science Foundation

(NSF) under Grant No. 2112471, and by the NSF–SNSF joint program under Grant No. ECCS 2444615.

### REFERENCES

- [1] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson *et al.*, “Extracting training data from large language models,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2633–2650.
- [2] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership inference attacks against machine learning models,” in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 3–18.
- [3] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, “The secret sharer: Evaluating and testing unintended memorization in neural networks,” in *28th USENIX security symposium (USENIX security 19)*, 2019, pp. 267–284.
- [4] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *Computer and Communications Security*, 2017, pp. 1175–1191.
- [5] M. Rathee, C. Shen, S. Wagh, and R. A. Popa, “ELSA: Secure aggregation for federated learning with malicious actors,” in *IEEE symposium on security and privacy (SP)*, 2023, pp. 1961–1979.
- [6] R. Behnia, M. Ebrahimi, A. Riasi, S. S. Chow, B. Padmanabhan, and T. Hoang, “Efficient secure aggregation for privacy-preserving federated machine learning,” *arXiv preprint arXiv:2304.03841*, 2023.
- [7] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, “Secure single-server aggregation with (poly) logarithmic overhead,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1253–1269.
- [8] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, “A hybrid approach to privacy-preserving federated learning,” in *Proceedings of the 12th ACM workshop on artificial intelligence and security*, 2019, pp. 1–11.
- [9] T. Elahi, G. Danezis, and I. Goldberg, “Privex: Private collection of traffic statistics for anonymous communication networks,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 1068–1079.
- [10] S. Yang, Y. Chen, S. Tu, and Z. Yang, “A post-quantum secure aggregation for federated learning,” in *Proceedings of the 2022 12th International Conference on Communication and Network Security*, 2022, pp. 117–124.
- [11] L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. A. Perlner, and D. Smith-Tone, *NIST Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology . . . , 2016, vol. 12.
- [12] “Nist post-quantum cryptography standardization,” <https://csrc.nist.gov/projects/post-quantum-cryptography>, 2024.
- [13] “Report on post-quantum cryptography,” [https://www.whitehouse.gov/wp-content/uploads/2024/07/REF\\_PQC-Report\\_FINAL\\_Send.pdf](https://www.whitehouse.gov/wp-content/uploads/2024/07/REF_PQC-Report_FINAL_Send.pdf), 2024.
- [14] “Nist selected post-quantum cryptography algorithms 2022,” <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>, 2024.
- [15] R. Behnia, M. O. Ozmen, A. A. Yavuz, and M. Rosulek, “TACHYON: fast signatures from compact knapsack,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2018, pp. 1855–1867.
- [16] R. Behnia and A. A. Yavuz, “Towards practical post-quantum signatures for resource-limited internet of things,” in *Proceedings of the 37th Annual Computer Security Applications Conference*, 2021, p. 119–130.
- [17] C. Dwork, “Differential privacy,” in *International colloquium on automata, languages, and programming*. Springer, 2006, pp. 1–12.
- [18] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep learning with differential privacy,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24–28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 308–318. [Online]. Available: <https://doi.org/10.1145/2976749.2978318>
- [19] Z. Chen, G. Vasilakis, K. Murdock, E. Dean, D. Oswald, and F. D. Garcia, “{VoltPillager}: Hardware-based fault injection attacks against intel {SGX} enclaves using the {SVID} voltage scaling interface,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 699–716.

- [20] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 991–1008.
- [21] M. A. P. Chamikara, D. Liu, S. Camtepe, S. Nepal, M. Grobler, P. Bertok, and I. Khalil, "Local differential privacy for federated learning," *arXiv preprint arXiv:2202.06053*, 2022.
- [22] S. Truex, L. Liu, K.-H. Chow, M. E. Gursoy, and W. Wei, "Ldp-fed: Federated learning with local differential privacy," in *Proceedings of the third ACM international workshop on edge systems, analytics and networking*, 2020, pp. 61–66.
- [23] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and trends® in machine learning*, vol. 14, no. 1–2, pp. 1–210, 2021.
- [24] B. Balle, P. Kairouz, B. McMahan, O. Thakkar, and A. Guha Thakurta, "Privacy amplification via random check-ins," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4623–4634, 2020.
- [25] S. Ramaswamy, O. Thakkar, R. Mathews, G. Andrew, H. B. McMahan, and F. Beaufays, "Training production language models without memorizing user data," *arXiv preprint arXiv:2009.10031*, 2020.
- [26] Y. Ma, J. Woods, S. Angel, A. Polychroniadou, and T. Rabin, "Flamingo: Multi-round single-server secure aggregation with applications to private federated learning," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 477–496.
- [27] Y. Guo, A. Polychroniadou, E. Shi, D. Byrd, and T. Balch, "Microfedml: Privacy preserving federated learning for small weights," in *The 4th Privacy-Preserving Machine Learning Workshop 2022*, 2022, p. available: <https://ia.cr/2022/714>.
- [28] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 238–268, 2018.
- [29] J. Bos, L. Ducas, E. Kiltz, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber: a cca-secure module-lattice-based kem," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 353–367.
- [30] A. A. Yavuz, "Eta: efficient and tiny authentication for heterogeneous wireless systems," in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, 2013, pp. 67–72.
- [31] R. Behnia, M. O. Ozmen, A. A. Yavuz, and M. Rosulek, "Tachyon: Fast signatures from compact knapsack," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 1855–1867.
- [32] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehle, "Crystals – dilithium: Digital signatures from module lattices," *Cryptology ePrint Archive*, Report 2017/633, 2017, <http://eprint.iacr.org/2017/633>.
- [33] L. Sun, J. Qian, and X. Chen, "Ldp-fl: Practical private aggregation in federated learning with local differential privacy," *arXiv preprint arXiv:2007.15789*, 2020.
- [34] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.
- [35] Y. Yang, B. Hui, H. Yuan, N. Gong, and Y. Cao, "{PrivateFL}: Accurate, differentially private federated learning via personalized data transformation," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1595–1612.
- [36] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly)logarithmic overhead," in *Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1253–1269. [Online]. Available: <https://doi.org/10.1145/3372297.3417885>
- [37] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.
- [38] C. Paquin, D. Stebila, and G. Tamvada, "Benchmarking post-quantum cryptography in tls," in *Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11*. Springer, 2020, pp. 72–91.
- [39] J. Hayes, B. Balle, and S. Mahloujifar, "Bounding training data reconstruction in dp-sgd," *Advances in neural information processing systems*, vol. 36, pp. 78 696–78 722, 2023.
- [40] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, nov 1979. [Online]. Available: <https://doi.org/10.1145/359168.359176>
- [41] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O’Hearn, "SpHincs: practical stateless hash-based signatures," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 368–397.
- [42] P. Li, T. Chen, and J. Liu, "Enhancing quantum security over federated learning via post-quantum cryptography," *arXiv preprint arXiv:2409.04637*, 2024.
- [43] R. Behnia, Y. Chen, and D. Masny, "On removing rejection conditions in practical lattice-based signatures," in *Post-Quantum Cryptography: 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20–22, 2021, Proceedings 12*. Springer, 2021, pp. 380–398.
- [44] D. Byrd, M. Hybinette, and T. H. Balch, "Abides: Towards high-fidelity market simulation for ai research," *arXiv preprint arXiv:1904.12066*, 2019.
- [45] "Crystals-kyber python implementation," <https://github.com/GiacomoPope/kyber-py>, 2024.
- [46] "Crystals-dilithium c implementation," [https://github.com/pq-crystals/dilithium\\_c](https://github.com/pq-crystals/dilithium_c), 2024.
- [47] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1. 2: Lightweight authenticated encryption and hashing," *Journal of Cryptology*, vol. 34, pp. 1–42, 2021.
- [48] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão *et al.*, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.
- [49] "Pytorch opacus," <https://github.com/pytorch/opacus>, 2024.
- [50] I. Mironov, "Rényi differential privacy," in *2017 IEEE 30th computer security foundations symposium (CSF)*. IEEE, 2017, pp. 263–275.
- [51] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
- [52] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.
- [53] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
- [54] J. N. Kather, C.-A. Weis, F. Bianconi, S. M. Melchers, L. R. Schad, T. Gaiser, A. Marx, and F. G. Zöllner, "Multi-class texture analysis in colorectal cancer histology," *Scientific reports*, vol. 6, no. 1, pp. 1–11, 2016.
- [55] Y. Zheng, S. Lai, Y. Liu, X. Yuan, X. Yi, and C. Wang, "Aggregation service for federated learning: An efficient, secure, and more resilient realization," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 988–1001, 2022.
- [56] J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun, "{ACORN}: input validation for secure aggregation," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4805–4822.
- [57] A. Roy Chowdhury, C. Guo, S. Jha, and L. van der Maaten, "Eiffel: Ensuring integrity for federated learning," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2535–2549.
- [58] X. Wang, X. Liu, and X. Yi, "Model extraction attack on mpc hardened vertical federated learning," in *International Conference on Provable Security*. Springer, 2024, pp. 63–82.
- [59] R. Behnia, A. Riasi, R. Ebrahimi, S. S. M. Chow, B. Padmanabhan, and T. Hoang, "Efficient secure aggregation for privacy-preserving federated machine learning," in *2024 Annual Computer Security Applications Conference (ACSAC)*, 2024, pp. 778–793.
- [60] H. Lycklama, L. Burkhalter, A. Viand, N. Küchler, and A. Hithnawi, "Rofl: Robustness of secure federated learning," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 453–476.
- [61] L. Zhao, J. Jiang, B. Feng, Q. Wang, C. Shen, and Q. Li, "Sear: Secure and efficient aggregation for byzantine-robust federated learning," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3329–3342, 2021.
- [62] X. Guo, Z. Liu, J. Li, J. Gao, B. Hou, C. Dong, and T. Baker, "Verifi: Communication-efficient and fast verifiable aggregation for federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1736–1751, 2020.

APPENDIX A  
DETAILED SIGNING ALGORITHMS

---

**Algorithm 4** Dilithium

 $(\text{sk}_{\mathcal{E}}, \text{pk}_{\mathcal{E}}) \leftarrow \text{KeyGen}(1^\kappa)$ 


---

```

1:  $\mathbf{A} \leftarrow \mathbb{R}_q^{k \times \ell}$ 
2:  $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow \mathbb{S}_\eta^\ell \times \mathbb{S}_\eta^k$ 
3:  $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ 
4: return  $(\text{pk} = (\mathbf{A}, \mathbf{t}), \text{sk} = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2))$ 

```

---

 $\sigma \leftarrow \text{Sign}(\text{sk}_{\Pi}, m)$ 


---

```

1:  $\mathbf{z} := \perp$ 
2: while  $\mathbf{z} = \perp$  do
3:    $\mathbf{y} \leftarrow \mathbb{S}_{\gamma_1 - 1}^\ell$ 
4:    $\mathbf{w}_1 := \text{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$ 
5:    $c \in \mathbb{B}_{60} := \mathcal{H}(m \parallel \mathbf{w}_1)$ 
6:    $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$ 
7:   if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ 
8:     or  $\|\text{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta,$ 
9:     then  $\mathbf{z} := \perp$ 
10: return  $\sigma = (\mathbf{z}, c)$ 

```

---

 $\{0, 1\} \leftarrow \text{Verify}(\text{pk}_{\Pi}, m, \sigma)$ 


---

```

1:  $\mathbf{w}'_1 := \text{HighBits}(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)$ 
2: if  $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$  and  $c = \mathcal{H}(m \parallel \mathbf{w}'_1)$  then return 1
3: else return 0

```

---

The optimized signing algorithms with precomputation (Algorithm 5) build upon the Dilithium algorithms [32] (Algorithm 4). The precomputation process aims to enhance signing efficiency by precomputing and storing parameters related to the private key  $\text{sk}_{\Pi}$ , reducing the need for new randomness and complex lattice computations during each runtime signing operation. The procedure iteratively computes values, including a matrix  $\mathbf{A}$  (derived from a seed  $\rho$ ) and a masking vector  $\mathbf{y}$ , generated through the ExpandMask function. This function combines the private key and random inputs to expand values. The computed sets are stored in  $\mathcal{LS}[\text{sk}_{\Pi}]$  until  $N$  precomputed groups are achieved.

During signing, the precomputed values in  $\mathcal{LS}[\text{sk}_{\Pi}]$  are used to generate signatures more efficiently. The algorithm iterates through  $\mathcal{LS}[\text{sk}_{\Pi}]$ , checking for compliance with validity conditions defined by threshold parameters  $(\gamma_1, \gamma_2)$ . When a valid set is identified, it is used for signature construction, and that set is removed from the precomputed list. If no precomputed sets meet the conditions, the algorithm reverts to the original Dilithium signing method, computing parameters in real-time. This fallback ensures that the signing process remains functional, even when precomputed values are unavailable or unsuitable.

APPENDIX B  
CRYPTOGRAPHIC SECURITY NOTIONS

**Definition 8** (EU-CMA). *Existential Unforgeability under Chosen Message Attack (EU-CMA) experiment*  $\text{Exp}_{\Pi, A}^{\text{EU-CMA}}$

---

**Algorithm 5** Dilithium with Precomputation

**Precomputing:**  $\mathcal{LS} \leftarrow \text{Precmp}(\text{sk}_{\Pi}, N)$ 


---

**Input:** Dilithium private key  $\text{sk}_{\Pi}$  and the number of groups for Dilithium parameters to be precomputed  $N$ .

**Output:**  $\mathcal{LS}$ , in which  $\mathcal{LS}[\text{sk}_{\Pi}]$  including  $N$  groups of precomputed Dilithium parameters for  $\text{sk}_{\Pi}$ 


---

```

1:  $\mathcal{LS}[\text{sk}_{\Pi}] := \emptyset$ 
2:  $\mathbf{A} \in \mathbb{R}_q^{k \times \ell} := \text{ExpandA}(\rho)$ 
3:  $u \in \{0, 1\}^{384} := \text{CRH}(tr)$ 
    $\triangleright$  In original Dilithium:  $\mu \leftarrow \text{CRH}(tr \parallel m)$ 
4:  $\kappa := 0$ 
5: while Length of  $\mathcal{LS}[\text{sk}_{\Pi}] < N$  do
6:    $y \in \mathbb{S}_{\gamma_1 - 1}^\ell := \text{ExpandMask}(K \parallel u \parallel \kappa)$ 
    $\triangleright$  Using  $u$  not  $\mu$ 
7:    $\mathbf{w} := \mathbf{A}\mathbf{y}$ 
8:    $\mathbf{w}_1 := \text{HighBits}(\mathbf{w}, 2\gamma_2)$ 
9:   if  $(A, u, y, w, w_1)$  not in  $\mathcal{LS}[\text{sk}_{\Pi}]$  then
10:     Add  $(A, u, y, w, w_1)$  into  $\mathcal{LS}[\text{sk}_{\Pi}]$ 
11:   end if
12:    $\kappa := \kappa + 1$ 
13: end while
14: return  $\mathcal{LS}$ 

```

---

**Signing with precomputation:**  $\sigma \leftarrow \text{PSgn}(\text{sk}_{\Pi}, m, \mathcal{LS})$ 
**Input:** Dilithium private key  $\text{sk}_{\Pi}$ , message to be signed  $m$ , and the list of precomputed Dilithium parameters  $\mathcal{LS}$ .

**Output:** the signature  $\sigma$ .

---

```

1: for  $(A, u, y, w, w_1)$  in  $\mathcal{LS}[\text{sk}_{\Pi}]$  do
2:    $c \in \mathbb{B}_{60} := \mathcal{H}(m \parallel u \parallel w_1)$ 
3:    $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$ 
4:    $(\mathbf{r}_0, \mathbf{r}_1) := \text{Decompose}(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$ 
5:   if  $\|\mathbf{z}\|_\infty \leq \gamma_1 - \beta$  and  $\|\mathbf{r}_0\|_\infty \leq \gamma_2 - \beta$  then
6:     Remove  $(A, u, y, w, w_1)$  from  $\mathcal{LS}[\text{sk}_{\Pi}]$ 
7:     return  $\sigma = (\mathbf{z}, c)$ 
8:   end if
9: end for
    $\triangleright$  If none of the precomputed parameters are satisfied, invoke the original Dilithium signing algorithm.
10:  $\mu \leftarrow \text{CRH}(tr \parallel m)$ 
11:  $z := \perp$ 
12: while  $z = \perp$  do
13:    $y \in \mathbb{S}_{\gamma_1 - 1}^\ell := \text{ExpandMask}(K \parallel \mu \parallel \kappa)$ 
14:    $\mathbf{w} := \mathbf{A}\mathbf{y}$ 
15:    $\mathbf{w}_1 := \text{HighBits}(\mathbf{w}, 2\gamma_2)$ 
16:    $c \in \mathbb{B}_{60} := \mathcal{H}(\mu \parallel w_1)$ 
17:    $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$ 
18:    $(\mathbf{r}_0, \mathbf{r}_1) := \text{Decompose}(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$ 
19:   if  $\|\mathbf{z}\|_\infty > \gamma_1 - \beta$  or  $\|\mathbf{r}_0\|_\infty > \gamma_2 - \beta$  then
20:      $z := \perp$ 
21:   end if
22: end while
23: return  $\sigma = (\mathbf{z}, c)$ 

```

---

for a signature scheme  $\Pi: (\text{KeyGen}, \text{Sign}, \text{Verify})$  with an adversary  $A$  is defined as follows.

- $(\text{sk}, \text{pk}) \leftarrow \Pi.\text{KeyGen}(1^\kappa)$
- $(m^*, \sigma^*) \leftarrow A^{\Pi.\text{Sign}(\cdot)}(\text{pk})$
- If  $1 \leftarrow \Pi.\text{Verify}(\cdot)$  and  $m^*$  was never queried to  $\text{Verify}(\cdot)$ , return 'success' otherwise, output ' $\perp$ '.

**Definition 9** (IND-CCA). *The indistinguishability of a key encapsulation scheme  $\mathcal{E} = \{\text{KeyGen}, \text{Encaps}, \text{Decaps}\}$  with key space  $\mathcal{K}$  under chosen ciphertext attack (IND-CCA) experiment  $\text{Expt}_{\mathcal{E}, A}^{\text{IND-CCA}}$  with an adversary  $A$  is defined as follows.*

- $(\text{sk}, \text{pk}) \leftarrow \Pi.\text{KeyGen}(1^\kappa)$
- $b \leftarrow \{0, 1\}, (c_x, x_0) \leftarrow \Pi.\text{Encaps}(\text{pk})$
- $x_1 \leftarrow \mathcal{K}$
- $b' \leftarrow A^{\Pi.\text{Encaps}(\cdot), \Pi.\text{Decaps}(\cdot)}(\text{pk}, c_x, x_b)$

*The advantage of the adversary in the above experiment is defined as  $\Pr[b = b'] \leq \frac{1}{2} + \varepsilon$  for a negligible  $\varepsilon$ .*