

SecAgg++: A Post-Quantum Privacy Preserving Federated Learning Protocol

Alessandro Perez
aperez@fbk.eu

Fondazione Bruno Kessler,
Department of Cybersecurity,
Trento, Italy

Giulia Cavicchioni
gcavicchioni@fbk.eu

Fondazione Bruno Kessler,
Department of Cybersecurity,
Trento, Italy

Silvio Cretti
scretti@fbk.eu

Fondazione Bruno Kessler,
Department of Cybersecurity,
Trento, Italy

Riccardo Longo
rlongo@fbk.eu

Fondazione Bruno Kessler,
Department of Cybersecurity,
Trento, Italy

Alessandro Tomasi
altomasi@fbk.eu

Fondazione Bruno Kessler,
Department of Cybersecurity,
Trento, Italy

Silvio Ranise
sranise@fbk.eu

Fondazione Bruno Kessler and
University of Trento
Trento, Italy

Abstract

Secure aggregation is a fundamental building block for privacy-preserving federated learning, yet most existing masking-based secure aggregation protocols do not provide post-quantum security. In this work, we present SecAgg++, a masking-based post-quantum secure aggregation protocol for the single-server setting. SecAgg++ integrates post-quantum cryptographic primitives while preserving the communication and computational efficiency of classical frameworks such as SecAgg and SecAgg+, and supports client dropout handling through an efficient recovery procedure.

CCS Concepts

• **Security and privacy** → **Cryptography**; *Public key (asymmetric) techniques*; *Public key encryption*;

Keywords

Secure Aggregation, Post-Quantum Cryptography, Federated Learning, Secret Sharing, Additive Masking

1 INTRODUCTION

Federated Learning (FL) enables distributed clients to collaboratively train a global machine learning model without transmitting their raw training data to a central server. Since local model updates may still leak sensitive information, practical FL systems rely on secure aggregation protocols to ensure that the server learns only the aggregate of the clients' updates.

In the single-server setting, secure aggregation protocols must satisfy several practical requirements. Secure aggregation should provide privacy in the presence of an honest-but-curious server that may collude with a subset of clients, scale efficiently with the number of participants, and remain robust against client dropouts caused by unreliable connectivity or limited client availability.

The foundation of cryptographically secure aggregation was established by SecAgg [8] and later optimized by SecAgg+ [4]. These protocols rely on pairwise additive masking combined with threshold secret sharing: clients mask their local updates using pairwise random masks, while the server reconstructs the masks of dropped participants during a recovery phase to ensure correct aggregation.

Despite their widespread adoption, existing masking-based protocols suffer from two important limitations. First, the cost of dropout recovery increases with the number of offline clients. Second, their security relies on classical public-key primitives, including Diffie–Hellman key agreement and classic digital signatures, which are vulnerable to quantum attacks based on Shor's algorithm [31]. This motivates the need for post-quantum (PQ) secure alternatives.

Designing a PQ counterpart of masking-based secure aggregation protocols is, however, non-trivial. While PQ primitives such as Key Encapsulation Mechanisms (KEM) may seem natural replacements of classical key-agreement protocols, their asymmetric nature fundamentally changes the structure of pairwise mask generation and introduces new challenges in handling client dropouts efficiently. Consequently, achieving PQ security while preserving scalability and dropout robustness requires a structural redesign of the protocol rather than a simple replacement of the underlying cryptographic primitives.

Contributions. In this work, we propose SecAgg++, a PQ secure aggregation protocol based on lattice assumptions. By leveraging ML-KEM for pairwise mask establishment and introducing a unilateral masking mechanism, SecAgg++ preserves the privacy guarantees and practical efficiency of SecAgg and SecAgg+. The proposed design introduces a trade-off: while client dropouts are tolerated before the update submission phase, surviving clients are required to remain online during the initial steps of the unmasking procedure. We implement SecAgg++ using the Flower Framework [6] and evaluate its performance.

Organization. In Section 2, we review the necessary background on privacy-preserving federated learning, with particular emphasis on masking-based secure aggregation, and introduce the PQ cryptographic primitives used throughout the paper. In Section 3, we discuss the state of the art of both classical and PQ masking-based secure aggregation protocols. In Section 4, we first provide a high-level overview of the design of SecAgg++ and then present the protocol in detail together with its experimental evaluation. Finally, the security analysis is presented in Section 5.

2 PROBLEM SETTING

2.1 Privacy Preserving Federated Learning

Federated Learning (FL) is a distributed learning paradigm in which multiple clients collaboratively train a global model under the coordination of a central server. Training in FL is performed through an iterative process in which clients interact via a server to update the global model. At each iteration t :

- (1) The server broadcasts the current global model, denoted by $\mathbf{x}(t)$, to the participating clients;
- (2) Each client performs local training on its private dataset and computes a local model update $\mathbf{x}_i(t)$;
- (3) Clients send their updates to the server, which aggregates them;
- (4) The server updates the global model and broadcasts $\mathbf{x}(t+1)$ for the next training round.

In practice, some clients may drop out during training due to unreliable communication or device availability. Therefore, the server aims to compute the aggregate of the surviving clients' updates. That is, if $\mathcal{U}(t)$ denotes the set of surviving clients at iteration t , the server computes $\mathbf{x}(t+1) = \frac{1}{|\mathcal{U}(t)|} \sum_{i \in \mathcal{U}(t)} \mathbf{x}_i(t)$. Moreover, since raw local model updates may leak sensitive information about the underlying data [36], the aggregation step represents a major privacy bottleneck. Secure aggregation protocols address this issue by enabling the server to obtain only the aggregate of the clients' updates without learning any individual model. Several privacy-preserving approaches have been proposed, including non-cryptographic techniques such as Differential Privacy, as well as cryptographic methods including Secure Multi-Party Computation [11], Homomorphic Encryption [40], and masking-based secure aggregation – see [22] for an SoK.

Since our protocol follows the masking-based secure aggregation paradigm, we focus exclusively on this class. In these protocols, each client masks its local update by adding a mask before transmission, the masks are removed only once the updates have been aggregated, ensuring that for the central server the individual updates remain hidden. Existing masking schemes can be broadly categorized into two paradigms based on how the masks are generated. The first paradigm relies on pairwise masking, where each client constructs its mask using pairwise secrets shared with other clients. The seminal SecAgg protocol follows this approach [8], but its communication cost scales with the number of pairwise exchanges, limiting scalability. Several improvements have been proposed, among which SecAgg+ [4] is one of the most representative. These protocols constitute the main baselines for our work and are reviewed in Section 3. Other examples include TurboAgg [32], which exploits a circular topology, and SwiftAgg [18], which reduces communication overhead by partitioning clients into groups. A second paradigm, introduced in [33], relies on one-shot aggregate-mask reconstruction, allowing the server to recover a single aggregate mask independently of the number of dropped clients.

2.2 Post-Quantum Cryptography

The security of most contemporary privacy-preserving FL protocols, particularly masking-based secure aggregation schemes, relies fundamentally on classical public-key cryptography. These classical

primitives base their security on the difficulty of solving certain number-theoretic problems, most notably integer factorization and the discrete logarithm problem. However, number-theoretic problems represent the primary domain in which quantum computing yields an exponential speedup, mainly via Shor's algorithm [31], which can solve integer factorization and discrete logarithms in polynomial time. Conversely, symmetric-key cryptography remains significantly more resilient to quantum attacks; quantum search algorithms, and in particular Grover's algorithm [17], offer only an asymptotic quadratic speedup over classical exhaustive search, and in practice the quantum advantage is minimal, due e.g. to the impossibility of parallelization.

To address the vulnerability of asymmetric primitives, the cryptographic community has developed Post-Quantum Cryptography (PQC), built upon mathematical problems that remain computationally intractable for both classical and quantum adversaries. Among the various branches of PQC, lattice-based cryptography has emerged as one of the leading candidates, providing efficient, compact, and versatile constructions for both key encapsulation and digital signatures [16, 25, 26], while relying on relatively strong security guarantees [28]. In particular, ML-KEM, the standardized post-quantum key-encapsulation mechanism, bases its security on the hardness of the Module Learning With Errors (M-LWE) problem and constitutes the core cryptographic primitive underlying the protocol proposed in Section 4.2. Further details are provided in Section 2.3.

2.3 Cryptographic primitives

This section reviews the cryptographic primitives used throughout the proposed protocol. Although the framework is compatible with different instantiations, we focus on primitives providing security against quantum adversaries.

Secret Sharing. A (t, n) -threshold secret sharing scheme enables a dealer to distribute a secret among n participants in such a way that any subset of at least t shares can reconstruct the secret, whereas any collection of fewer than t shares reveals no information about it. Throughout this work, we instantiate secret sharing using Shamir's secret sharing scheme [30]. The scheme consists of two algorithms:

Sharing Shamir(s, t, n) $\rightarrow \{\text{share}_s(i, j)\}_{j \in \mathcal{N}}$: given a secret s held by participant i , a threshold t , and a set of participant identifiers $\mathcal{N}$; it outputs one share for each participant;

Reconstruction Combine($\{\text{share}_s(i, j)\}_{j \in \mathcal{V}}$) $\rightarrow s$: given the shares of a subset of the participants $\mathcal{V} \subseteq \mathcal{N}$ with $|\mathcal{V}| \geq t$; it recovers the secret.

Key Encapsulation Mechanism (KEM). In the PQ setting, shared keys are typically established using a KEM, rather than a key agreement protocol as in classical cryptography. A KEM consists of the following algorithms:

Key Generation KeyGen(pp): on input the public parameters pp , it generates a public encapsulation key pub and the corresponding private decapsulation key prv ;

Encapsulation Encaps(pub): on input the recipient's public key pub , it generates a ciphertext c and a shared secret K ;

Decapsulation $\text{Decaps}(\text{prv}, c)$: on input the private key prv and the ciphertext c , it recovers the shared secret K' , where correctness guarantees that $K' = K$.

Unlike classical key agreement protocols such as Diffie-Hellman, where both parties contribute symmetrically to deriving the shared secret, a KEM assigns distinct roles to the communicating parties. The sender encapsulates a shared secret using the recipient's public key, while the recipient recovers it using the corresponding private key. This asymmetric interaction is an intrinsic structural property of KEMs and has important implications for protocol design, as illustrated in Section 4.1. In practice, the shared secret is typically used as input to a key derivation function to compute one or more cryptographic keys for subsequent operations. Among the standardized post-quantum KEMs, we employ ML-KEM768 [26] in our construction. ML-KEM is obtained from an underlying IND-CPA-secure public-key encryption scheme by applying a variant of the Fujisaki-Okamoto transform, yielding an IND-CCA2-secure KEM.

Authenticated Encryption. Authenticated Encryption with Associated Data (AEAD) provides confidentiality for an encrypted message while simultaneously ensuring the integrity and authenticity of both the message and cleartext associated data. An AEAD scheme consists of the following algorithms:

Encryption $\text{Enc}(k, n, a, m) \rightarrow c$: given a key k , a nonce n , associated data a , and a message m , it outputs a ciphertext c ;

Decryption $\text{Dec}(k, n, a, c) \rightarrow m$: on input a key k , nonce n , associated data a , and ciphertext c , it outputs either the original message m or the failure symbol $\perp$.

Correctness requires that $\text{Dec}(k, n, a, \text{Enc}(k, n, a, m)) = m$ for any valid input. A secure AEAD scheme satisfies indistinguishability under chosen-plaintext attacks (IND-CPA) and ciphertext integrity (INT-CTXT), ensuring that an adversary cannot distinguish encryptions of chosen messages or forge a valid ciphertext, except with negligible probability. In our framework, we instantiate this primitive using AES-GCM [23].

Hybrid Encryption (KEM-AEAD Composition). To transport pairwise-mask seeds and Shamir shares between neighbours, we employ the natural composition of ML-KEM and AEAD into a single public-key encryption scheme. Let ℓ_{nonce} denote the fixed AEAD nonce length and ℓ_{ct} the fixed length of a ML-KEM ciphertext, both determined by the chosen parameter set:

Encryption $\text{Enc}(\text{pub}, m) \rightarrow c$:

```

( $ml\_encap, ss$ )  $\leftarrow$  ML-KEM.Encaps(pub)
 $key \leftarrow$  HKDF( $ss$ )
 $nonce \leftarrow \{0, 1\}^{\ell_{\text{nonce}}}$ 
 $ct \leftarrow \text{Enc}_{\text{AEAD}}(nonce, AD = ml\_encap, key, m)$ 
 $c \leftarrow nonce \parallel ml\_encap \parallel ct$ 

```

Decryption $\text{Dec}(\text{prv}, c) \rightarrow m$:

```

 $nonce, ml\_encap, ct \leftarrow c$ 
 $ss \leftarrow \text{ML-KEM.Decaps}(prv, ml\_encap)$ 
 $key \leftarrow \text{HKDF}(ss)$ 
 $m \leftarrow \text{Dec}_{\text{AEAD}}(nonce, AD = ml\_encap, key, ct)$ 

```

The KEM ciphertext ml_encap is included as the AEAD's associated data, cryptographically binding the symmetric key and plaintext to this specific encapsulation and preventing an adversary from splicing a valid ml_encap from one exchange with an AEAD ciphertext from another. HKDF [21] is used as the key-derivation function mapping the raw ML-KEM shared secret ss to a symmetric key of the length required by AEAD.

Pseudorandom Generator. A Pseudorandom Generator (PRG) deterministically expands a short random seed into a longer output that is computationally indistinguishable from a uniformly random string of the same length. A PRG consists of a single algorithm:

Expansion $\text{PRG}(s) \rightarrow y$: on input a seed $s \in \{0, 1\}^\lambda$, outputs a pseudorandom string $y \in \{0, 1\}^{\ell(\lambda)}$, where $\ell(\lambda) \gg \lambda$.

Note that the output of PRG can naturally be interpreted as a value in $\mathbb{Z}_q$ where $q = 2^{\ell(\lambda)}$, and the distribution of these values is indistinguishable from a uniform random in $\mathbb{Z}_q$. We instantiate PRG using AES-256-CTR [14].

Hash Function. In our construction we additionally use a hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^{256}$. Unlike the pseudorandom generator PRG, which is evaluated on uniformly random secret seeds, H must provide pseudorandom outputs even on adversarially chosen inputs. We model H as a random oracle in our security analysis, and instantiate it in practice with SHA3-256[15].

2.4 Threat model

Following the threat model of SecAgg [8] and its generalizations, we consider an honest-but-curious central server that faithfully executes the protocol while attempting to infer individual client updates from the protocol transcript. We further assume that up to τ clients may collude with each other, and possibly with the server, to infer local model updates of other clients. Consistent with the post-quantum setting introduced in Section 2.2, we assume a cryptographically relevant quantum adversary, namely one capable of breaking classical public-key cryptographic primitives.

3 STATE OF THE ART FOR MASKING-BASED SECURE AGGREGATION

3.1 Classical Secure Aggregation Baselines

The seminal Secure Aggregation protocol (SecAgg) [8] and its extension SecAgg+ [4] established the foundation for privacy-preserving federated learning through a robust double-masking mechanism.

To protect its privacy during a given training round t , each client i hides its local model update $\mathbf{x}_i(t)$, or simply $\mathbf{x}_i$, by adding two distinct layers of masks before transmission. The first layer consists of a set of pairwise masks derived from shared secret seeds established with its neighbours, a subset (but possibly all) of the participants, via a Diffie-Hellman key agreement:

$$s_{i,j} = \text{KeyAgree}(\text{prv}_i, \text{pub}_j) = \text{KeyAgree}(\text{prv}_j, \text{pub}_i) \quad (1)$$

where prv_i and pub_i represent, respectively, the private and public keys of client i . The second layer is a self-mask generated from a private seed u_i , which preserves privacy in case of delayed clients. Let $\mathcal{N}_i$ denote the set of neighbours to which i belongs. The masked

model update y_i transmitted by client i to the central server is:

$$y_i = \mathbf{x}_i + \text{PRG}(u_i) + \sum_{j \in \mathcal{N}_i: i < j} \text{PRG}(s_{i,j}) - \sum_{j \in \mathcal{N}_i: i > j} \text{PRG}(s_{j,i}),$$

where PRG denotes a pseudorandom generator.

To handle potential client dropouts during execution, each client distributes Shamir secret shares of both its private key prv_i and its self-mask seed u_i to its neighbours $\mathcal{N}_i$. During the recovery phase, if a client survives, the server collects its shares to reconstruct its self-mask seed u_i ; if a client drops out, the server instead collects its shares to reconstruct its private key prv_i , allowing the reconstruction of the missing pairwise masks. Therefore, the server can recover only the aggregate model update.

The protocol strictly guarantees client model update privacy against an honest-but-curious server colluding with up to $t < \tau$ malicious clients, provided that the number of surviving clients satisfies $n - d > \tau$, where d denotes the total number of dropped clients and τ represents the threshold parameter of the underlying Shamir secret sharing scheme.

While the original SecAgg protocol provides strong privacy guarantees, it requires a fully connected communication graph, with each client communicating with every other participant, i.e., $\mathcal{N}_i = [1, \dots, n]$, where n denotes the total number of clients participating in the FL process. As a result, the protocol incurs a communication cost of $\mathcal{O}(n^2)$. To overcome this bottleneck, SecAgg+ preserves the core double-masking technique, privacy guarantees, and dropout resilience, but replaces the complete communication graph with a sparse, regular random graph where the neighbours size is $|\mathcal{N}_i| = \mathcal{O}(\log n)$. Restricting each client to communicate with only $\mathcal{O}(\log n)$ neighbours, substantially reduces both the communication and computational overhead compared to SecAgg.

Both SecAgg and SecAgg+ rely on classical cryptographic primitives and do not provide security against quantum adversaries, motivating the post-quantum secure aggregation protocol presented in Section 4.

3.2 Post-Quantum Secure Aggregation: State of the Art and Challenges

A significant number of post-quantum privacy-preserving federated learning protocols rely on lattice-based cryptography. In particular, the CKKS homomorphic encryption scheme [13] and its multi-key variants [12] are widely adopted in privacy-preserving federated learning due to their favorable trade-off between computational efficiency and numerical precision [29, 38]. More recently, OPA [20] and Willow [5] introduced communication-efficient one-shot secure aggregation protocols based on lattice assumptions, while alternative constructions based on coding-theoretic assumptions have also been proposed [7]. These approaches, however, do not follow the pairwise-masking paradigm underlying SecAgg.

In contrast, post-quantum adaptations of pairwise masking-based secure aggregation protocols have received considerably less attention. In this masking paradigm, pairwise secrets are established between every pair of participating clients via a key agreement protocol. Replacing this primitive with a post-quantum KEM, however, is not trivial. The asymmetric encapsulation-decapsulation paradigm differs fundamentally from the symmetric nature of traditional key agreements, where both parties execute identical operations to

jointly compute a shared secret. In particular, as detailed in Section 4.1, this asymmetry complicates the management of client dropouts during the recovery phase.

To address this issue, various approaches have been explored. For instance, [41] employs a reconciliation-based variant of FrodoKEM [9] to mimic a key-agreement primitive, while PQSF [39] adopts the reconciliation-based NewHope protocol [3] to derive pairwise masks. Although reconciliation-based schemes naturally recover the symmetric behaviour of classical key exchange, they have largely been replaced in practice by KEM-based constructions, and the current standardization efforts (most notably by NIST) have fully converged on KEMs. Indeed, even the NewHope designers later suggested using a KEM-based construction over the original reconciliation mechanism due to its reduced complexity [2, Section 1.3]. Finally, [37] proposes a much more complex three-round MPC-based secure aggregation protocol based on single masking and seed-homomorphic pseudorandom generators.

These motivate the need for masking-based secure aggregation protocols that are compatible with standardized PQ KEMs while preserving the functionality and security guarantees of their classical counterparts.

4 SECAGG++ PROTOCOL

We now formally present our secure aggregation protocol. Let $\mathcal{U}$ denote the set of clients participating in a given training round. Each client $i \in \mathcal{U}$ holds a private local update vector $\mathbf{x}_i$ of length m . For simplicity, we assume that each $\mathbf{x}_i$ has entries in the ring $\mathbb{Z}_q$, with q a large integer. In practice we use $q = 2^{32}$. To avoid modular reduction during aggregation, local vector entries are constrained to a bounded range, guaranteeing that the worst-case aggregate sum remains strictly less than q . The objective of the protocol is to allow the server to compute the aggregate:

$$\sum_{i \in \mathcal{U}} \mathbf{x}_i \in \mathbb{Z}_q^m$$

without revealing any individual update $\mathbf{x}_i$ to the server or to any other client.

4.1 A First Attempt: Replacing Diffie–Hellman with ML-KEM

A natural first approach to constructing a post-quantum secure aggregation protocol is to start from the classical SecAgg or SecAgg+ frameworks and replace their non-post-quantum primitives with quantum-resistant counterparts. Specifically, in the original protocols, the pairwise seeds used to generate masks are established via a Diffie–Hellman key exchange (Equation (1)). A core property of Diffie–Hellman is its symmetry: both parties actively contribute with their private keys to derive the shared secret. Since key establishment in the post-quantum setting is typically achieved with KEMs, a straightforward modification is to replace the Diffie–Hellman with a lattice-based primitive such as ML-KEM. However, this plug-and-play modification introduces a fundamental structural limitation. Unlike Diffie–Hellman, a KEM encapsulation is non-interactive and unidirectional: only the encapsulator contributes in the mask generation, while the decapsulator’s private

key is the sole key enabling the recovery of the shared secret. Assuming a total ordering on the client set, for each pair (i, j) with $i < j$, client i acts as the encapsulator (sender) and client j as the decapsulator (receiver). Client i executes $\text{Encaps}(\text{pub}_j) \rightarrow (s_{i,j}, c_{i,j})$ to generate a shared secret $s_{i,j}$ and its corresponding ciphertext $c_{i,j}$, which client j then decapsulates using prv_j , yielding $s_{j,i} = s_{i,j}$. The critical flaw emerges during the dropout recovery phase:

- If the decapsulator j drops, the server can recover $s_{i,j}$ by reconstructing prv_j from its Shamir shares (distributed during the setup phase) by executing $\text{Decaps}(\text{prv}_j, c_{i,j}) \rightarrow s_{i,j}$.
- If the encapsulator i drops: reconstructing the sender's private key prv_i does not allow to retrieve information. Since the encapsulation algorithm $\text{Encaps}(\text{pub}_j)$ depends solely on the receiver's public key pub_j , the private key prv_i has no algebraic relation to either the ciphertext $c_{i,j}$ or the resulting secret $s_{i,j}$. Thus, the only key capable of recovering $s_{i,j}$ is the receiver's private key prv_j . However, since j is active, reconstructing prv_j would expose j 's updates.

A naive solution would require each client to distribute Shamir shares of every pairwise secret $s_{i,j}$ they generate, but this forces clients to distribute a quadratic (in the size of the neighbours) number of shares. To mitigate this asymmetry, one might consider a symmetrization technique such as a double ML-KEM exchange, in which both parties simultaneously act as encapsulator and decapsulator and derive the pairwise mask as $\text{PRG}(s_{i,j} \| s_{j,i})$. While this restores the symmetric contribution of the two clients to the mask generation process, it does not eliminate the dropout recovery bottleneck. Indeed, if either client drops out, reconstructing the pairwise mask still requires revealing the surviving client's secret key, thereby compromising the privacy of its local update. These observations suggest that a replacement of classical key agreements with post-quantum KEMs is insufficient. Instead, the protocol architecture itself must be redesigned to integrate KEM asymmetry while preserving the security guarantees of classical secure aggregation. The construction presented in Section 4.2 addresses this challenge.

4.2 Masking and unmasking in SecAgg++

To overcome the asymmetry introduced by post-quantum KEMs described in the previous section, we adopt a *unilateral masking* approach: each client independently generates masks for its neighbours and distributes them using a post-quantum hybrid encryption mechanism, combining ML-KEM with an AEAD (see Section 2.3). This design naturally aligns with the encapsulation-decapsulation workflow of post-quantum KEMs while enabling efficient dropout recovery.

Let $\mathcal{U}$ denote the set of participating clients, with $|\mathcal{U}| = n$. For each client i , let $\mathcal{N}_i \subseteq \mathcal{U}$ denote its neighbours.

Masking Phase. Each client i samples independently two seeds: a self-mask seed u_i together with a master seed b_i , which is later used to derive a collection of pairwise seeds $\{k_j^i\}_{j \in \mathcal{N}_i}$, where the superscript identifies the sender and the subscript identifies the intended receiver. Intuitively, k_j^i represents the masking contribution generated by the client i for the neighbour j . The client i distributes

each seed k_j^i using hybrid encryption (Section 2.3). The recipient j then recovers the seed using its private key.

After local training, client i masks its model update $\mathbf{x}_i$ as:

$$\mathbf{y}_i = \mathbf{x}_i + \text{PRG}(u_i) + \sum_{j \in \mathcal{N}_i} \text{PRG}(k_j^i) \mod q.$$

The self-mask is crucial for protecting against late client responses in the dropout recovery procedure. Specifically, it ensures the server cannot learn an individual update if a client is mistakenly marked as dropped but later submits its masked update. For further details, see the original SecAgg construction [8, Section 4.0.3]. In addition to the masked update $\mathbf{y}_i$, client i computes the aggregate of all pairwise masks generated for its neighbours:

$$\sigma_i = \sum_{j \in \mathcal{N}_i} \text{PRG}(k_j^i) \mod q,$$

As will become clear in the dropout recovery and unmasking phases, the quantities σ_i allow the server to remove pairwise masks. At the end of the training round, the server aggregates the masked updates from the set of surviving clients, denoted by $\mathcal{U}_1 \subseteq \mathcal{U}$:

$$\sum_{i \in \mathcal{U}_1} \mathbf{y}_i \mod q. \quad (2)$$

Recall that q is chosen sufficiently large to prevent modular wrap-around.

Unmasking Phase and Dropout Handling. The objective of the recovery phase is to remove all masking terms to reconstruct the aggregate model $\sum_{i \in \mathcal{U}_1} \mathbf{x}_i$.

Reconstructing the aggregate requires removing both the pairwise and self-masks from the received update sum $\sum_{i \in \mathcal{U}_1} \mathbf{y}_i$. Recall that each participating client i , together with its masked update, uploads the auxiliary aggregate mask $\sigma_i = \sum_{j \in \mathcal{N}_i} \text{PRG}(k_j^i)$. If all clients complete the protocol, then the set of generated masks coincides with the set of received masks, and therefore:

$$\sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{N}_i} \text{PRG}(k_j^i) = \sum_{i \in \mathcal{U}} \sigma_i \mod q.$$

In this case, subtracting the aggregation of the σ_i removes all pairwise masks. However, if a client d drops after distributing its pairwise seeds, its masked update is not included in the aggregation (2), whereas the masks generated by d still contribute to the σ_i of its surviving neighbours. To account for this discrepancy, the server reconstructs the master seed b_d from the Shamir shares distributed during the setup phase and derives all pairwise masks generated by d , yielding:

$$\sigma_d = \sum_{i \in \mathcal{N}_d \cap \mathcal{U}_1} \text{PRG}(k_i^d) \mod q.$$

Repeating the procedure for any dropped client d , the server recovers the correction term:

$$\Delta_1 = \sum_{i \in \mathcal{U}_1} \sigma_i + \sum_{d \in \mathcal{U} \setminus \mathcal{U}_1} \sigma_d \mod q.$$

Therefore, it can compute:

$$l \sum_{i \in \mathcal{U}_1} \mathbf{y}_i - \Delta_1 = \sum_{i \in \mathcal{U}_1} (\mathbf{x}_i + \text{PRG}(u_i)) - \sum_{d \in \mathcal{U} \setminus \mathcal{U}_1} \sum_{i \in \mathcal{N}_d \cap \mathcal{U}_1} \text{PRG}(k_i^d) \mod q,$$

showing that the model update is still masked by the pairwise mask of surviving clients with their dropped neighbours. To correct for

this over-subtraction, the server requests from each surviving client i their masks for dropped neighbours. More precisely, for every dropped client d and surviving client $i \in \mathcal{N}_d \cap \mathcal{U}_1$, the value k_d^i is derived, allowing the server to compute the correction term:

$$\Delta_2 = \sum_{d \in \mathcal{U} \setminus \mathcal{U}_1} \sum_{i \in \mathcal{N}_d \cap \mathcal{U}_1} \text{PRG}(k_d^i) \mod q.$$

Importantly, revealing the pairwise masks with dropped clients does not compromise the privacy of the surviving participants, whose updates remain hidden by their self masks and by the masks contributed by the other active clients.

Lastly, the server handles the self-masks. During the setup phase, each client distributed Shamir secret shares of its self-mask seed u_i . Therefore, the server reconstructs the self-mask for all $i \in \mathcal{U}_1$ and recovers the final aggregate as:

$$\sum_{i \in \mathcal{U}_1} \mathbf{x}_i = \sum_{i \in \mathcal{U}_1} \mathbf{y}_i - \sum_{i \in \mathcal{U}_1} \text{PRG}(u_i) - \Delta_1 + \Delta_2 \mod q. \quad (3)$$

The protocol supports client dropouts, but requires surviving clients to remain online during the initial steps of the recovery phase. Unlike SecAgg+, once the server receives the masked updates, it must request the pairwise masks of dropped clients before completing the unmasking procedure. Consequently, all clients in $\mathcal{U}_1$ must remain online until the pairwise masks have been removed.

4.3 Protocol Description

Our protocol, SecAgg++, is described in Algorithms 1–5. Each client i runs four stages: Setup (Algorithm 1), Share-Keys (Algorithm 2), Masked-Vector Collection (Algorithm 3), and Unmask (Algorithm 4). The server then reconstructs the secrets needed to cancel pairwise and self masks and outputs the aggregated update. For the sake of clarity, we summarize the additional notation used throughout the protocol.

- $\text{pub}_i, \text{prv}_i$: ML-KEM public and secret keys of client i ;
- Client participation throughout the protocol:
 - $\mathcal{U} \subseteq [1, \dots, n]$: clients that complete the setup phase;
 - $\mathcal{U}_1 \subseteq \mathcal{U}$: clients that submit their masked update $\mathbf{y}_i$;
 - $\mathcal{U}_2 \subseteq \mathcal{U}_1$: clients that remain online during the recovery phase and provide the required unmasking information.

Algorithm 1 SecAgg++ Client Setup

Require: Protocol parameters from server.

Ensure: Long-term ML-KEM key pair.

- 1: $(\text{pub}_i, \text{prv}_i) \leftarrow \text{ML-KEM-keygen}()$
 - 2: Send pub_i to the server. ▷ i joins $\mathcal{U}$.
-

After Algorithm 2, every client i holds the pairwise seeds $\{k_j^i\}_{j \in \mathcal{N}_i}$ received from its neighbours and the seeds $\{k_j^i\}_{j \in \mathcal{N}_i}$ it derived itself. The self-key k_i^i is derived but never transmitted; it appears symmetrically in the masked vector and in the derived-key sum, so it cancels out in the aggregate.

Algorithm 4 produces, for every surviving client i , the self-mask shares of active neighbours and the master-seed shares of stage-2 dropped neighbours, together with the correction vector δ_i . The server uses these values in Algorithm 5 to remove from the aggregate the leftover pairwise masks and the active clients' self masks.

Algorithm 2 SecAgg++ Client i : Share Keys

Require: Neighbour graph $\mathcal{N}_i$, public keys $\{\text{pub}_j\}_{j \in \mathcal{N}_i}$, threshold t , number of shares n .

Ensure: Encrypted shares sent to server; received shares stored locally.

- 1: $b_i \xleftarrow{\$} \{0, 1\}^{256}; \quad u_i \xleftarrow{\$} \{0, 1\}^{256}$
 - 2: $\{\text{share}_b(i, j)\}_{j \in \mathcal{N}_i} \leftarrow \text{Shamir}(b_i, t, n)$
 - 3: $\{\text{share}_u(i, j)\}_{j \in \mathcal{N}_i} \leftarrow \text{Shamir}(u_i, t, n)$
 - 4: **for** each neighbour $j \in \mathcal{N}_i$ **do**
 - 5: $k_j^i \leftarrow H(b_i \parallel \text{encoding}(j))$
 - 6: **end for**
 - 7: **for** each neighbour $j \in \mathcal{N}_i, j \neq i$ **do**
 - 8: $m_j \leftarrow (k_j^i, \text{share}_b(i, j), \text{share}_u(i, j))$
 - 9: $c_j^i \leftarrow \text{Enc}(\text{pub}_j, m_j)$
 - 10: Send c_j^i to the server (to be forwarded to j).
 - 11: **end for**
 - 12: **for** each forwarded ciphertext c_i^j from a neighbour $j \in \mathcal{N}_i$ **do**
 - 13: $(k_i^j, \text{share}_b(j, i), \text{share}_u(j, i)) \leftarrow \text{Dec}(\text{prv}_i, c_i^j)$
 - 14: Store received tuple indexed by j .
 - 15: **end for**
-

Algorithm 3 SecAgg++ Client: Collect Masked Vector

Require: Local model update $\mathbf{x}_i$, received pairwise keys $\{k_j^i\}$, derived keys $\{k_j^i\}$, self-mask seed u_i , modulus q .

Ensure: Masked vector $\mathbf{y}_i$ and derived-key sum σ_i .

- 1: $\text{received} \leftarrow \sum_{j \in \mathcal{N}_i} \text{PRG}(k_j^i) \mod q$
 - 2: $\text{self} \leftarrow \text{PRG}(u_i) \mod q$
 - 3: $\mathbf{y}_i \leftarrow \mathbf{x}_i + \text{received} + \text{self} \mod q$
 - 4: $\sigma_i \leftarrow \sum_{j \in \mathcal{N}_i} \text{PRG}(k_j^i) \mod q$
 - 5: Send $(\mathbf{y}_i, \sigma_i)$ to the server. ▷ i joins $\mathcal{U}_1$.
-

Algorithm 4 SecAgg++ Client: Unmask

Require: Active set $\mathcal{U}_1$, stage-2 dropped set $\mathcal{U} \setminus \mathcal{U}_1$, received shares, derived keys.

Ensure: Shares and correction vector δ_i sent to server.

- 1: $\text{share_list} \leftarrow []; \quad \text{owner_list} \leftarrow []$
 - 2: **for** each active client $\ell \in \mathcal{U}_1 \cap \mathcal{N}_i$ **do**
 - 3: Append $\text{share}_u(\ell, i)$ to share_list and ℓ to owner_list .
 - 4: **end for**
 - 5: **for** each client $d \in (\mathcal{U} \setminus \mathcal{U}_1) \cap \mathcal{N}_i$ **do**
 - 6: **if** a share $\text{share}_b(d, i)$ was received **then**
 - 7: Append $\text{share}_b(d, i)$ to share_list and d to owner_list .
 - 8: **end if**
 - 9: **end for**
 - 10: $\delta_i \leftarrow \sum_{d \in (\mathcal{U} \setminus \mathcal{U}_1) \cap \mathcal{N}_i} \text{PRG}(k_d^i) \mod q$
 - 11: Send $(\text{owner_list}, \text{share_list}, \delta_i)$ to the server. ▷ i joins $\mathcal{U}_2$.
-

Algorithms 1–5 implement the construction of Section 4.2. The correctness of Algorithm 5 follows from the symmetric cancellation of the pairwise masks k_j^i (via the σ_i and δ_i values) and the threshold reconstruction of each self-mask seed.

Algorithm 5 SecAgg++ Server: Unmask and Aggregate

Require: Masked vectors $\{y_i\}_{i \in \mathcal{U}_1}$, derived-key sums $\{\sigma_i\}_{i \in \mathcal{U}_1}$, correction vectors $\{\delta_i\}_{i \in \mathcal{U}_2}$, neighbour graph, threshold t .

Ensure: Aggregated, dequantized model update.

```

1:  $\text{agg} \leftarrow \sum_{i \in \mathcal{U}_1} y_i - \sum_{i \in \mathcal{U}_1} \sigma_i \mod q$ 
2: for each stage-2 dropped client  $d \in \mathcal{U} \setminus \mathcal{U}_1$  do
3:   Collect  $\{\text{share}_b(d, i)\}$  from online neighbours  $i \in \mathcal{N}_d \cap \mathcal{U}_2$ .
4:   if at least  $t$  shares collected then
5:      $b_d \leftarrow \text{combine}(\{\text{share}_b(d, i)\})$ 
6:     for each surviving neighbour  $i \in \mathcal{N}_d \cap \mathcal{U}_1$  do
7:        $k_i^d \leftarrow H(b_d \parallel \text{encoding}(i))$ 
8:        $\text{agg} \leftarrow \text{agg} - \text{PRG}(k_i^d) \mod q$ 
9:     end for
10:  else
11:    abort: insufficient shares to reconstruct  $b_d$ .
12:  end if
13: end for
14:  $\text{agg} \leftarrow \text{agg} + \sum_{i \in \mathcal{U}_2} \delta_i \mod q$ 
15: for each active client  $\ell \in \mathcal{U}_1$  do
16:   Collect  $\{\text{share}_u(\ell, i)\}$  from surviving neighbours  $i \in \mathcal{N}_\ell \cap \mathcal{U}_2$ .
17:   if at least  $t$  shares collected then
18:      $u_\ell \leftarrow \text{combine}(\{\text{share}_u(\ell, i)\})$ 
19:      $\text{agg} \leftarrow \text{agg} - \text{PRG}(u_\ell) \mod q$ 
20:   else
21:     abort: insufficient shares to reconstruct  $u_\ell$ .
22:   end if
23: end for
24:  $\text{final\_agg} \leftarrow \text{dequantize}(\text{agg})$ 
25: return  $\text{final\_agg}$ 

```

4.4 Experimental Evaluation and Comparison

In this section, we summarize the experimental performance of our protocol, SecAgg++, comparing it with the state-of-the-art SecAgg+ [4]. Our evaluation considers scalability, robustness to client dropouts, and wall-clock overhead across a range of federation sizes.

Hardware and deployment. All experiments were run on a single server-class machine (2× 8-core Intel Xeon Silver 4110 @ 2.1 GHz, 64 GB RAM). The system is deployed using the Flower Framework [6], with the server and every client running as independent OS processes on the same machine; server-client communication is handled transparently by gRPC [1], as integrated into Flower, so no protocol-specific network code is required on either the server or client applications.

Dataset and model. We use MNIST, partitioned IID into 50 disjoint shards; experiments with fewer than 50 clients use the corresponding subset of shards. Each client trains a compact CNN with 66,218 parameters:

- `Input(shape=(28,28,1))`
- `Conv2D(32, kernel_size=(3,3), activation="relu")`
- `Activation("relu")`
- `AveragePooling2D(pool_size=(3,3))`
- `Flatten()`
- `Dense(32, activation="relu")`

- `Dense(10, activation="softmax")`

Federated learning configuration. We use FedAvg, implemented directly in Flower, with `fraction_fit = fraction_evaluate = 1.0` (full client participation per round), 1 local epoch, batch size 50, and 5 communication rounds. We sweep the number of clients over $\{10, 20, 30, 40, 50\}$, repeating each configuration for 20 random seeds to obtain confidence intervals. Both protocols are configured identically wherever the parameter is shared: the fraction of clients included in each client's neighbourhood (share) ranges in $\{1.0, 0.5, 0.3\}$, the reconstruction rate (the Shamir threshold expressed as a fraction of the available shares) is 0.6, and the maximum per-coordinate weight for quantization is 1000.

Dropout emulation. As in the original SecAgg protocol, we consider the worst-case dropout scenario, where clients drop out after distributing their secret shares but before submitting their masked model updates. Consequently, the server must recover and remove the masks of the dropped clients during the unmasking phase. For each configuration we additionally sweep the target dropout fraction $\rho \in \{0.0, 0.1, 0.2\}$. Dropout is emulated client-side and deterministically seeded per round, so that the same dropout percentage is replayed identically for SecAgg+ and SecAgg++ given the same seed, thus isolating the protocol comparison from dropout variance. Concretely, each client independently recomputes the round's drop set from a seeded RNG and self-excludes if selected:

```

quota = round(num_clients * drop_percentage)
rng = Random(f"{seed}-{server_round}")
ranking = shuffle(range(num_clients), rng)
dropped = set(ranking[:quota])
if my_index in dropped: raise Exception("Client dropped")

```

Results. Figure 1 reports the wall-clock duration delta $\Delta = T_{\text{SecAgg}+} - T_{\text{SecAgg}++}$ where T denotes the total execution time of 5 training rounds, including both client- and server-side computation and communication costs. Results are shown for each share fraction, dropout rate, and client count, averaged over 20 seeded runs. Two trends are consistent across all three panels.

First, at $\rho = 0.0$ (no dropout), Δ is small and has no consistent sign: it ranges from -8.0 s (shares = 0.3, $n = 50$) to $+4.7$ s (shares = 1.0, $n = 50$), always within one combined standard deviation of the two measurements (e.g. at shares = 0.3, $n = 50$: $\Delta = -8.0$ s against a pooled $\sigma \approx 8.9$ s). This confirms that the two protocols are computationally equivalent when no dropout recovery is triggered: the dropout-free path performs the same count of KEM/DH operations and hash/PRG expansions in both protocols, differing only in primitive choice.

Second, at $\rho \in \{0.1, 0.2\}$, Δ is consistently positive and grows with n , at shares = 1.0, $\rho = 0.2$, Δ reaches 126.4 s at $n = 50$ (pooled $\sigma \approx 9.1$ s, i.e. a $\sim 14\sigma$ effect), versus 77.5 s at $\rho = 0.1$. This is the direct wall-clock signature of the $O(\kappa d)$ EC scalar multiplications SecAgg+ performs during dropout recovery, where κ represents the neighborhood size, being replaced by $O(\kappa d)$ hash evaluations in SecAgg++. The effect scales with κ , consistent with dropout-recovery cost scaling with the number of dropped clients.

Finally, F1 scores are statistically indistinguishable between protocols at every configuration (all within ± 0.01 of each other), confirming that the protocol substitution affects only wall-clock cost, not model quality or convergence.

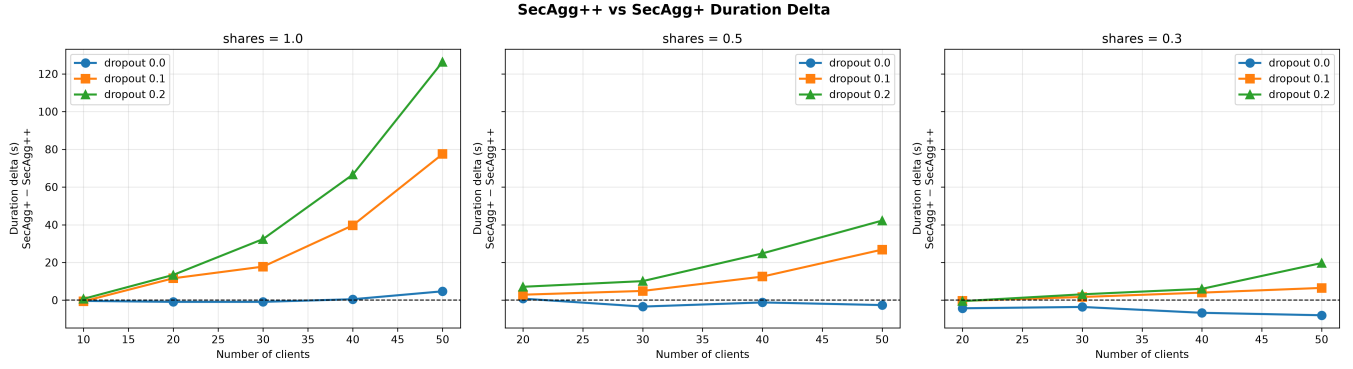


Figure 1: Time difference $\Delta = T_{\text{SecAgg+}} - T_{\text{SecAgg++}}$ between total execution time of 5 training rounds, including both client- and server-side computation and communication costs, between SecAgg+ and SecAgg++ across different number of clients and different dropout rates at varying share percentages.

Scalability. As expected, the running time of both protocols increases with the client dropout rate and with the neighbourhood size κ , which in our experiments is determined by the share fraction. In the dropout-free case, SecAgg+ and SecAgg++ exhibit identical asymptotic behaviour, requiring $O(\kappa)$ messages and per-client operations, with κ neighborhood size. The running time of both protocols increases with the share fraction, which controls the number of pairwise masks generated by each client. The two protocols diverge only in the dropout-recovery path (Algorithm 5): SecAgg+ reconstructs each dropped client’s private key and then performs one Diffie–Hellman key agreement per surviving neighbour, giving $O(\kappa d)$ elliptic-curve scalar multiplications for d dropped clients; SecAgg++ instead performs $O(\kappa d)$ hash evaluations, which are substantially cheaper in practice.

Communication overhead. Since all processes in our deployment communicate over a local gRPC channel, Figure 1 does not capture the network cost of the larger post-quantum key material. ML-KEM-768 public keys and ciphertexts (1184 B and 1088 B) are substantially larger than their X25519 counterparts (32 B each). However, each pairwise seed distribution in SecAgg++ is functionally equivalent to a key exchange in a TLS1.3 handshake, a setting where the impact of post-quantum KEMs on real networks has been extensively measured. Paquin et al. [27] show that on reliable links handshake completion time is dominated by computation rather than message size, and that lattice-based KEMs perform on par with ECDH. More recent measurements confirm that ML-KEM performs comparably to elliptic-curve key exchange in both TLS 1.3 and QUIC, under ideal as well as delayed network conditions [24, 35]. Finally, Kampanakis and Childs-Klein [19] show that the relative impact of ML-KEM on total connection time diminishes as the amount of transferred data grows, staying below 5% on high-bandwidth, stable networks. In federated learning, where every key exchange is followed by the upload of a full model update, this is the favourable regime.

Security. SecAgg++ strictly strengthens the security assumption underlying key establishment: SecAgg+’s pairwise secrets rely on

the hardness of the discrete logarithm problem, broken in polynomial time by Shor’s algorithm [31], whereas SecAgg++’s pairwise secrets are established via ML-KEM, whose security reduces to the Module Learning With Errors problem, believed hard even for quantum adversaries (see Section 2.2).

Reproducibility. To support evaluation of our contributions, we provide the repository containing artifacts described below, available at <https://github.com/aleph-fbk/SecAggPlusPlus>. The repository includes:

- Source code implementing SecAgg++, including the post-quantum key-encapsulation and encryption primitives (ML-KEM, AEAD composition) described in Section 4.3.
- Simulation scripts used to produce the comparative evaluation between SecAgg+ and SecAgg++ across varying client counts, dropout rates, and subgroup sizes (Section 4.4).
- Configuration files specifying the parameter sets (client counts, dropout rates, thresholds, security parameter λ) used to generate each reported result.
- A README documenting setup instructions, dependencies, and the mapping between scripts and reported results/figures.

4.5 Limitations and Design Variants

We conclude this section by making explicit two design choices of SecAgg++ that trade flexibility for efficiency, and by describing the corresponding protocol variants that recover the classical behaviour.

Dropout policy. SecAgg and SecAgg+ tolerate client dropouts at every stage of a round: as long as at least t shares of the relevant secret reach the server, a client may leave after advertising its key, after distributing its shares, after submitting its masked update, or during the unmasking phase itself. SecAgg++ tolerates dropouts during setup, key distribution, and update submission exactly as its classical counterparts do. The main distinction lies in the final unmasking step: clients that successfully submit masked updates must remain online for one additional round if they have dropped neighbors, enabling the server to finalize unmasking of unilateral pairwise masks. Crucially, this operational requirement

impacts only round completion, never privacy; even if a participant drops during this narrow window, individual model updates remain protected. Furthermore, in dropout-free execution rounds, this requirement vanishes entirely. While full dropout resilience of SecAgg could be recovered by secret-sharing each pairwise key individually, doing so would scale overall share traffic from $O(n\kappa)$ to $O(n\kappa^2)$ in the neighborhood size κ . We therefore consider this brief liveness requirement a practical trade-off for maintaining post-quantum efficiency without quadratic communication overhead.

Key reuse and forward secrecy. In Algorithm 1 each client generates its ML-KEM key pair once and reuses it across training rounds, whereas SecAgg and SecAgg+ advertise a fresh Diffie-Hellman key pair in every round. This choice is what allows SecAgg++ to skip the Client Setup stage from the second round onward, but it comes at the cost of forward secrecy. We note that key reuse does not weaken the guarantees of Section 5, which hold round by round: ML-KEM is IND-CCA2-secure under an unbounded number of encapsulations to the same public key, so the reductions of Theorem 5.4 apply unchanged. The loss is confined to the case of a subsequent key compromise. Forward secrecy is recovered simply by executing Algorithm 1 at the beginning of every round (or every R rounds, for a tunable trade-off), which makes the round structure of SecAgg++ identical to that of SecAgg+. The additional cost per round is one ML-KEM key generation per client and the distribution of one public key to each of its κ neighbours. As discussed in Section 4.4, ML-KEM-768 public keys are larger than their X25519 counterparts, but the resulting handshake-like exchange performs on par with elliptic-curve key exchange on real networks, and its relative weight decreases as the volume of transferred data grows. Our measurements already support this conclusion empirically: the first round of every experiment includes the key-generation and distribution stage for both protocols, and at $\rho = 0.0$ the difference between SecAgg+ and SecAgg++ lies within the measurement noise (Figure 1). Repeating this stage in every round therefore results in a negligible per-round overhead in our experiments.

5 HONEST-BUT-CURIOUS SECURITY

To establish the privacy guarantees of the proposed protocol, we consider the honest-but-curious server model introduced in Section 2.4. In addition, we allow up to $t < \tau$ colluding clients, where τ denotes the reconstruction threshold of the underlying Shamir secret sharing scheme. Since the pseudorandom masks are added independently to each coordinate of the model update, it suffices to investigate a single coordinate. Accordingly, for client i , we denote by x_i a coordinate of its local model update $\mathbf{x}_i$, and by y_i the corresponding coordinate of its masked representation $\mathbf{y}_i$.

5.1 Information-Theoretic Privacy of the Masking Layer

The server's observations during a training round consist of the masked updates $y_1, \dots, y_n$ and the auxiliary values $\sigma_1, \dots, \sigma_n$, where n denotes the number of surviving clients. We model the relation between these observations and the underlying private quantities through a linear system represented by a matrix $\mathbf{M} \in \mathbb{Z}_q^{n(n+1) \times 2n}$. The columns of $\mathbf{M}$ correspond to the values observed by the server,

namely $\{y_i\}_{i \in [1, \dots, n]}$ and $\{\sigma_i\}_{i \in [1, \dots, n]}$, while the rows correspond to the unknown variables, namely the updates $\{x_i\}_{i \in [1, \dots, n]}$ and the pairwise masking $\{k_j^i\}_{i, j \in [1, \dots, n]}$. An entry in $\mathbf{M}$ is equal to 1 if the unknown variable u contributes to the observed messages, 0 otherwise. Note that, because the server reconstructs the self-mask u_i , as shown in Equation 3, $\text{PRG}(u_i)$ is already known to the server and are thus excluded from the system's unknowns. Also, if a client d drops, no y_d and σ_d are observed and x_d never enters the transcript; hence, the corresponding columns and rows are omitted from $\mathbf{M}$. For each surviving neighbour $j \in \mathcal{N}_d$, the server reconstructs k_j^d from the surviving participants and subtracts $\text{PRG}(k_j^d)$ from y_j . Consequently, we also omit the rows corresponding to k_j^d . What remains is exactly the submatrix of surviving clients, with no references to the dropped client d . By construction, the only non-zero entries are:

$$\mathbf{M}[x_i, y_i] = 1, \quad \mathbf{M}[k_j^i, y_j] = 1, \quad \mathbf{M}[k_i^j, \sigma_i] = 1,$$

for every pair of neighbouring clients (i, j) . For simplicity, we also omit columns corresponding to non-neighbouring pairs, since the associated masking variables never appear in any message. Notice that when investigating the privacy of a specific target client ℓ , the relevant submatrix restricts to its neighbours $\mathcal{N}_\ell$, yielding dimensions $(|\mathcal{N}_\ell|+1)|\mathcal{N}_\ell| \times 2|\mathcal{N}_\ell|$. For notational simplicity throughout this section, we write $\mathbf{M} \in \mathbb{Z}_q^{m(m+1) \times 2m}$, where $m = |\mathcal{N}_\ell|$. Structurally, $\mathbf{M}$ is represented in Equation (4). In particular, its $2m$ columns are partitioned into two blocks, corresponding to the observed messages $y_1, \dots, y_m$ and $\sigma_1, \dots, \sigma_m$. The rows are partitioned into m blocks, separated by dashed lines, with each block corresponding to a client $i \in [1, \dots, m]$. Within each block, the first row corresponds to the local model update x_i , while the remaining rows correspond to the pairwise keys $\{k_j^i\}_{j \in \mathcal{N}_i}$.

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & 0 & \ddots & & & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & & \ddots & & \vdots \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & 0 & & & \ddots & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ \hline 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & 0 & \ddots & & & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & & \ddots & & \vdots \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & 0 & & & \ddots & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ \hline \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & 0 & \ddots & & & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & & \ddots & & \vdots \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & 0 & & & \ddots & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}. \quad (4)$$

Lemma 5.1. *An honest-but-curious server can isolate the private weight x_i of a target client $i \in [1, \dots, n]$ from its observations if and only if there exists a coefficient vector $\mathbf{z} \in \mathbb{Z}_q^{2m}$ that, given the incidence matrix $\mathbf{M}$ in Equation (4), we have that $\mathbf{M}\mathbf{z} = \mathbf{e}_i$, where $\mathbf{e}_i \in \mathbb{Z}_q^{m(m+1)}$ is the vector having 1 in the i th position and 0 elsewhere.*

PROOF. Let $\mathbf{v} \in \mathbb{Z}_q^{m(m+1)}$ be the column vector containing all private unknowns in a given training round. We assume the entries of $\mathbf{v}$ ordered and $\mathbf{v} := [\mathbf{x} \parallel \mathbf{k}]^\top$ where $\mathbf{x} = [x_1, \dots, x_m]$ is the vector of client weights and $\mathbf{k} = [k_1^1, \dots, k_m^m]$ is the vector containing the pairwise keys.

($\Rightarrow$) Assume the server can isolate the private weight x_i from its observations. By definition, this implies there exists some vector $\mathbf{z} \in \mathbb{Z}_q^{2m}$ allowing to retrieve x_i , independently of the values chosen for the other updates or random masks. Let $\mathbf{c}$ be the observed vector, that is $\mathbf{c} = \mathbf{M}^\top \mathbf{v}$ for some unknown $\mathbf{v} \in \mathbb{Z}_q^{m(m+1)}$:

$$\mathbf{z}^\top \mathbf{c} = x_i \implies \mathbf{z}^\top (\mathbf{M}^\top \mathbf{v}) = (\mathbf{M}\mathbf{z})^\top \mathbf{v} = \mathbf{e}_i^\top \mathbf{v}.$$

Rearranging the terms and collecting $\mathbf{v}$:

$$(\mathbf{M}\mathbf{z} - \mathbf{e}_i)^\top \mathbf{v} = 0$$

Since x_i is arbitrary and $\mathbf{k}$ uniformly distributed, this must hold for all possible $\mathbf{v} \in \mathbb{Z}_q^{m(m+1)}$, that is:

$$\mathbf{M}\mathbf{z} - \mathbf{e}_i = \mathbf{0} \implies \mathbf{M}\mathbf{z} = \mathbf{e}_i,$$

and the thesis follows.

($\Leftarrow$) Assume there exists a vector $\mathbf{z} \in \mathbb{Z}_q^{2m}$ such that $\mathbf{M}\mathbf{z} = \mathbf{e}_i$. The server can compute the inner product between $\mathbf{z}$ and its vector of observed messages $\mathbf{c} = \mathbf{M}^\top \mathbf{v}$. We can therefore expand it:

$$\mathbf{z}^\top \mathbf{c} = \mathbf{z}^\top (\mathbf{M}^\top \mathbf{v}) = (\mathbf{M}\mathbf{z})^\top \mathbf{v}.$$

Substituting $\mathbf{M}\mathbf{z} = \mathbf{e}_i$ gives:

$$\mathbf{z}^\top \mathbf{c} = \mathbf{e}_i^\top \mathbf{v}.$$

Since the inner product $\mathbf{e}_i^\top \mathbf{v}$ projects onto the i -th index of the unknown vector $\mathbf{v}$, one obtains the local model update x_i , and the claim follows. $\square$

Lemma 5.2. *Let $\mathbf{M} \in \mathbb{Z}_q^{m(m+1) \times 2m}$ be the incidence matrix described in Equation (4). For $m \geq 2$, the linear system:*

$$\mathbf{M}\mathbf{z} = \mathbf{e}_i$$

has no solution for any choice of $i \in [1, \dots, m(m+1)]$.

PROOF. By contradiction assume there exists a vector $\mathbf{z} \in \mathbb{Z}_q^{2m}$ such that $\mathbf{M}\mathbf{z} = \mathbf{e}_i$. Denoting with $\mathbf{M}_i$ the matrix obtained from $\mathbf{M}$ deleting the i -th row, the system:

$$\mathbf{M}_i \mathbf{z} = \mathbf{0} \tag{5}$$

admits a non-trivial solution $\mathbf{z}$. Consider the set of $2m$ rows indexed by $R_0 = \{x_1, \dots, x_m, k_1^1, \dots, k_m^m\}$. The $2m \times 2m$ submatrix formed by these rows has the block triangular structure:

$$\mathbf{S}_0 = \begin{bmatrix} \mathbb{I}_m & \mathbf{0} \\ \mathbb{I}_m & \mathbb{I}_m \end{bmatrix},$$

where $\mathbb{I}_m$ is the identity of order m . The determinant of $\mathbf{S}_0$ can be easily computed:

$$\det(\mathbf{S}_0) = \det(\mathbb{I}_m) \cdot \det(\mathbb{I}_m) = 1.$$

Assume the i th row is removed from $\mathbf{M}$ to obtain $\mathbf{M}_i$. The following cases can occur:

- (1) Case 1: $i \notin R_0$. All $2m$ rows of R_0 remain in $\mathbf{M}_i$, so $\mathbf{S}_0$ is a submatrix of $\mathbf{M}_i$ with $\det(\mathbf{S}_0) = 1$, invertible in $\mathbb{Z}_q$. Therefore, $\mathbf{M}_i \in \mathbb{Z}_q^{(m(m+1)-1) \times 2m}$ has full McCoy rank [10, Definition 4.10].
- (2) Case 2: i corresponds to the row indexed by k_j^j for some $j \in [1, \dots, m]$. We replace the row indexed by k_j^j with the row k_ℓ^j for any $\ell \neq j$. In this case, the resulting submatrix is lower-triangular:

$$\mathbf{S}_1 = \begin{bmatrix} \mathbb{I}_m & \mathbf{0}_n \\ A & \mathbb{I}_m \end{bmatrix}$$

with $A \in \mathbb{Z}_q^{m \times m}$. Therefore $\mathbf{S}_1$ is a submatrix of $\mathbf{M}_i$ having determinant 1, and hence $\mathbf{M}_i$ is a full-rank matrix over $\mathbb{Z}_q$.

- (3) Case 3: i corresponds to the row indexed by x_j for some $j \in [m]$. In this case, we construct $\mathbf{S}_2$ using the remaining $m-1$ weight rows $\{x_p\}_{p \neq m}$, all m key rows $\{k_p^p\}_{p \in [m]}$, and one off-diagonal key row k_ℓ^j ($\ell \neq j$).

Subtracting row k_ℓ^j from row k_j^j , we perform row operations transforming $\mathbf{S}_2$ into a block lower-triangular matrix:

$$\mathbf{S}_2 \sim \begin{bmatrix} C & \mathbf{0}_n \\ \mathbb{I}_m & \mathbb{I}_m \end{bmatrix},$$

where C is a lower-triangular matrix with 1s on its principal diagonal. Thus, $\det(\mathbf{S}_2) = 1$ and $\mathbf{M}_i$ is a full-rank.

In all cases, $\mathbf{M}_i$ has McCoy rank $2m$ [10]. Consequently, from [10, Theorem 5.3] the homogeneous system (5) admits the unique trivial $\mathbf{z} = \mathbf{0}$, yielding a contradiction. $\square$

Theorem 5.3. *In the honest-but-curious model, the protocol guarantees privacy against both the server and colluding clients:*

- **Server Privacy:** *The server learns no information about any individual client update x_i beyond the global aggregate $\sum_{i=1}^n x_i$.*
- **Client Privacy:** *Individual non-colluding clients gain no information about the local model updates of other clients.*
- **Collusion Resilience:** *Any collusion of at most $t < \tau$ clients (possibly in collusion with the server), learns no information about the private updates of the remaining honest clients, where τ is the reconstruction threshold of the underlying Shamir secret sharing scheme.*

PROOF. The statement follows from Lemma 5.1 and Lemma 5.2. For a colluding subset of clients the adversary knows their private weights and pairwise keys. By removing from $\mathbf{M}$ the rows corresponding to these variables we reduce ourselves to the proof of Lemma 5.1 and Lemma 5.2 for a submatrix $\mathbf{M}'$. Since $t < \tau$, $\mathbf{M}'$ is full-rank: from Lemma 5.2, $\mathbf{M}'\mathbf{z} = \mathbf{e}_i$ has no non-trivial solution for any honest client i , proving that no linear combination of observations can isolate an honest update x_i . $\square$

5.2 Computational Privacy

We define privacy via an indistinguishability game $\text{PRIV}_{\mathcal{A}}^b$, played between a challenger and an adversary $\mathcal{A}$ corrupting the server and a coalition $T \subseteq [1, \dots, n]$ of clients, $|T| < \tau$.

- (1) $\mathcal{A}$ selects two assignments of honest-client updates, $\{x_i^0\}_{i \notin T}$ and $\{x_i^1\}_{i \notin T}$, with $\sum_{i \notin T} x_i^0 = \sum_{i \notin T} x_i^1$, together with the honest clients' key material generated per the protocol.
- (2) The challenger samples $b \leftarrow \{0, 1\}$, runs the protocol honestly with honest-client inputs $\{x_i^b\}_{i \notin T}$, and gives $\mathcal{A}$ the full transcript: all ciphertexts c_j^ℓ , all (y_ℓ, σ_ℓ) , and all values the server reconstructs during recovery.
- (3) $\mathcal{A}$ outputs a guess b' .

$\text{Adv}_{\mathcal{A}}^{\text{PRIV}} := |\Pr[b' = b] - \frac{1}{2}|$. The protocol is (τ, t) -private if $\text{Adv}_{\mathcal{A}}^{\text{PRIV}}$ is negligible for every $\mathcal{A}$.

Theorem 5.4. Assume Enc/Dec (the ML-KEM–AEAD hybrid encryption, Section 2.3) is IND-CCA2-secure, and PRG is indistinguishable from uniform, both against the adversary class under consideration. Then for every $\mathcal{A}$ in the PRIV game there exist $\mathcal{B}_1, \mathcal{B}_2$ of comparable running time such that:

$$\text{Adv}_{\mathcal{A}}^{\text{PRIV}} \leq \text{Adv}_{\text{Enc/Dec}}^{\text{IND-CCA2}}(\mathcal{B}_1) + \text{Adv}^{\text{PRG}}(\mathcal{B}_2).$$

PROOF. Fix a single model coordinate (the argument applies coordinate-wise, since masking is applied component-wise). Let:

$$G_0 = \text{PRIV}_{\mathcal{A}}^b$$

be the real game, for the challenger's bit b .

$G_0 \rightarrow G_1$ (**ciphertext confidentiality**). For every pair of honest clients i, j , replace the plaintext of

$$c_j^i = \text{Enc}(\text{pub}_i, (k_j^i, \text{share}_b(i, j), \text{share}_u(i, j)))$$

with an encryption of a fixed dummy message of equal length; ciphertexts touching T remain genuine. Any $\mathcal{A}$ whose guessing probability changes between G_0 and G_1 yields a reduction $\mathcal{B}_1$ against IND-CCA2 of Enc/Dec, so:

$$|\Pr[\mathcal{A} \rightarrow 1 \mid G_0] - \Pr[\mathcal{A} \rightarrow 1 \mid G_1]| \leq \text{Adv}_{\text{Enc/Dec}}^{\text{IND-CCA2}}(\mathcal{B}_1).$$

In G_1 , every k_j^i for an honest-honest pair is information-theoretically absent from $\mathcal{A}$'s view except through $\text{PRG}(k_j^i)$ inside y_i, σ_j .

$G_1 \rightarrow G_2$ (**mask pseudorandomness**). For every honest-honest pair i, j , replace $\text{PRG}(k_j^i)$ with an independently, freshly sampled uniform vector. A standard hybrid over honest-honest pairs gives a reduction $\mathcal{B}_2$ against PRG's indistinguishability from uniform, so:

$$|\Pr[\mathcal{A} \rightarrow 1 \mid G_1] - \Pr[\mathcal{A} \rightarrow 1 \mid G_2]| \leq \text{Adv}^{\text{PRG}}(\mathcal{B}_2).$$

G_2 : **the challenger's bit is information-theoretically hidden**. In G_2 , every pairwise-mask value k_j^i contributed by an honest party is uniform and derived independently, exactly the hypothesis under which Lemma 5.1 and Lemma 5.2 were established. By Lemma 5.2, for $n \geq 2$ the system $\mathbf{Mz} = \mathbf{e}_\ell$ has no solution for any honest client ℓ ; by Lemma 5.1, this means no linear combination of the transcript $\{y_\ell, \sigma_\ell\}$ isolates any individual honest x_ℓ . Since the transcript's only dependence on $\{x_\ell\}_{\ell \notin T}$ is through the values $\{y_\ell\}$, each masked by an independent uniform quantity not appearing elsewhere except in the single combination summing to $\sum_{\ell \notin T} x_\ell$ (the aggregate), and since $\mathcal{A}$'s two challenge assignments x^0, x^1 were constrained to share this same sum, the joint distribution of the transcript in G_2 is identical under $b = 0$ and $b = 1$:

$$\Pr[\text{transcript} \mid G_2, b = 0] = \Pr[\text{transcript} \mid G_2, b = 1].$$

Hence $\Pr[\mathcal{A} \rightarrow 1 \mid G_2, b = 0] = \Pr[\mathcal{A} \rightarrow 1 \mid G_2, b = 1]$, giving $\text{Adv}_{\mathcal{A}}^{G_2} = 0$ exactly. Summing the two computational hops via the triangle inequality gives the claimed bound. $\square$

COROLLARY 5.5 (CLIENT-ONLY COLLUSIONS). *The protocol is private against any collusion of at most $t < \tau$ colluding clients without a corrupt server.*

PROOF. The adversary considered in Theorem 5.4 is strictly stronger, as it obtains from the server all information available during the aggregation and recovery, since all the messages are sent through a secure channel. The claim therefore follows immediately from Theorem 5.4. $\square$

6 CONCLUSION AND FUTURE WORKS

In this paper, we presented SecAgg++, a post-quantum secure aggregation protocol for federated learning in the single-server setting which incorporates post-quantum cryptographic primitives into masking-based secure aggregation while maintaining the communication and computational efficiency of classical protocols [4, 8]. We implemented the protocol in Flower and proved privacy against an honest-but-curious server and colluding corrupt clients. The primary trade-off of our construction lies in its dropout flexibility: while clients may drop before submitting their masked updates, surviving participants must remain online through the initial steps of the unmasking phase. An interesting direction for future work is therefore to design a post-quantum masking-based secure aggregation protocol that recovers the full dropout flexibility of SecAgg. Other possible directions include extending the security analysis to malicious server model, experimentally comparing SecAgg++ with alternative post-quantum secure aggregation approaches, including homomorphic encryption [5, 20] or functional encryption [34], and developing a system-level implementation in a language such as C++ or Rust for better performances.

Acknowledgments

This work was supported by Ministero delle Imprese e del Made in Italy (IPCEI Cloud DM 27 giugno 2022 – IPCEI-CL-0000007) and European Union (Next Generation EU).

References

- [1] 2026. gRPC: A high performance, open source universal RPC framework. <https://grpc.io/>
- [2] Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Thomas Pöppelmann, Peter Schwabe, and Douglas Stebila. [n.d.]. Algorithm Specifications and Supporting Documentation. Version 1 ([n.d.]), 1–44.
- [3] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. 2016. Post-quantum key Exchange—A new hope. In *25th USENIX security symposium (USENIX Security 16)*. 327–343.
- [4] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrede Lepoint, and Mariana Raykova. 2020. Secure Single-Server Aggregation with (Poly) Logarithmic Overhead. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 1253–1269.
- [5] James Bell-Clark, Adrià Gascón, Baiyu Li, Mariana Raykova, and Philipp Schoppmann. 2025. Willow: secure aggregation with one-shot clients. In *Annual International Cryptology Conference*. Springer, 285–318.
- [6] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchu Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Hei Li Kwing, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. 2020. Flower: A Friendly Federated Learning Research Framework. *arXiv preprint arXiv:2007.14390* (2020).
- [7] Sebastian Bitzer, Maximilian Egger, Mumin Liu, and Antonia Wachter-Zeh. 2026. Post-Quantum Secure Aggregation via Code-Based Homomorphic Encryption. *arXiv preprint arXiv:2601.13031* (2026).

- [8] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1175–1191.
- [9] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. 2016. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 1006–1018.
- [10] William Clough Brown. 1993. Matrices over commutative rings. (No Title) (1993).
- [11] David Byrd and Antigoni Polychroniadou. 2020. Differentially private secure multi-party computation for federated learning in financial applications. In *Proceedings of the first ACM international conference on AI in finance*. 1–9.
- [12] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2019. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 395–412.
- [13] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*. Springer, 409–437.
- [14] Morris Dworkin. 2001. *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. NIST Special Publication 800-38A. National Institute of Standards and Technology. doi:10.6028/NIST.SP.800-38A
- [15] Morris Dworkin. 2015. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Federal Information Processing Standards Publication (FIPS PUB) 202. National Institute of Standards and Technology. doi:10.6028/NIST.FIPS.202
- [16] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang, et al. 2018. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. *Submission to the NIST's post-quantum cryptography standardization process* 36, 5 (2018), 1–75.
- [17] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 212–219.
- [18] Tayyeb Jahani-Nezhad, Mohammad Ali Maddah-Ali, Songze Li, and Giuseppe Caire. 2022. Swiftagg: Communication-efficient and dropout-resistant secure aggregation for federated learning with worst-case security guarantees. In *2022 IEEE international symposium on information theory (ISIT)*. IEEE, 103–108.
- [19] Panos Kampanakis and Will Childs-Klein. 2024. The impact of data-heavy, post-quantum TLS 1.3 on the Time-To-Last-Byte of real-world connections. *Cryptology ePrint Archive*, Paper 2024/176. doi:10.14722/madweb.2024.23010
- [20] Harish Karthikeyan and Antigoni Polychroniadou. 2025. OPA: one-shot private aggregation with single client interaction and its applications to federated learning. In *Annual International Cryptology Conference*. Springer, 319–353.
- [21] Hugo Krawczyk. 2010. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In *Advances in Cryptology – CRYPTO 2010*, Tal Rabin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 631–648.
- [22] Mohamad Mansouri, Melek Önen, Wafa Ben Jaballah, and Mauro Conti. 2023. Sok: Secure aggregation based on cryptographic schemes for federated learning. *Proceedings on Privacy Enhancing Technologies (PoPETs)* (2023), 140–157. doi:10.56553/popets-2023-0009
- [23] David McGrew. 2008. An Interface and Algorithms for Authenticated Encryption. RFC 5116. doi:10.17487/RFC5116
- [24] José A Montenegro, Ruben Rios, and Javier Bonilla. 2025. Comparative analysis of post-quantum handshake performance in QUIC and TLS protocols. *Computer Networks* (2025), 111957.
- [25] National Institute of Standards and Technology. 2024. *Module-Lattice-Based Digital Signature Standard*. Federal Information Processing Standards Publication (FIPS PUB) 204. National Institute of Standards and Technology. doi:10.6028/NIST.FIPS.204
- [26] NIST. 2024. FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard. doi:10.6028/NIST.FIPS.203
- [27] Christian Paquin, Douglas Stebila, and Goutam Tamvada. 2019. Benchmarking Post-Quantum Cryptography in TLS. *Cryptology ePrint Archive*, Paper 2019/1447. <https://eprint.iacr.org/2019/1447>
- [28] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)* 56, 6 (2009), 1–40.
- [29] Sinem Sav, Apostolos Pyrgelis, Juan R Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. 2020. POSEIDON: Privacy-preserving federated neural network learning. *arXiv preprint arXiv:2009.00349* (2020).
- [30] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
- [31] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.
- [32] Jinhyun So, Başak Güler, and A Salman Avestimehr. 2021. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory* 2, 1 (2021), 479–489.
- [33] Jinhyun So, Chaoyang He, Chien-Sheng Yang, Songze Li, Qian Yu, Ramy E Ali, Basak Guler, and Salman Avestimehr. 2022. Lightsecagg: a lightweight and versatile design for secure aggregation in federated learning. *Proceedings of Machine Learning and Systems* 4 (2022), 694–720.
- [34] Enrico Sorbera, Federica Zanetti, Giacomo Brandi, Alessandro Tomasi, Roberto Doriguzzi-Corin, and Silvio Ranise. 2025. Adaptive Federated Learning with Functional Encryption: A Comparison of Classical and Quantum-safe Options. In *2025 3rd International Conference on Foundation and Large Language Models (FLLM)*. 1188–1195. doi:10.1109/FLLM67465.2025.11391241
- [35] Markus Sosnowski, Florian Wiedner, Eric Hauser, Lion Steger, Dimitrios Schoini-anakis, Sebastian Gallenmüller, and Georg Carle. 2023. The Performance of Post-Quantum TLS 1.3. In *Companion of the 19th International Conference on Emerging Networking EXperiments and Technologies* (Paris, France) (CoNEXT 2023). Association for Computing Machinery, New York, NY, USA, 19–27. doi:10.1145/3624354.3630585
- [36] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. 2019. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE conference on computer communications*. IEEE, 2512–2520.
- [37] Shisong Yang, Yuwen Chen, Shanshan Tu, and Zhen Yang. 2022. A post-quantum secure aggregation for federated learning. In *Proceedings of the 2022 12th international conference on communication and network security*. 117–124.
- [38] Li Zhang, Jianbo Xu, Pandi Vijayakumar, Pradip Kumar Sharma, and Uttam Ghosh. 2022. Homomorphic encryption-based privacy-preserving federated learning in IoT-enabled healthcare system. *IEEE transactions on network science and engineering* 10, 5 (2022), 2864–2880.
- [39] Xia Zhang, Haitao Deng, Rui Wu, Jingjing Ren, and Yongjun Ren. 2024. PQSF: post-quantum secure privacy-preserving federated learning. *Scientific Reports* 14, 1 (2024), 23553.
- [40] Bian Zhu and Ling Niu. 2025. A privacy-preserving federated learning scheme with homomorphic encryption and edge computing. *Alexandria Engineering Journal* 118 (2025), 11–20.
- [41] Ruozhou Zuo, Haibo Tian, Zhiyuan An, and Fangguo Zhang. 2022. Post-quantum privacy-preserving aggregation in federated learning based on lattice. In *International Symposium on Cyberspace Safety and Security*. Springer, 314–326.