Fault Tolerant and Malicious Secure Federated Learning

Ferhat Karakoç[1], Alptekin Küpçü[2], and Melek Önen[3]

[1] Ericsson Research, İstanbul, Turkey
ferhat.karakoc@ericsson.com
[2] Koç University, İstanbul, Turkey
akupcu@ku.edu.tr
[3] EURECOM, Sophia-Antipolis, France
melek.onen@eurecom.fr

Abstract. Federated learning (FL) is one of the promising collaborative machine learning methods finding many usage application scenarios in different domains such as healthcare ([19]) and/or telecommunication (5G, 5G beyond and 6G [25]). It also enhances privacy by allowing users to contribute to the global model training without sharing their training data. However, the local model updates exposed by users can still leak sensitive information. To prevent such leakage, secure aggregation protocols are utilized to hide the individual local model updates from the aggregator. Enhancing privacy in this way creates an open door for security attacks because the server is no longer able to analyze received updates for detection of poisoning type of attacks. Although there are considerable number of studies that address the privacy and security aspects individually, solutions against the combination of these attacks have started to appear recently in a few studies. When we add some additional requirements such as aggregation unforgeability and robustness against user drop-outs, the number of solutions becomes very limited. Most of the proposals addressing all these aspects at the same time require two or more non-colluding aggregators, which may not be a realistic assumption in most of the use cases. To address this gap, we introduce new secure aggregation protocols involving one aggregator only. Each proposed protocol addresses a subset of the requirements where as the final one, FULLSA3, is secure against malicious clients and robust against user drop-outs. As a side contribution, we design a new batch oblivious range verification protocol.

Keywords: Federated learning · secure aggregation · privacy · poisoning attacks · oblivious range verification.

1 Introduction

With the advent of enablers in data collection and data processing, and because the utilization of huge amount of data brings real value in various aspects such as better user experience, reduction in operational costs, and sustainability, many

applications and frameworks have started to adopt the data-driven design (such as 6G technologies [25], healthcare solutions [19]). Accordingly, federated learning (FL) has become one of the preferred machine learning techniques whereby data is no longer collected and stored in a central server and remains at users' premises.

Although FL enhances privacy protection by allowing the users (i.e., clients or participants) to contribute to the global model training without exposing their local training data (by training the model locally and sharing model parameters updates only), the model updates that they share during training still leak information about the local training data of the user [24]. To address such vulnerability, secure aggregation protocols such as [2], are recently proposed, whereby the FL server receives privacy-protected model updates and aggregates them without having access to them. Nevertheless, these protocols prevent the FL server, also called the aggregator, from performing some analysis on local model updates to detect anomalies in the local model updates caused by, for example, some model poisoning attack attempts or some unexpected behaviors of users. Thus, addressing the combination of security and privacy aspects is needed. Some additional requirements such as validation of the aggregation result and robustness against user drop-outs can also raise during the execution of the federated learning process. Although there is a significant number of studies in the literature about the security and privacy aspects of federated learning (e.g., [23, 12]), we can only witness a very limited number of attempts to solve all these expectations, at once.

In this paper, we aim at addressing all these requirements through incrementally designed three new secure aggregation protocols. These protocols consider the following goals:

- **aggregator obliviousness**: the aggregator (i.e., the FL server) should not be able to learn individual local model updates;
- **aggregate unforgeability**: the aggregator should not be able to alter the aggregation result.
- **fault tolerance**: the FL procedure should continue even if some users drop out.
- **input-range unforgeability**: the users should not be able to input local model update parameters that are outside a range defined by the aggregator (to resist poisoning).
- **single server model**: the architecture requires only one FL server (i.e., does not require two or more non-colluding FL servers).

While the first protocol, FULLSA1, addresses aggregation obliviousness, input-range unforgeability and fault-tolerance, FULLSA2 improves it by additionally ensuring aggregate unforgeability. Finally, the last protocol, FULLSA3, considers all design goals altogether. The security model also becomes stronger, incrementally. Indeed, FULLSA1 assumes a setting with a semi-honest aggregator and malicious users; FULLSA2 considers a semi-honest aggregator who may lie about the aggregate result, and, finally FULLSA3 is the last protocol where all parties are considered malicious. All the proposed protocols work under the

Table 1. Literature review on single-server secure aggregation schemes. Notation: * (malicious aggregator for input privacy only) . ** (users send their shares each time period). *** (with ACORN-robust). SH (semi-honest), M (Malicious).

Solution	aggregate unforgeability	input-range unforgeability	adversary model		fault tolerance	aggregation efficiency
			client	server		
Joye-Libert [7]	no	no	SH	SH	no	-
PUDA [12]	yes	no	SH	M	no	no
Bonawitz et al. [2]	no	no	SH	SH	yes	yes
Mansouri et al. [15]	no	no	SH	SH	yes	-
Karakoc et al. [10]	yes	yes	M	SH	no	no
RoFL [13]	no	yes	M	SH	yes	-
EIFFEL [21] *	yes	yes	M	M*	yes **	-
ACORN [1]	no	yes	M ***	M*	yes	yes
Our FULLSA1	no	yes	M	SH	yes	yes
Our FULLSA2	yes	yes	M	M*	yes	yes
Our FULLSA3	yes	yes	M	M	yes	yes

single-server model. Another by-default goal is naturally performance. We show that our protocols are scalable in the number of users.

Overview of our protocols. To develop FULLSA1 and FULLSA2, we have started by studying the secure aggregation solution in [10] and improve it by addressing fault-tolerance and offering better performance. Indeed, we first improve the oblivious programmable pseudo-random function introduced in [10] to convert it into a batch oblivious range verification protocol. Instead of obliviously verifying the range of each model parameter one-by-one and executing one oblivious transfer protocol per parameter, the verification is performed on a vector-basis. We also propose another optimization to reduce the number of oblivious transfer operations (which significantly improves the performance) whenever the threshold values defined for input-range verification consist of some specific values. Note that, once again this oblivious programmable pseudo-random function can be used as a standalone solution, as described in [10].

On the other hand, to address fault-tolerance, we utilize the idea in [15] making the Joye-Libert encryption scheme [7] robust against user drop-outs. Whenever some users drop out, other online users are expected to encrypt a zero-value, collaboratively, on behalf of the dropped users. In [15], some of the malicious users may still try to alter this process resulting in the encryption of a non-zero value and hence falsifying the aggregate result; To prevent this kind of malicious behavior, we add a verification step involving bilinear pairings so that the aggregator can validate the correct encryption of '0'.

To move from FULLSA1 to its extended version, FULLSA2, we use the idea in [12] to additionally ensure aggregation unforgeability whereby homomorphic signature-based bilinear pairings are used.

Finally, since the underlying batch oblivious range verification protocol is secure only if the server is semi-honest, to achieve security against a malicious server we adopt the approach of [13] and replace the oblivious verification protocol with a range zero-knowledge proof (ZKP).

Related work. In recent years, there have been several studies proposing new secure aggregation solutions addressing the security and privacy aspects simul-

taneously. Table 1 presents a brief comparison of our protocol with the existing solutions working in the single server model. This table does not include the following studies which cannot be easily compared:

- The studies [6, 8] propose the usage of multi-hop communication for anonymization of the local model update ownership. Since the aggregator can assess the plaintext local model updates, the server can analyze the received local model update for the detection of security attacks.
- Some other designs such as ELSA [18] and Prio [5] require two non-colluding servers so that these servers cannot access the individual local model updates thanks to the underlying two-party computation protocol but can perform operations on the secret shares of the local model updates in a collaborative way. Both ELSA and Prio, by-design, assume that at least one server is semi-honest and hence do not consider a fully malicious setting. Therefore, we only compare with single-server solutions.
- Flamingo [14] works on a single server model but requires a small group of clients (named as "encryptors") communicating with the server for decryption of the results.

As shown in Table 1, earlier solutions, namely [7, 11, 2, 10, 13] consider that at least one player is semi-honest and some of them do not guarantee fault-tolerance against user dropouts. On the other hand, EIFFEL [21] and ACORN [1] consider a malicious aggregator only for input privacy and not for the integrity of the aggregator. While our FULLSA2 is comparable to EIFFEL and ACORN, EIFFEL involves clients for reconstruction and ACORN does not ensure aggregate unforgeability.

2 Preliminaries

Table 2 provides the notation used throughout the paper.

Threshold Joye-Libert scheme: This scheme introduced in [15], improving the Joye-Libert scheme [7], consists of the following six algorithms.

- $Setup(1^\kappa) \rightarrow (pp, \{ek_i\}_{i\in[n]}, ek_A)$: Takes security parameter κ in unary as input and outputs randomly selected n keys $\{ek_i \mid 1 \leq i \leq n\}$ and $ek_A = -\sum_1^n ek_i$ and public parameter $pp = (N, H_1)$ where $N = p \times q$, p and q are randomly selected values and H_1 is a cryptographic hash function.
- $SKShare(pp, ek_i, t, n) \rightarrow \{(j, [ek_i]_j) \mid 1 \leq j \leq n\}$: outputs secret sharing of ek_i.
- $ShareProtect(pp, \{[ek_i]_j\}, \tau) \rightarrow [y_i']_j$: outputs encryption of '0' under the secret share $[ek_i]_j$.
- $ShareCombine(\{j, [c_{i0}]_j\}_{j\neq i}, \tau.) \rightarrow c_{0i,\tau}$ outputs the reconstructıon of the encryption of '0' under ek_i.
- $Protect(pp, ek_i, \tau, x_{i,\tau}) \rightarrow c_{i,\tau}$:

$$c_{i,\tau} \leftarrow (1 + x_{i,\tau}N)H_1(\tau)^{ek_i} \mod N^2$$

- $Agg(pp, ek_A, \tau, \{c_{i,\tau}\}_{i\in[n]}) \rightarrow \sum_1^n x_{i,\tau}$:

n, n_D, n_A	: number of users, dropped users, available users, respectively
U, U_D, U_A	: set of all users, set of dropped users, set of available users, respectively
$\mathcal{U}_i$	: user number i from set U
$x_{i,j,t}$	: j-th model parameter value at time-period t from client $\mathcal{U}_i$
$X_{i,t}$	: set of items of user $\mathcal{U}_i$ such that $X_{i,t} = \{x_{i,j,t}\}$
n_X	: number of data points to be aggregated (size of X)
$[x]_i$	: i-th secret share of x
$x[i]$	: i-th most significant bit of x where $x[0]$ is the least significant bit
ℓ	: Bit length of data points (x)
$A[i]$	: i-th element in the set A
g_1, g_3	: generator of group $\mathbb{G}_1$
g_2	: generator of group $\mathbb{G}_2$
e	: bilinear pairing such that $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.
H_1	: hash function outputting in mod N^2.
H_2	: hash function outputting in $\mathbb{G}_1$.
ek_i, ek'_i	: encryption key for $\mathcal{U}_i$, used in Joye-Libert encryption scheme [7] and in Shi-Chan encryption scheme [23], respectively
tk_i, tk'_i	: tag key for $\mathcal{U}_i$ used in tag computation for input range validation and for aggregator obliviousness, respectively
ek_A, ek'_A	: decryption/aggregation key for $\mathcal{A}$ such that $ek_A = -\sum ek_i$ and $ek'_A = -\sum ek'_i$
Λ	: set of threshold values for model parameters such that $\Lambda = \{\lambda_j \mid 1 \leq j \leq n_X\}$
$[ek_d]_i$	: i-th secret share of the key of d-th user

Table 2. Notation

$$c_\tau \leftarrow \Pi_1^n c_{i,\tau} = (1 + N \sum_1^n x_{i,\tau}) H_1(\tau)^{\sum_1^n ek_i}$$

$$(H_1(\tau)^{ek_A} c_\tau - 1) N^{-1} = \sum_1^n x_{i,\tau} \mod N$$

PUDA: PUDA [12] is a solution providing a mechanism to validate the aggregation result. The steps of the solution are given in Protocol 1 by following the convention in [10]. The solution needs a trusted key dealer $\mathcal{KD}$ for generation of the following parameters and keying materials in the setup phase.

- Cyclic groups $\mathbb{G}_1$ with generator g_1, $\mathbb{G}_2$ with generator g_2, $\mathbb{G}_T$, each of prime order p, a bilinear mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.
- A hash function $H_2 : \{0,1\}^* \to \mathbb{G}_1$.
- Encryption keys $ek'_i \in \mathbb{Z}_p$ for each user and decryption key $ek'_A = -\sum_i^n ek'_i$ known by the aggregator.
- Tag keys $tk'_i \in \mathbb{Z}_p$ for each user.
- Verification key $VK = (vk_1, vk_2) = (g_2^{\sum tk'_i}, g_2^a)$ known by the aggregator.
- A group element g_1^a for the users where a is a randomly generated number. g_1^a kept secret from the aggregator.

Protocol 1 PUDA [12, 10]

Parameters. $H_2, p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e$.
Inputs. $\mathcal{U}_i$ inputs $x_{i,t}$, tk'_i, ek'_i and g_1^a. $\mathcal{A}$ inputs ek'_A. $\mathcal{DA}$ inputs the verification key VK.
Outputs. $\mathcal{A}$ outputs $sum_t = \sum_{i=1}^{n} x_{i,t}$. $\mathcal{DA}$ outputs the result of the verification.
Protocol steps:

1. Each $\mathcal{U}_i$ computes ciphertext $c_{i,t} = H_2(t)^{ek'_i} g_1^{x_{i,t}}$ and tag value $\sigma_{i,t} = H_2(t)^{tk'_i}(g_1^a)^{x_{i,t}}$, and then sends them to $\mathcal{A}$
2. $\mathcal{A}$ computes the sum from $V_t = (\prod_i c_{i,t}) H_2(t)^{ek'_A} = g_1^{sum_t}$ and the aggregated tag $\sigma_t = \prod_i \sigma_{i,t} = H_2(t)^{\sum_i tk'_{i,t}} (g_1^a)^{sum_t}$, and then outputs sum_t and σ_t and sends them to $\mathcal{DA}$.
3. $\mathcal{DA}$ check the equation $e(\sigma_t, g_2) == e(H_2(t), vk_1) e(g_1^{sum_t}, vk_2)$ and outputs the check result.

Karakoc et al. secure aggregation protocol: Karakoc et al. proposed a solution [10] to convert PUDA into a secure aggregation protocol having the input-range unforgeability. For that purpose, they introduce a threshold oblivious programmable pseudo-random function protocol where the receiver inputs a value x and the sender inputs a threshold value λ and a secret value a. At the end of the protocol, the receiver learns the encryption result of g_1^{ax} if $x \leq \lambda$. Otherwise the protocol aborts. The sender outputs the key used in the encryption of g^{ax}. Protocol 2 presents the steps of their solution.
Oblivious Transfer (OT) [17] and its extensions can be used for the receiver to receive a subset of the sender's items. We employ Functionality 1 for simplicity, but employ OT-extension implementation [20].

3 Our solutions

In this section, we introduce three new FaULt toLerant Secure Aggregation protocols called FULLSA1, FULLSA2, and FULLSA3 for Functionality 4. We also present a new Batch Oblivious Range Verification protocol, realizing Functionality 2, which is utilized by FULLSA1 and FULLSA2. While FULLSA1 and FULLSA2 consider a semi-honest aggregator, FULLSA3 replaces this building block with range ZKPs in order to be secure against a malicious aggregator.

The first protocol, FULLSA1, realizes the functionality securely against the following attacks:

- The semi-honest aggregator tries to learn information about the data points of the users;
- One or more malicious users try to poison the aggregation by providing data point values which are greater than threshold values set by the aggregator;
- One or more users can drop out (accidentally or maliciously).

FULLSA2 improves FULLSA1 to make the protocol secure against the semi-honest aggregator who wants to alter the aggregation result after the computa-

Protocol 2 Karakoc et al. secure aggregation [10]

Parameters. $H_2, p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e$
Inputs. $\mathcal{U}_i$ inputs $x_{i,t}$, tk'_i, ek'_i. $\mathcal{A}$ inputs ek'_A and vk_1.
Outputs. $\mathcal{A}$ aborts the protocol if at least one $x_{i,t}$ is larger than λ, otherwise outputs $sum_t = \sum_{i=1}^{n} x_{i,t}$.
Protocol steps:

1. Each $\mathcal{U}_i$ computes $c_{i,t} = H_2(t)^{ek'_i} g_1^{x_{i,t}}$ and sends the result to $\mathcal{A}$.
2. $\mathcal{A}$ chooses a_t randomly, runs Protocol 3 with each $\mathcal{U}_i$. $\mathcal{U}_i$ learns $o_{oprf_{i,t}} = k_{opprf_{i,t}}(g_1^a)^{x_{i,t}}$ if $x_{i,t} \leq \lambda$, otherwise $o_{oprf_{i,t}}$ becomes a random number.
3. Each $\mathcal{U}_i$ computes the tag value $\sigma_{i,t} = H_2(t)^{tk'_i} o_{oprf_{i,t}}$ and sends the result to $\mathcal{A}$.
4. $\mathcal{A}$, computes sum_t from $V_t = (\prod_i c_{i,t}) H_2(t)^{ek'_A} = g_1^{sum_t}$, and computes $\sigma_t = \prod_i (\sigma_{i,t}/k_{opprf_{i,t}}) = H_2(t)^{\sum_i tk'_{i,t}} (g_1^a)^{sum_t}$.
5. $\mathcal{A}$ checks the equation $e(\sigma_t, g_2) == e(H_2(t), vk_1) e(g_1^{sum_t}, g_2^a)$
6. If the verification fails, $\mathcal{A}$ aborts the protocol. Otherwise $\mathcal{A}$ outputs sum_t.

Functionality 1 Oblivious Transfer $\mathcal{F}^{OT}$

Inputs. Sender inputs x_0, x_1, receiver inputs a bit b.
Outputs. Receiver outputs x_b, sender outputs nothing.

Functionality 2 Batch Oblivious Range Verification $\mathcal{F}^{BORF}$

Parameters. g, $\mathbb{G}$, n_X.
Inputs. P_2 inputs a set of items $X = \{x_j \mid 1 \leq j \leq n_X\}$, P_1 inputs a set of threshold values $\Lambda = \{\lambda_j \mid 1 \leq j \leq n_X\}$, a set of weights $W = \{w_j \mid 1 \leq j \leq n_X\}$, and a key k.
Outputs. If $\forall j\ x_j \leq \lambda_j$ then P_2 learns a masked weighted sum $wsum = k \times g^{\sum x_j w_j}$, otherwise P_2 learns $\perp$. P_1 outputs nothing.

tion of the aggregation. FULLSA3 improves FULLSA2 to make it secure against a malicious aggregator.

3.1 Batch Oblivious Range Verification

We first introduce a Batch Oblivious Range Verification protocol with Protocol 3, realizing Functionality 2, which is utilized by FULLSA1 and FULLSA2. The batch oblivious range verification protocol is based on the oblivious graph evaluation technique used for private set membership protocol in [4] and later utilized for oblivious equality testing in [9] and oblivious comparison protocol in [10]. Details of the protocol are presented in Protocol 3. This solution can also be used as a standalone building block for other purposes.

Figure 1 visualizes such a constructed graph with an example where $\lambda_1 = (100)$ and $\lambda_2 = (010)$. Also, Figure 2 illustrates potential message exchanges between the user and the aggregator when $X = \{(010), (001)\}$ and $\Lambda = \{(100), (010)\}$.

Protocol 3 Batch Oblivious Range Verification

Parameters. A generator g of a group $\mathbb{G}$ and the set size n_X of the receiver P_2.
Inputs. P_2 inputs a set of items $X = \{x_j \mid 1 \leq j \leq n_X\}$, P_1 inputs a set of threshold values $\Lambda = \{\lambda_j \mid 1 \leq j \leq n_X\}$, a set of weights $W = \{w_j \mid 1 \leq j \leq n_X\}$, and a key k.
Outputs. If $\forall j\ x_j \leq \lambda_j$ then P_2 learns $wsum = k \times g^{\sum x_j w_j}$.
Protocol steps:

1. P_1 prepares $(S^0_{j,i}, S^1_{j,i})$ for $1 \leq i \leq \ell$ and $1 \leq j \leq n_X$ as follows:
   (a) For $1 \leq j \leq n_X$ chooses symmetric keys $k^{<}_{j,\ell-1}$, $k^{=}_{j,\ell-1}$, $k^{>}_{j,\ell-1}$, and $ok_{j,\ell-1}$ randomly and a random value r_j and then computes $S^0_{j,\ell}$ and $S^1_{j,\ell}$ as follows:

$$S^0_{j,\ell} = \begin{cases} \{k^{=}_{j,\ell-1}, g^{0\times 2^{\ell-1} w_j} r_j^{ok_{j,\ell-1}}\} & \text{if } \lambda_j[\ell-1] = 0 \\ \{k^{<}_{j,\ell-1}, g^{0\times 2^{\ell-1} w_j} r_j^{ok_{j,\ell-1}}\} & \text{otherwise} \end{cases}$$

$$S^1_{j,\ell} = \begin{cases} \{k^{>}_{j,\ell-1}, g^{1\times 2^{\ell-1} w_j} r_j^{ok_{j,\ell-1}}\} & \text{if } \lambda_j[\ell-1] = 0 \\ \{k^{=}_{j,\ell-1}, g^{1\times 2^{\ell-1} w_j} r_j^{ok_{j,\ell-1}}\} & \text{otherwise} \end{cases}$$

   (b) For $i = \ell - 1$ to 1 and for each $j \in \{1, ..., n_X\}$ chooses four symmetric keys $k^{<}_{j,i-1}$, $k^{=}_{j,i-1}$, $k^{>}_{j,i-1}$, and $ok_{j,i-1}$ randomly and computes $S^0_{j,i}$ and $S^1_{j,i}$ as follows.
      – If $\lambda_j[i-1] = 0$,

$$S^0_{j,i} = \{E_{k^{<}_{j,i}}(k^{<}_{j,i-1}), E_{k^{=}_{j,i}}(k^{=}_{j,i-1}), E_{k^{>}_{j,i}}(k^{>}_{j,i-1}), g^{0\times 2^{i-1} w_j} r_j^{ok_{j,i-1}}\}$$

$$S^1_{j,i} = \{E_{k^{<}_{j,i}}(k^{<}_{j,i-1}), E_{k^{=}_{j,i}}(k^{>}_{j,i-1}), E_{k^{>}_{j,i}}(k^{>}_{j,i-1}), g^{1\times 2^{i-1} w_j} r_j^{ok_{j,i-1}}\}$$

      – Otherwise,

$$S^0_{j,i} = \{E_{k^{<}_{j,i}}(k^{<}_{j,i-1}), E_{k^{=}_{j,i}}(k^{<}_{j,i-1}), E_{k^{>}_{j,i}}(k^{>}_{j,i-1}), g^{0\times 2^{i-1} w_j} r_j^{ok_{j,i-1}}\}$$

$$S^1_{j,i} = \{E_{k^{<}_{j,i}}(k^{<}_{j,i-1}), E_{k^{=}_{j,i}}(k^{=}_{j,i-1}), E_{k^{>}_{j,i}}(k^{>}_{j,i-1}), g^{1\times 2^{i-1} w_j} r_j^{ok_{j,i-1}}\}$$

2. P_1 and P_2 run $\ell \times j$ oblivious transfer protocols where in the (j,i)-th OT P_1 inputs $(S^0_{j,i}, S^1_{j,i})$ and P_2 inputs $x_j[i-1]$ and P_2 learns $S^{x_j[i-1]}_{j,i}$.
3. P_2 first sets $result = \prod_{j=1}^{n_X} g^{2^{\ell-1} x_j[\ell-1] w_j} r_j^{ok_{j,\ell-1}}$ from the received messages $S^{x_j[\ell-1]}_{j,\ell}$ for $1 \leq j \leq n_X$. P_2 also sets $k'_{j,\ell-1}$ as the key in the received messages. Then performs the following operations for $i = \ell - 1$ to 1
   (a) P_2 sets $result = result \times \prod_{j=1}^{n_X} g^{2^{i-1} x_j[i-1] w_j} r_j^{ok_{j,i-1}}$ from the received messages $S^{x_j[i-1]}_{j,i}$ for $1 \leq j \leq n_X$. P_2 will also be able to decrypt only one of the ciphertexts in the received (j,i)-th message using $k'_{j,i}$ and sets $k'_{j,i-1}$ as the decryption result.
4. P_1 computes $mask = k \times \prod_{j=1}^{n_X} r_j^{-\sum_{i=0}^{\ell-1} ok_{j,i}}$ and divides $mask$ into n_X shares sk_j such that $\bigoplus sk_j = mask$. Also computes $\{E_{k^{<}_{j,0}}(sk_j) \mid 1 \leq j \leq n_X\}$ and $\{E_{k^{=}_{j,0}}(sk_j) \mid 1 \leq j \leq n_X\}$, and sends the encryption results to P_2.
5. P_2 will be able to decrypt one of the ciphertexts, only, for each j received in Step 4 using $k'_{j,0}$. Finally, after obtaining the secret shares sk_j via decryption, P_2 constructs $mask$ and computes $result = result \times mask$ and outputs $result$.

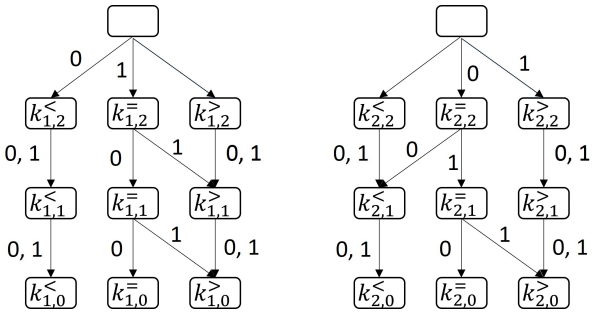

Fig. 1. Graph representation of key encryptions executed by the aggregator for $\lambda_1 = (100)$ (left) and $\lambda_2 = (010)$ (right).

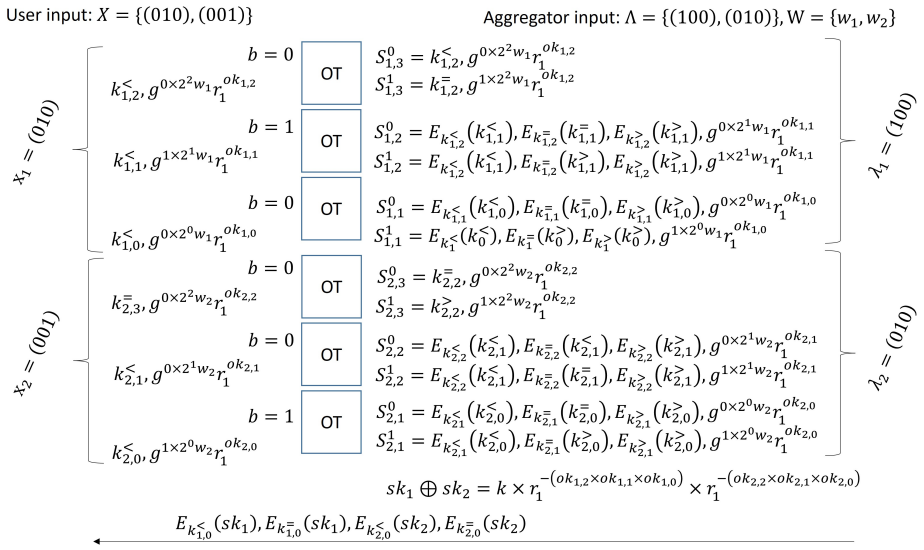

Fig. 2. Execution of Protocol 3 for $X = \{(010), (001)\}$ and $\Lambda = \{(100), (010)\}$.

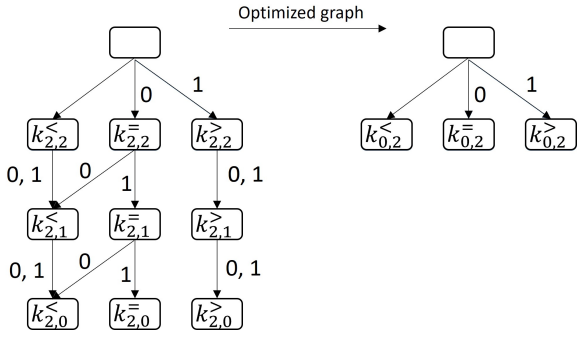

Fig. 3. Optimization for $\lambda = (011)$.

Algorithm 1 Aggregation result verification algorithm.

1: **if** $\forall j \quad e(\sigma'_{j,t}, g_2) == e(H_2(t,j), vk_1)e(g_1^{sum_{j,t}}, vk_2)$ **then**
2: Output that the aggregation result has not been altered
3: **else**
4: Output that the aggregation result has been altered
5: **end if**

Optimization. Note that the threshold values in Λ do not always need to be hidden from the users. In that case, there is an opportunity to optimize the protocol to increase its performance both in terms of communication and computation. For example, if the last l bits of λ_i are all '1' then there is no need to execute encryption and OT operations for the last l bits mainly because all potential inputs will inherently be smaller than λ_i. This optimization is visualized in Figure 3 for which $\lambda = (011)$. Only the first bit will need encryption and OT operations and all the remaining bits are automatically considered as being less than or equal to 1.

3.2 FULLSA1

We provide detailed steps of our first secure aggregation protocol in Protocol 5 together with FULLSA2 because the latter consists of an extension of FULLSA1 with operations and parameters that are highlighted in blue. Indeed, FULLSA1 addresses aggregator obliviousness in the semi-honest adversary model and user drop-out cases whereas FULLSA2 additionally allows the verification of the aggregation result (highlighted in blue).

Before the execution of the protocol, some operations need to be performed, beforehand. Protocol 4 describes this setup phase for FULLSA1 and FULLSA2 and mainly includes the generation of parameters and keys to be used in the protocol. FULLSA1 mainly consists of executing Protocol 3 with each online user in order to obliviously verify the range of their inputs and provide the relevant integrity material if nothing malicious is detected. Regarding dropped users, the aggregator is able to reconstruct their encrypted values together with thir tags thanks to the shares received by online users.

3.3 FULLSA2

FULLSA2 improves FULLSA1 by allowing anyone, who holds the verification key $VK = (vk_1, vk_2) = (g_2^{\sum tk'_i}, g_2^a)$, to verify the aggregation result by running Algorithm 1. The additional steps on top of FULLSA1 are marked with blue in Protocol 5.

In addition to the setup steps of FULLSA1, another tag key (tk'_i) should be created by each user and the verification key $VK = (vk_1, vk_2) = (g_2^{tk'_{i,t}}, g_2^a)$ should be generated for the setup for FULLSA2 as shown in step 6 of Protocol 4. Also, the users should know g_1^a, and g_2^a can be known by anyone who wants

Protocol 4 Setup protocol for FULLSA1. Additional steps for FULLSA2 are shown in blue.

1. The users and the aggregator agree on cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order p with a bilinear mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ where g_1 and g_2 are respectively generators of $\mathbb{G}_1$ and $\mathbb{G}_2$.
2. The users and the aggregator agree on a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_1$.
3. The users create encryption keys (ek_i) from $\mathbb{Z}_p$ randomly and the aggregator learns the decryption key $(ek_A = -\sum ek_i)$. A trusted party, who does not need to be online during the execution of the protocol, can be used for this operation or a secure computation protocol can be used.
4. The users create tag keys (tk_i) from $\mathbb{Z}_p$ randomly and send $g_2^{tk_i}$ to the aggregator.
5. Each $\mathcal{U}_i$ also creates secret shares for their encryption and tag keys such that by using a threshold secret sharing scheme such as [22] and sends the secret shares to other users so that at least t users can compute encryption of '0' and tag computation of '0' on behalf of the user.
6. Each user creates another tag key tk'_i and a verification key pair $VK = (vk_1, vk_2) = (g_2^{\sum tk'_i}, g_2^a)$ is generated and published. A trusted party, who does not need to be online during the execution of the protocol, can be used for this operation or a secure computation protocol can be used.

to verify the aggregation result. This information helps prevent the aggregator from sending an incorrect aggregate result.

3.4 FULLSA3

FULLSA3 improves FULLSA2 by making it secure against a malicious secure aggregator. To achieve this, the semi-honest secure batch verification protocol is replaced with a zero-knowledge protocol. As stated in [13], there seem to be only two ways to achieve security against malicious aggregators 1) needing more than one server where at least one of them should be semi-honest secure (non-colluding), or 2) utilization of zero-knowledge proofs. The former one is followed in some constructions such as ELSA[18], but since at least one of the servers should be semi-honest secure, so one could argue that at the end the protocol cannot be claimed to be secure against a malicious aggregator. Thus, it seems that zero-knowledge proofs become the only option to have a secure aggregation protocol in the full malicious adversary setting (i.e., all the participants can behave maliciously). Therefore, we utilize zero-knowledge proofs in FULLSA3 and integrate Bulletproofs as range proofs [3] as a realization of the protocol like in the Rofl solution [13].

We first give the zero knowledge functionality, which is needed in FULLSA3, in Functionality 3 and then present our malicious secure fault tolerant secure aggregation protocol (FULLSA3) with Protocol 6.

Protocol 5 Our secure aggregation protocols FULLSA1 (in black) and FULLSA2 (in black and blue).

Parameters. H_1, H_2 p, g_1, g_3, $\mathbb{G}_1$, g_2, $\mathbb{G}_2$, $\mathbb{G}_T$, e.
Inputs. Each non-dropped user $\mathcal{U}_i$ inputs $X_{i,t}$, ek_i, tk_i and tk'_i, $\{[ek_d]_i \mid 1 \leq d \leq n\}$, and g_1^a. $\mathcal{A}$ inputs ek_A, $\{g_2^{tk_i} \mid 1 \leq i \leq n\}$ and Λ.
Outputs. The $\mathcal{A}$ outputs an alert if $\exists x_{i,j,t}$. such that $x_{i,j,t} > \lambda_j$, otherwise outputs $\{sum_{j,t} \mid 1 \leq j \leq n_X \; sum_{j,t} = \sum_{i=1}^{n_A} x_{U_A[i],j,t}\}$.
Protocol steps:

1. $\mathcal{A}$ executes Protocol 3 with each non-dropped $\mathcal{U}_i$. In the protocol $\mathcal{A}$ and $\mathcal{U}_i$ respectively assume the role of P_1 and P_2. $\mathcal{A}$ randomly generates a weight set $W_t = \{w_{j,t} \mid 1 \leq j \leq n_X\}$ and inputs W and Λ to the protocol. $\mathcal{U}_i$ inputs $X_{i,t}$. At the end of the protocol $\mathcal{A}$ learns $k_{i,t}$ and U_i learns $o_{i,t}$ such that $o_{i,t} = k_{i,t} \times g_1^{\sum x_{i,j,t} w_{j,t}}$ if $\forall x_{i,j,t} \leq \lambda_j$. Otherwise, $o_{i,t}$ becomes a random value.
2. Each non-dropped $\mathcal{U}_i$ selects a set of random values $\{r_{i,j,t} \mid 1 \leq j \leq n_X\}$, computes the set of ciphertexts $\{y_{i,j,t} \mid 1 \leq j \leq n_X\}$ and the tag values $\sigma_{i,t}$ and $\sigma'_{i,j,t}$ as follows:
$$y_{i,j,t} = (1 + (x_{i,j,t} + r_{i,j,t})N) H_1(t,j)^{ek_i} \mod N^2$$
$$\sigma_{i,t} = H_2(t)^{tk_i} o_{i,t} g_1^{\sum r_{i,j,t}}$$
$$\sigma'_{i,j,t} = H_2(t,j)^{tk'_i} (g_1^a)^{x_{i,j,t}} g_3^{r_{i,j,t}}$$
and then sends $\{y_{i,j,t}\}$, $\sigma_{i,t}$ and $\{\sigma'_{i,j,t}\}$ to $\mathcal{A}$.
3. $\mathcal{A}$ broadcasts U_D.
4. Each non-dropped $\mathcal{U}_i$ sends its random number set $\{r_{i,j,t}\}$ and ciphertext sets $\{\{[y_{d,j,t}]_i \mid 1 \leq j \leq n_X, 1 \leq d \leq n_D\}$ and tag values $\{[\sigma_{d,t}]_i \mid 1 \leq d \leq n_D\}$ and $\{[\sigma'_{d,j,t}]_i \mid 1 \leq j \leq n_X, 1 \leq d \leq n_D\}$ for the dropped users such that
$$[y_{d,j,t}]_i = H_1(t,j)^{[ek_d]_i} \mod N^2$$
$$[\sigma_{d,t}]_i = H_2(t)^{[tk_d]_i}$$
$$[\sigma'_{d,j,t}]_i = H_2(t,j)^{[tk'_d]_i}$$
5. $\mathcal{A}$ performs the following steps:
   (a) constructs $y_{d,j,t}$, $\sigma_{d,t}$ and $\sigma'_{d,j,t}$ for the dropped users by using the *ShareCombine* function presented in [15].
   (b) for each dropped user, checks if the following equation holds $e(\sigma_{d,t}, g_2) = e(H_2(t), g_2^{tk_d})$
   (c) computes sums for each data point and the weighted sum as follows:
$$sum_{j,t} = (H_1(t,j)^{ek_A} \prod_{i=0}^{n} y_{i,j,t}) N^{-1} \mod N - \sum_{i=0}^{n_A} r_{U_A[i],j,t}$$
$$sum_t = \sum_{j=0}^{n_X} sum_{j,t} \times w_j$$
   (d) computes the aggregated tag value as follows:
$$\sigma_t = (\prod_{i=0}^{n_A} \sigma_{U_A[i],t} \times g_1^{-\sum_{j=0}^{n_X} r_{U_A[i],j,t}} \times k_{U_A[i],t}^{-1}) \times (\prod_{i=0}^{n_D} \sigma_{U_D[i],t})$$
   (e) checks the equation $e(\sigma_t, g_2) = e(H_2(t), g_2^{\sum tk_i}) e(g_1^{sum_t}, g_2)$
   (f) computes the following and publishes $\{sum_{j,t}, \sigma'_{j,t}\}$.
$$\sigma'_{j,t} = (\prod_{i=0}^{n_A} \sigma'_{U_A[i],j,t} \times g_3^{-r_{U_A[i],j,t}}) \times (\prod_{i=0}^{n_D} \sigma'_{U_D[i],j,t})$$
6. Anyone holding the verification key (vk_1, vk_2) can check if the aggregation results ($sum_{j,t}$ for $1 \leq j \leq n_X$) have been altered, by running Algorithm 1 where $vk_1 = g_2^{\sum tk'_i}$ and $vk_2 = g_2^a$.

Functionality 3 NIZK for input validation $\mathcal{F}^{NIZK}$

Parameters. H_2, generators g_1 and g_3 of the group $\mathbb{G}_1$, t.
Inputs. The prover inputs a set of items $X = \{x_j \mid 1 \leq j \leq n_X\}$, a set of random values $\{r_j \mid 1 \leq j \leq n_X\}$ and an encryption key ek', the verifier inputs a set of threshold values $\Lambda = \{\lambda_j \mid 1 \leq j \leq n_X\}$ and a set of ciphertexts $\{c_j \mid 1 \leq j \leq n_X\}$.
Outputs. The verifier outputs $SUCCESS$ if $c_j = H_2(t,j)^{ek'} g_1^{x_j} g_3^{r_j}$ and $x_j \leq \lambda_j$ for $1 \leq j \leq n_X$. Otherwise returns an error.

4 Security

Similar to [18], we present an ideal functionality for secure aggregation that is fault-tolerant and has input-range unforgeability (also called verified inputs [21] or input validation [1]). Unlike [18], we also present the aggregator as an entity other than the functionality so that one can consider a malicious aggregator as well. The ideal functionality is defined in Functionality 4.

Note that step 2 of $\mathcal{F}^{AGG}$ ensures *input-range unforgeability. Aggregator obliviousness* is satisfied since the aggregator only learns the sum from the functionality. *Aggregate unforgeability* would be satisfied as long as $\Upsilon = 1$.

Moreover, we can provide a threshold parameter $\tau > 1$ such that if the number of users that did not send $\perp$ (essentially $|U_A|$ in our case) is less than τ, then the functionality sets $sum_{j,t} = \perp$. In the protocols, this would mean that when the aggregator announces dropped users U_D, each user checks this threshold, and stops if $|U_A| < \tau$. To simplify the definition, protocols, and proofs, we present them without this detail. Security of the following theorem is proven via ideal-real indistinguishability.

Theorem 1. *Assuming that the underlying oblivious transfer protocol is secure against malicious receiver and the underlying encryption scheme is semantically-secure, then our Batch Oblivious Range Verification protocol achieves functionality $\mathcal{F}^{BORF}$ against malicious P_2.*

Proof (of Theorem 1). We present a simulator that interacts with malicious P_2 in the real world, and simulates it in the ideal world.

In step 1, the simulator acts as in the protocol. In step 2, acting as $\mathcal{F}^{OT}$ against malicious P_2, the simulator learns its real inputs x_j. The simulator sends those in the ideal world to $\mathcal{F}^{BORF}$. In return, the simulator receives $wsum = k \times g^{\Sigma x_j w_j}$ if $\forall j$ we have $x_j \leq \lambda_j$, or $wsum = \perp$ otherwise. Then in step 4, if $wsum \neq \perp$ the simulator performs according to the protocol. Otherwise ($wsum = \perp$), the simulator picks a random mask and continues. This is indistinguishable by P_2 since if the values are within the bounds, the simulator acted exactly as an honest P_1 in $\mathcal{F}^{OT}$ hybrid model, and if some value is out of bounds, then P_2 will reconstruct a random value at step 5, which is indistinguishable from the real random value it would obtain due to the semantic security of the underlying encryption scheme.

Finally, the simulator outputs whatever the adversary outputs. □

Protocol 6 Our secure aggregation protocol FULLSA3.

Parameters. $H1$, H_2, p, g_1, g_3, $\mathbb{G}_1$, g_2, $\mathbb{G}_2$, $\mathbb{G}_T$, e.
Inputs. Each non-dropped user $\mathcal{U}_i$ inputs $X_{i,t}$, ek_i, ek'_i, tk'_i, and $\{[ek_d]_i \mid 1 \leq d \leq n\}$. $\mathcal{A}$ inputs ek_A, ek'_A, $\{g_2^{tk_i} \mid 1 \leq i \leq n_X\}$, and Λ.
Outputs. $\mathcal{A}$ outputs an alert if $\exists x_{i,j,t}$ such that $x_{i,j,t} > \lambda_j$, otherwise outputs $\{sum_{j,t} \mid 1 \leq j \leq n_X\}$ where $sum_{j,t} = \sum_{i=1}^{n_A} x_{U_A[i],j,t}$.
Protocol steps:

1. Each non-dropped $\mathcal{U}_i$ selects a set of random values $\{r_{i,j,t}\}$ and computes and sends $\{y_{i,j,t}\}$, $\{\sigma'_{i,j,t}\}$ and $\{c_{i,j,t}\}$ for $1 \leq j \leq n_X$ such that

$$y_{i,j,t} = (1 + (x_{i,j,t} + r_{i,j,t})N)H_1(t,j)^{ek_i} \mod N^2$$

$$\sigma'_{i,j,t} = H_2(t,j)^{tk'_i}(g_1^a)^{x_{i,j,t}} g_3^{r_{i,j,t}}$$

$$c_{i,j,t} = H_2(t,j)^{ek'_i} g_1^{x_{i,j,t}} g_3^{r_{i,j,t}}$$

2. $\mathcal{A}$ acts as the verifier in a zero knowledge protocol realizing Functionality 3 with each non-dropped user $\mathcal{U}_i$ acting as the prover. This way, users prove that their $x_{i,j,t}$ values are within bounds.
3. $\mathcal{A}$ broadcasts U_D.
4. Each non-dropped $\mathcal{U}_i$ sends their random number set $\{r_{i,j,t}\}$ and ciphertext sets $\{\{[y_{d,j,t}]_i \mid 1 \leq j \leq n_X, 1 \leq d \leq n_D\}$ and $\{[c_{d,j,t}]_i \mid 1 \leq j \leq n_X, 1 \leq d \leq n_D\}$, and tag values $\{[\sigma'_{d,j,t}]_i \mid 1 \leq j \leq n_X, 1 \leq d \leq n_D\}$ for the dropped users such that

$$[y_{d,j,t}]_i = H_1(t,j)^{[ek_d]_i} \mod N^2$$

$$[c_{d,j,t}]_i = H_2(t,j)^{[ek'_d]_i} \mod N^2$$

$$[\sigma'_{d,j,t}]_i = H_2(t,j)^{[tk'_d]_i}$$

5. $\mathcal{A}$ performs the following steps:
   (a) constructs $y_{d,j,t}$, $c_{d,j,t}$ and $\sigma'_{d,j,t}$ for the dropped users by using the *ShareCombine* function presented in [15].
   (b) for each dropped user, and for $1 \leq j \leq n_X$, checks if the following equation holds $e(\sigma'_{d,j,t}, g_2) = e(H_2(t,j), g_2^{tk'_d})$
   (c) computes sums for each data point and the weighted sum as follows:

$$sum^y_{j,t} = (H_1(t,j)^{ek_A} \prod_{i=0}^{n} y_{i,j,t})N^{-1} \mod N$$

$$sum^c_{j,t} = (H_2(t,j)^{ek'_A} \prod_{i=0}^{n} c_{i,j,t})$$

   (d) checks if the equation holds $g_1^{sum^y_{j,t}} = sum^c_{j,t}$
   (e) computes $sum_{j,t} = sum^y_{j,t} - \sum_{i=0}^{n_A} r_{U_A[i],j,t}$ and $sum_t = \sum_{j=0}^{n_X} sum_j \times w_j$
   (f) computes the following and publishes $\{sum_{j,t}, \sigma'_{j,t}\}$.

$$\sigma'_{j,t} = (\prod_{i=0}^{n_A} \sigma'_{U_A[i],j,t} \times (g_1^a)^{r_{U_A[i],j,t}}) \times (\prod_{i=0}^{n_D} \sigma'_{U_D[i],j,t})$$

Functionality 4 Secure Aggregation Ideal Functionality $\mathcal{F}^{AGG}$

Parameters. Unforgeability parameter Υ.

1. Each user U_i sends its value set $X_{i,t}$ for the time step t, or $\perp$. Here, $\perp$ means that the user is not available for that time step (used for fault-tolerance). The aggregator sends the public bounds Λ for inputs.
2. The functionality then verifies each $x_{i,j,t} \in X_{i,t}$ where $X_{i,t} \neq \perp$ against bounds $\lambda_j \in \Lambda$ and calculates the aggregate output $sum_{j,t} = \Sigma_i x_{i,j,t}$ only if all verifications pass ($\forall i, j \quad x_{i,j,t} \leq \lambda_j$). Otherwise, sets $sum_{j,t} = \perp$. Then, the functionality sends $sum_{j,t}$ to the aggregator.
3. The aggregator has three options. It can respond with $ABORT$ or $CONTINUE$, or it can return a different $sum'_{j,t}$.
4. If the aggregator returns $ABORT$, then the functionality sends $ABORT$ to each user. If the aggregator returns $CONTINUE$, then the functionality sends $sum_{j,t}$ to each user. If the aggregator returns $sum'_{j,t}$ and the unforgeability parameter $\Upsilon = 0$, then the functionality sends $sum'_{j,t}$ to each user. If the aggregator returns $sum'_{j,t} \neq sum_{j,t}$ and $\Upsilon = 1$, then the functionality sends $INVALID$ to each user.

Theorem 2. *Assuming that the underlying oblivious transfer protocol is secure against semi-honest sender, then our Batch Oblivious Range Verification protocol achieves functionality $\mathcal{F}^{BORF}$ against semi-honest P_1.*

Proof (of Theorem 2). We present a simulator that simulates the view and output of P_1, only by knowing its input and output. The input of P_1 is essentially the bounds Λ and weights W and a random k. Its output is nothing.

The only interaction from P_2 to P_1 in the protocol is during the oblivious transfer. But, in $\mathcal{F}^{OT}$-hybrid world, P_1 receives nothing from $\mathcal{F}^{OT}$. Thus, the simulator acts exactly as in the protocol, outputs nothing as the view, and outputs nothing as the output. This is indistinguishable from real. □

Theorem 3. *Assuming that the underlying Batch Oblivious Range Verification protocol is secure, then FULLSA1 achieves functionality $\mathcal{F}^{AGG}$ with $\Upsilon = 0$ against malicious users.*

Proof (of Theorem 3). We present a simulator that interacts with malicious users in the real world, and simulates them in the ideal world.

In step 1, acting as $\mathcal{F}^{BORF}$ against malicious users, the simulator learns their real inputs.

During the remaining steps of the protocol, the simulator acts according to the protocol on behalf of honest users and the aggregator. For honest users, the simulator performs the simulation internally with random valid inputs (less than or equal to the bounds), but this is invisible to the adversary. If, at some point acting as the aggregator, some value is missing (not received from malicious users) or any of the verification checks in step 5 of Protocol 5 fails regarding malicious users, then the simulator sets that malicious user's input as $\perp$. If all

checks proceed, then the simulator sets that malicious user's input as the one learned as $\mathcal{F}^{BORF}$ in step 1.

The simulator then sends those in the ideal world to $\mathcal{F}^{AGG}$. In return, the simulator receives $sum_{j,t}$ values. The simulator publishes the received aggregate as the real aggregator.

The simulator is indistinguishable from real honest entities as far as the adversary is concerned, and it outputs whatever the adversary outputs. □

Theorem 4. *Assuming that the underlying Batch Oblivious Range Verification protocol and the Threshold Joye-Libert encryption scheme are secure, and the Decisional Diffie-Hellman assumption holds in $\mathbb{G}_1$, then FULLSA1 achieves functionality $\mathcal{F}^{AGG}$ with $\Upsilon = 0$ against a semi-honest aggregator in the Random Oracle model.*

Proof (of Theorem 4). We present a simulator that simulates the view and output of the aggregator, only by knowing its input and output. The aggregator's input is essentially the bounds Λ, and its output is the $sum_{j,t}$ values.

Knowing $sum_{j,t}$, the simulator splits them among the users to calculate $X_{i,t}$ sets for each user U_i so that all the bounds are satisfied: $\forall i, j$ s.t. $x_{i,j,t} \in X_{i,t}$ and $\lambda_j \in \Lambda$, we have $x_{i,j,t} \leq \lambda_j$. Then, acting as the users and the aggregator, the simulator performs the steps exactly as in the protocol. Finally, the simulator outputs the view of the aggregator, and the output $sum_{j,t}$ values it received.

The view output by the simulator is indistinguishable from the aggregator's view from its interaction with real users due to the following:

- In step 1 of Protocol 5, functionality $\mathcal{F}^{BORF}$ hides the users' inputs from the aggregator.
- In step 2 of the protocol, we provide a sketch of indistinguishability, due to space. Due to the security of the Threshold Joye-Libert encryption scheme (which relies on the hash function H_1 being a random oracle, and the Discrete Composite Residuosity assumption [16]), the adversary cannot distinguish $x_{i,j,t}$ values hidden within $y_{i,j,t}$ values. Moreover, due to $r_{i,j,t}$ values inside $\sigma_{i,t}$ values, $x_{i,j,t}$ values are information-theoretically hidden.
- In step 4 of the protocol, when the random numbers $r_{i,j,t}$ are given to the adversary, the adversary can get rid of the randomness in $y_{i,j,t}$ and $\sigma_{i,t}$. But, information about $x_{i,j,t}$ can only be recovered after aggregation due to $H_1(t,j)^{ek_i}$ in $y_{i,j,t}$ and $H_2(t)^{tk_i}$ in $\sigma_{i,t}$. Remember that to get rid of $H_1(t,j)^{ek_i}$ in $y_{i,j,t}$, values from multiple users need to be aggregated and canceled out with ek_A. A formal reduction would be very similar to the proof of *aggregator obliviousness* in PUDA [12], which relies on H_1 being a random oracle and the Decisional Diffie-Hellman assumption holding in $\mathbb{G}_1$.
- In step 4 of the protocol, regarding the dropped users, the simulation is exactly the same as an honest user interaction. The adversary already knows that 0 is the value used for dropped users. □

Note that, a semi-honest aggregator proof does not cover aggregate unforgeability, since the adversary performs the computation correctly. But, FULLSA1 does not provide a verification mechanism for the aggregate $sum_{j,t}$ values the

aggregator outputs. Therefore, if the adversary outputs some $sum'_{j,t} \neq sum_{j,t}$, then the simulator forwards it to $\mathcal{F}^{AGG}$ with $\Upsilon = 0$. Thus, FULLSA1 does not achieve aggregate unforgeability.

Theorem 5. *Assuming that the underlying Batch Oblivious Range Verification protocol is secure, then FULLSA2 achieves functionality $\mathcal{F}^{AGG}$ with $\Upsilon = 1$ against malicious users.*

Proof (of Theorem 5). Proof is the same as that of Theorem 3.

Theorem 6. *Assuming that the underlying Batch Oblivious Range Verification protocol and the Threshold Joye-Libert encryption scheme are secure, and the Decisional Diffie-Hellman assumption holds in $\mathbb{G}_1$, then FULLSA2 achieves functionality $\mathcal{F}^{AGG}$ with $\Upsilon = 1$ against a semi-honest aggregator in the Random Oracle model.*

Proof (of Theorem 6). The same argument in Theorem 4 apply, with $\Upsilon = 1$, with some difference: Regarding the indistinguishability of the view produced by the simulator against the view of the aggregator against honest users, FULLSA2 now additionally includes $\sigma'_{i,j,t}$ values. But, again assuming that g_1^a and g_3 generate the same group, $x_{i,j,t}$ values are again information-theoretically hidden at step 2. Similarly, at step 4, $H_2(t,j)^{tk'_i}$ in $\sigma'_{i,j,t}$ ensures that $x_{i,j,t}$ values cannot be recovered individually. A formal reduction would again be very similar to the proof of *aggregator obliviousness* in [12, 10], which relies on H_2 being a random oracle and the Decisional Diffie-Hellman assumption in $\mathbb{G}_1$.

Considering Algorithm 1, published $sum_{j,t}$ and $\sigma'_{j,t}$ values can be verified to ensure aggregate unforgeability (which is the reason that $\Upsilon = 1$). The verification for each j works as follows:

$$e(\sigma'_{j,t}, g_2) == e(H_2(t,j), vk_1)e(g_1^{sum_{j,t}}, vk_2) \quad (1)$$

$$e(\sigma'_{j,t}, g_2) == e(H_2(t,j), g_1^{\Sigma tk'_i})e(g_1^{sum_{j,t}}, g_2^a) \quad (2)$$

$$e(H_2(t,j)^{\Sigma tk'_i}(g_1^a)^{sum_{j,t}}, g_2) == e(H_2(t,j), g_1^{\Sigma tk'_i})e(g_1^{sum_{j,t}}, g_2^a) \quad (3)$$

This ensures aggregate unforgeability for FULLSA2 (and FULLSA3). □

Theorem 7. *Assuming that the underlying zero-knowledge protocol is extractable against malicious provers, then FULLSA3 achieves functionality $\mathcal{F}^{AGG}$ with $\Upsilon = 1$ against malicious users.*

Proof (of Theorem 7). The proof is very similar to the proof of Theorem 3. The only difference is that the simulator learns the adversarial inputs acting as $\mathcal{F}^{NIZK}$ instead of $\mathcal{F}^{BORF}$.

We present a simulator that interacts with malicious users in the real world, and simulates them in the ideal world.

In step 2, acting as $\mathcal{F}^{NIZK}$ against malicious users, the simulator learns their real inputs and sets accordingly (or $\perp$ if a bound check fails).

During the remaining steps of the protocol, the simulator acts according to the protocol on behalf of honest users and the aggregator. For honest users, the simulator performs the simulation internally with random valid inputs (less than or equal to the bounds), but this is invisible to the adversary. If, at some point acting as the aggregator, some value is missing (not received from malicious users) or any of the verification checks in step 5 of Protocol 6 fails regarding malicious users, then the simulator sets that malicious user's input as $\perp$. If all checks proceed, then the simulator sets that malicious user's input as the one learned as $\mathcal{F}^{NIZK}$ in step 2.

The simulator then sends those in the ideal world to $\mathcal{F}^{AGG}$. In return, the simulator receives $sum_{j,t}$ values. The simulator publishes the received aggregate as the real aggregator. □

Theorem 8. *Assuming that the underlying Threshold Joye-Libert encryption scheme is secure and the NIZK scheme is zero-knowledge, and the Decisional Diffie-Hellman assumption holds in $\mathbb{G}_1$„ then FULLSA3 achieves functionality $\mathcal{F}^{AGG}$ with $\Upsilon = 1$ against a malicious aggregator in the Random Oracle model.*

Proof (of Theorem 8). We present a simulator that interacts with the malicious aggregator in the real world and simulates it in the ideal world.

Note that, acting as the aggregator in the ideal world, the simulator learns the $sum_{j,t}$ values from $\mathcal{F}^{AGG}$.

Then, knowing $sum_{j,t}$, the simulator splits them among the users to calculate $X_{i,t}$ sets for each user U_i so that all the bounds are satisfied: $\forall i, j$ s.t. $x_{i,j,t} \in X_{i,t}$ and $\lambda_j \in \Lambda$, we have $x_{i,j,t} \leq \lambda_j$. Then, acting as the users, the simulator performs the steps exactly as in the protocol.

During this interaction, if the adversarial aggregator aborts at any stage, the simulator sends $ABORT$ to $\mathcal{F}^{AGG}$. Otherwise, the simulator sends $CONTINUE$ and stops by outputting whatever the adversary outputs.

Note that, it is possible for the adversarial aggregator to output some $sum'_{j,t} \neq sum_{j,t}$. In that case, the simulator forwards those to $\mathcal{F}^{AGG}$, which may send $INVALID$ to the users. In our protocol, this corresponds to the users detecting an inconsistency during the verification of $sum_{j,t}, \sigma'_{j,t}$ values, as explained before for FULLSA2.

The actions of the simulator is indistinguishable from the aggregator's interaction with real users due to the following:

- In step 1 of the protocol, we provide a sketch of indistinguishability, due to space. Due to the security of the Threshold Joye-Libert encryption scheme (which relies on the hash function H_1 being a random oracle, and the Discrete Composite Residuosity assumption [16]), the adversary cannot distinguish $x_{i,j,t}$ values hidden within $y_{i,j,t}$ values. Moreover, due to $r_{i,j,t}$ values inside $\sigma'_{i,j,t}$ values, $x_{i,j,t}$ values are information-theoretically hidden (as long as g_1^a and g_3 generate the same group).
- In step 2 of the protocol, $x_{i,j,t}$ values are hidden due to the zero-knowledge property of the underlying NIZK protocol.
- In step 4 of the protocol, when the random numbers $r_{i,j,t}$ are given to the adversary, the same argument as in the previous proofs apply, which relies

on H_2 being a random oracle and the Decisional Diffie-Hellman assumption holding in $\mathbb{G}_1$.
– In step 4 of the protocol, regarding the dropped users, the simulation is exactly the same as an honest user interaction. The adversary already knows that 0 is the value used for dropped users. □

5 Performance Analysis

In the performance analysis, we consider $\mathbb{G}_1$ implementation as an elliptic curve group. To estimate the cost of our protocols, we use experimental measurements of the various operations such as group operations, hash computation and exponentiation operation in mod N^2. These operations are executed on a regular personal computer (4 GB RAM, Intel i5-430M 2.27 GHz CPU, UBUNTU 18.04 operating system running on a USB stick). To evaluate the cost of group operations in our execution environment, we use an open-source library[4]. The group operations are in BN224 curve, the hash function is SHA-256, and the length of N is 2048 bits. The results observed by taking the average of 10 executions are presented in Table 3.

Table 3. Cost of operations. The hash operation is SHA-256, the group operations are in BN224 curve, and the length of N is 2048 bits.

Operation ID	Operation	Required time
O_1	Multiplication in $\mathbb{G}_1$	2.54 ms
O_2	Multiplication in $\mathbb{G}_2$	7.65 ms
O_3	Multiplication in $\mathbb{G}_T$	0.04 ms
O_4	Pairing	10.77 ms
O_5	Hash operation	0.003 ms
O_6	Multiplication in mod N^2	0.02 ms
O_7	Exponentiation in mod N^2	28 ms

Another primitive used in our secure aggregation protocols is the *ShareCombine* function. At the server side, *ShareCombine* needs to be executed for each dropped client. The overhead of ShareCombine, the additional operation executed in the user drop out case, is depicted in Table 4. The threshold in the table denotes the number of users that should contribute their shares so that the dropped users' zero-value can be integrated into the aggregate. The machine used has an Intel i7-7800X 3.50GHz processor and 126 GB of RAM. Interestingly, the number of dropped users n_D does not affect the time much which is good news for the aggregator. On the other hand, the dimensionality n_X of the data plays a linear role, as expected.

In the cost estimation of the protocols, we don't count the operations that can be done offline without knowing the execution specific inputs X, Λ, and W.

[4] https://github.com/IAIK/pairings_in_c

Table 4. Overhead of ShareCombine (O_8) at step 5 of our aggregation protocols.

n	threshold	n_X	n_D	Time (ms)
50	33	1,000	1	0.2531
50	33	1,000	10	0.2530
50	33	5,000	1	1.2554
50	33	5,000	10	1.2508
50	33	10,000	1	2.5160
50	33	10,000	10	2.5089
100	66	1,000	1	1.2141
100	66	1,000	10	1.2126
100	66	5,000	1	6.0138
100	66	5,000	10	6.0341
100	66	10,000	1	12.0465
100	66	10,000	10	12.0602
150	100	1,000	1	3.1347
150	100	1,000	10	3.1436
150	100	5,000	1	15.9199
150	100	5,000	10	15.7185
150	100	10,000	1	31.1136
150	100	10,000	10	31.1930

Table 5 depicts the estimations for the computational cost of the protocols for different set size values. The cost analysis of each protocol is explained in the following paragraphs.

Batch Oblivious Range Verification. Main operations in this protocol are public key operations for the base oblivious transfer that is used for OT-extension, symmetric key encryption operations, and multiplication operations in $\mathbb{G}_1$. Note that base oblivious transfer can be executed in advance (offline) before the execution of the protocol, so the communication and computation cost of the base OT can be ignored for the online phase. When the cost of multiplication in $\mathbb{G}_1$ and the cost of symmetric key encryption are compared, it is observed that group

Table 5. Estimated approximate total runtime at the client side (C) and at the server side (S) per client, based on the most costly operations, for our Batch Oblivious Range Verification Function (BORF), FULLSA1 and FULLSA2. In the FULLSA2 column, the cost for additional operations on top of FULLSA1 is given. Estimated costs for different values of n_X is computed under the values $n = 150$ and $n_D = 10$.

	n_X	BORF	FULLSA1	FULLSA2
C	n_X	-	$2n_XO_6$	$+n_XO_1$
S		n_XO_1	$(2n_DO_4 + 3O_4 + 3n_DO_8)/n + n_XO_6 + (n_X + 1)O_1$	$+n_XO_1$
C	1000	-	40 ms	+2540 ms
S		2540 ms	2565 ms	+2540 ms
C	5000	-	200 ms	+12700 ms
S		12700 ms	12807 ms	+12700 ms
C	10000	-	400 ms	+25400 ms
S		25400 ms	25610 ms	+25400 ms

multiplication becomes the dominant operation in terms of computation cost as seen in Table 3. The group multiplication operations in Protocol 3 are executed only in Step 1 by the aggregator. Thus the server needs to perform $\ell \times 2 \times n_X$ group multiplication operations. When the optimization for the oblivious graph explained in Section 3.1 is followed, then the cost in the server side becomes $2 \times n_X$. Also note that the number of group multiplication operations in Step 1 of the protocol can be reduced to n_X because the operations to generate random masks ($r_j^{ok_j}$) can be done offline, without knowing the actual data. With all these optimizations, the complexity at the server side becomes only n_X.

FULLSA1. FULLSA1 presented in Protocol 5 requires each client to perform $2 \times n_X$ multiplication operations in mod N^2 to compute $y_{i,j,t}$ for $1 \leq j \leq n_X$. Note that the operations $H_2(t)^{tk_i}$ and $g_1^{\sum r_{i,j,t}}$ can be done offline. In addition to the execution of *ShareCombine* function for the dropped users, the server also needs to perform two pairing operations for each dropped user. The server should also perform n_X multiplication operations in mod N for each client, one multiplication in $\mathbb{G}_1$ per client and three pairing operations only once. Also note that the cost of step 1 is n_X multiplication in $\mathbb{G}_1$.

FULLSA2. FULLSA2 presented in Protocol 5 is constructed by including some additional steps, shown in blue in the protocol, on top of FULLSA1. These additional steps introduce a cost of n_X multiplication operations in $\mathbb{G}_1$ for computation of $(g_1^a)^{x_{i,j,t}}$ to each client. The server additionally needs to perform n_X multiplication in $\mathbb{G}_1$ per client to compute $g_3^{-r_{U_A[i],j,t}}$.

Remark. It is worth noting that the impact of dropouts in the accuracy of the model is out of the scope of this paper as it was the case for all papers tackling the same problem.

Discussion. We don't provide expected costs for FULLSA3, but it is obvious that the overhead of FULLSA3 becomes considerably high compared to FULLSA2 because it involves zero-knowledge proofs to make the protocol secure against a malicious aggregator. Also Step 5.b of FULLSA3 (Protocol 6) requires $2n_X$ pairing operations for each dropped user. Instead of this amount of pairing operations, zero knowledge proofs can be used for ensuring of encryption of '0' for the dropped users.

As seen from Table 5, although there is no considerable cost on top of batch oblivious range verification for execution of FULLSA1, to make the protocol secure against the potential malicious behavior of changing the aggregation result by the aggregator, the cost doubles in FULLSA2. Since each solution in the literature has different properties and different setup for experiments, we couldn't find a fair way to compare the performance of our protocols with the existing solutions.

Acknowledgments

This work was partially supported by the Scientific and Technological Research Council of Turkey (TUBITAK) through 119E088, and the 1515 Frontier Research and Development Laboratories Support Program under Project 5169902, and has been partly funded by the Hexa-X II project which has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation program and Grant Agreement No 101095759. The work was also supported by the 3IA Côte d'Azur programme reference number ANR-19-P3IA-0002.

References

1. Bell, J., Gascón, A., Lepoint, T., Li, B., Meiklejohn, S., Raykova, M., Yun, C.: ACORN: input validation for secure aggregation. In: USENIX Security (2023)
2. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: ACM CCS (2017)
3. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: IEEE S&P (2018)
4. Ciampi, M., Orlandi, C.: Combining private set-intersection with secure two-party computation. In: SCN (2018)
5. Corrigan-Gibbs, H., Boneh, D.: Prio: Private, robust, and scalable computation of aggregate statistics. In: USENIX NSDI (2017)
6. Domingo-Ferrer, J., Blanco-Justicia, A., Manjón, J.A., Sánchez, D.: Secure and privacy-preserving federated learning via co-utility. IEEE Internet Things J. **9**(5), 3988–4000 (2022)
7. Joye, M., Libert, B.: A scalable scheme for privacy-preserving aggregation of time-series data. In: FC (2013)
8. Karakoç, F., Karaçay, L., Cnudde, P.Ç.D., Gülen, U., Fuladi, R., Soykan, E.U.: A security-friendly privacy-preserving solution for federated learning. Comput. Commun. **207**, 27–35 (2023)
9. Karakoç, F., Nateghizad, M., Erkin, Z.: SET-OT: A secure equality testing protocol based on oblivious transfer. In: ARES (2019)
10. Karakoç, F., Önen, M., Bilgin, Z.: Secure aggregation against malicious users. In: ACM SACMAT (2021)
11. Leontiadis, I., Elkhiyaoui, K., Molva, R.: Private and dynamic time-series data aggregation with trust relaxation. In: CANS (2014)
12. Leontiadis, I., Elkhiyaoui, K., Önen, M., Molva, R.: PUDA - privacy and unforgeability for data aggregation. In: CANS (2015)
13. Lycklama, H., Burkhalter, L., Viand, A., Küchler, N., Hithnawi, A.: Rofl: Robustness of secure federated learning. In: IEEE S&P (2023)
14. Ma, Y., Woods, J., Angel, S., Polychroniadou, A., Rabin, T.: Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In: IEEE S&P (2023)
15. Mansouri, M., Önen, M., Jaballah, W.B.: Learning from failures: Secure and fault-tolerant aggregation for federated learning. In: ACSAC (2022)
16. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT (1999)

17. Rabin, M.O.: How to exchange secrets by oblivious transfer. Tech. rep., Harvard Aiken Computation Laboratory Technical Report TR-81 (1981)
18. Rathee, M., Shen, C., Wagh, S., Popa, R.A.: ELSA: secure aggregation for federated learning with malicious actors. In: IEEE S&P (2023)
19. Rieke, N., Hancox, J., Li, W., et al.: The future of digital health with federated learning. NPJ Digital Medicine **3** (2020)
20. Roy, L.: Softspokenot: Communication–computation tradeoffs in ot extension. Cryptology ePrint Archive, Paper 2022/192 (2022)
21. Saeed, A., Zhao, Y., Dukkipati, N., Zegura, E.W., Ammar, M.H., Harras, K., Vahdat, A.: Eiffel: Efficient and flexible software packet scheduling. In: USENIX NSDI (2019)
22. Shamir, A.: How to share a secret. ACM Communications (1979)
23. Shi, E., Chan, T.H., Rieffel, E.G., Chow, R., Song, D.: Privacy-preserving aggregation of time-series data. In: NDSS (2011)
24. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: IEEE S&P (2017)
25. Sirohi, D., Kumar, N., Singh Rana, P., Tanwar, R., Iqbal, R., Hijjii, M.: Federated learning for 6G-enabled secure communication systems: a comprehensive survey. Artif Intell Rev **56** (2023)