Secure Aggregation Against Malicious Users

Ferhat Karakoç
Ericsson Research
Istanbul, Turkey
ferhat.karakoc@ericsson.com

Melek Önen
EURECOM
Sophia-Antipolis, France
melek.onen@eurecom.fr

Zeki Bilgin
Arçelik Research
Istanbul, Turkey
zeki.bilgin@arcelik.com

ABSTRACT

Secure aggregation protocols allow an *aggregator* to compute the sum of multiple users' data in a privacy-preserving manner. Existing protocols assume that users from whom the data is collected, are fully trusted on the correctness of their individual inputs. We believe that this assumption is too strong, for example when such protocols are used for federated learning whereby the aggregator receives all users' contributions and aggregate them to train and obtain the joint model. A malicious user contributing with incorrect inputs can generate model poisoning or backdoor injection attacks without being detected. In this paper, we propose the first secure aggregation protocol that considers users as potentially malicious. This new protocol enables the correct computation of the aggregate result, in a privacy preserving manner, only if individual inputs belong to a legitimate interval. To this aim, the solution uses a newly designed oblivious programmable pseudo-random function. We validate our solution as a proof of concept under a federated learning scenario whereby potential backdoor injection attacks exist.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**.

KEYWORDS

secure aggregation; oblivious pseudo-random function; malicious users

ACM Reference Format:
Ferhat Karakoç, Melek Önen, and Zeki Bilgin. 2021. Secure Aggregation Against Malicious Users. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies (SACMAT) (SACMAT '21), June 16–18, 2021, Virtual Event, Spain.*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

1 INTRODUCTION

With the advent of the IoT and cloud technologies, data becomes a critical corporate asset. More and more companies are nowadays collecting huge amounts of data from a variety of sources and use data analytics tools to acquire meaningful insights and make value out of them. Among these tools, secure aggregation protocols have been intensively studied in the last two decades. The basic setting of such protocols consists of multiple parties coordinating with an *aggregator* whose goal is to compute the sum of parties' inputs without leaking any information on individual parties' private inputs beyond the aggregated value itself.

There exist many secure aggregation solutions in the literature (for example, see [12, 17]). These solutions mostly focus on the problem of data privacy, i.e., on keeping parties' individual inputs confidential while enabling the aggregator to compute and reveal the sum of the inputs. On the other hand, all parties involved in the aggregation protocol, are assumed to be fully trusted on the correctness and integrity of the inputs and computation. While few of the solutions such as [12] consider the aggregator as a potentially malicious adversary, in this paper, we consider the existence of malicious parties who can send bogus inputs instead of their legitimate inputs and consequently render the computation useless. We study this stronger threat model whereby collaborating parties are considered as potentially malicious and build the first secure aggregation protocol under this model. Similar to [12], our solution allows the aggregator to compute the sum of the collected inputs in a privacy preserving manner while at the same time enabling the aggregation of some individual tags that help verify the correctness of the computation. On the other hand, additionally, this verification tag is correctly computed only if the collected individual inputs belong to a specific interval and more specifically whenever the individual values are lower than a specific threshold.

Hence, our new solution extends the secure aggregation scheme in [12] by introducing a preliminary phase during which users are only able to compute valid verification tags when their individual value is below a certain threshold value. Otherwise, the users hold random tags and therefore the verification will fail. This preliminary phase involves a newly designed oblivious programmable pseudo-random function (OPPRF) that is executed between the aggregator and each user in order to avoid the aggregator to discover the private inputs and to output specific integrity values when the user's input is legitimate. The design of this new OPPRF scheme is based on the private set membership (PSM) protocol introduced in [4]. This particular PSM protocol is transformed into a secure comparison protocol similar to [8].

As a proof of concept, we further apply our new secure aggregation protocol to a particular federated learning scenario exposed to model backdoor injection attacks. In [14], authors developed a construction of a machine learning model whereby multiple parties collaborate to the training of the model with their private local model parameters without revealing them to other parties including the aggregator. More specifically, a trusted aggregator initializes a model and sends its parameters to the parties; Each party retrains the model using its local dataset and sends the updates of the model parameters to the aggregator who, in turn, merges these individual updates to obtain the global model. In [1], authors show that these

SACMAT '21, June 16–18, 2021, Virtual Event, Spain.

ACM ISBN 978-1-4503-8365-3/21/06…$15.00
https://doi.org/10.1145/XXXXXX.XXXXXX

solutions suffer from malicious parties intervening the training phase in the pursuit of their aims. We evaluate the performance of our solution in this particular context and validate its effectiveness against backdoor attacks.

Our contributions. Our contributions can be summarized as follows:

- We propose the design of a secure aggregation protocol that considers both the aggregator and the collaborating parties as potential adversaries;
- As a building block, we develop a new OPPRF construction which enables the aggregator to blindly check that the value is below a given threshold. This construction is based on Ciampi and Orlandi's PSM protocol [4] and can also be independently used;
- We introduce a privacy enhanced federated learning (FL) that is secure against malicious users by applying our secure aggregation protocol on FL in the aggregation step;
- We implement our secure aggregation to evaluate its performance;
- We validate the solution towards an FL scenario exposed to backdoor attacks, by identifying that the most sensitive parameters (to backdoor attacks) belong to the last layer bias values of neural network and applying our secure aggregation to these actual parameters.

Related work. There have been several studies on privacy preserving aggregation such as [7, 11, 17] whereby all players are considered as honest-but-curious. In [12], authors study a stronger threat model in which the adversary is the aggregator and may tamper with the aggregated result. Recently, some solutions [3, 9] instantiate secure aggregation for federated learning whereby data collected from collaborating users actually are machine learning models' parameters. These solutions, consider users from which data are collected in a privacy preserving manner, as potential adversaries: Youssef et al [9], study potential poisoning attacks launched by users against a collaborative learning scheme. The solution employs aggregation over secret shared inputs. As opposed to our solution, the security of their scheme relies on the existence of two non colluding servers. Furthermore, in [3], authors combine secret sharing with random masking and digital signatures. Although the study considers active adversaries they do not guarantee the correctness of the result. Furthermore, their solution requires the collaboration of all users among each other whereas in our solution each user only communicates with the aggregator. In [18], authors propose a collaborative linear machine learning framework named *Helen* whereby collaborating parties are assumed malicious. Authors make use of zero knowledge proofs when parties perform local computations. As stated in the paper, Helen does not protect against "bad data". Hence our solution can be considered as complementary to Helen. Finally, a recent solution, FLGuard, was proposed to protect the federated learning process against multiple backdoor attacks in [15]. This solution involves a secure two-party protocol to satisfy the privacy requirements while preventing the backdoor attacks. Although this solution supports the protection against multiple backdoors, its runtime is significant compared to our proposal's one.

Outline. In Section 2, we give brief information about the underlying primitives of our protocols. We present our OPPRF and secure aggregation protocols in Section 3. We analyze the setting of federated learning, which we apply our secure aggregation protocol on, in Section 4. Sections 5 and 6 respectively include security and performance analyses.

2 PRELIMINARIES

2.1 PUDA Protocol

The PUDA protocol [12] is a secure aggregation protocol which, in addition to the privacy of users' data, also ensures the correctness of the aggregation operation and the integrity of the aggregated result against malicious aggregators. The protocol involves a trusted key dealer $\mathcal{KD}$, a data analyzer $\mathcal{DA}$, an aggregator $\mathcal{A}$ and the users $\mathcal{U}_i$ whose data are collected and aggregated. The main role of $\mathcal{KD}$ is to generate the global parameters and the required keying material during a preliminary setup phase. $\mathcal{KD}$ further becomes offline and does not take any role in the actual protocol. $\mathcal{A}$ aggregates the users' data and $\mathcal{DA}$ receives the result and verifies its correctness.

More specifically, during the setup phase, $\mathcal{KD}$ takes the security parameter κ as input and generates the following parameters:

- cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order p where g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.
- a bilinear map e where $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for $a, b \in \mathbb{Z}_p$; It is easy to compute $e(g_1^a, g_2^b)$; $e(g_1, g_2) \neq 1$.
- a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_1$.
- an encryption key ek_i from $\mathbb{Z}_p$ for each user $\mathcal{U}_i$. ek_i is sent to $\mathcal{U}_i$, securely ($1 \leq i \leq n$ where n is the number of users).
- $sk_A = -\sum_i^n ek_i$. sk_A is further sent to $\mathcal{A}$.
- g_1^a where a is randomly generated. g_1^a is sent to each user.
- the verification key $VK = (vk_1, vk_2) = (g_2^{\sum tk_i}, g_2^a)$ whereby $g_2^{tk_i}$ is received from each user $\mathcal{U}_i$. This verification key is further sent to $\mathcal{DA}$.

After the setup phase, during the actual execution of the protocol, users encrypt their private input and compute an integrity tag. The aggregator computes the aggregated value using the received encrypted inputs and decrypts the result using the decryption key. $\mathcal{A}$ also computes the aggregated integrity tag. The sum and the integrity tag are sent to $\mathcal{DA}$ for validation of the result. The steps of the protocol are specified in Protocol 1.

2.2 Oblivious Transfer

An oblivious transfer (OT) [16] is a secure two-party protocol between a sender and a receiver where the sender inputs two messages (m_0, m_1) and the receiver inputs a choice bit (b); The receiver outputs m_b while the sender outputs nothing.

2.3 Ciampi-Orlandi Private Set Membership Protocol

Private set membership (PSM) is a multi-party protocol where parties holding private sets can learn the intersection of these sets and nothing more. Ciampi and Orlandi propose a two-party PSM protocol in [4] which outputs an encrypted result of the intersection instead of the intersection itself. The solution mainly consists of

Protocol 1: PUDA Protocol

Public Parameters. generated by $\mathcal{KD}$ during the setup phase: $H, p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e$.

Inputs. $\mathcal{U}_i$ holds a value $x_{i,t}$ for time interval t, a tag key tk_i, an encryption key ek_i and g_1^a. $\mathcal{A}$ has sk_A. $\mathcal{DA}$ has the verification key VK.

Outputs. $\mathcal{A}$ outputs $sum_t = \sum_{i=1}^{n} x_{i,t}$ and $\mathcal{DA}$ outputs the result of the verification.

Protocol steps for each time interval t:

(1) Each $\mathcal{U}_i$ computes ciphertext $c_{i,t}$ and tag value $\sigma_{i,t}$ for its input $x_{i,t}$ as follows

$$c_{i,t} = H(t)^{ek_i} g_1^{x_{i,t}}, \sigma_{i,t} = H(t)^{tk_i} (g_1^a)^{x_{i,t}}$$

(2) $\mathcal{A}$ computes $V_t = (\prod_i c_{i,t}) H(t)^{sk_A} = g_1^{sum_t}$, extracts the sum value from V_t by brute-force, and finally generates the aggregated tag as $\sigma_t = \prod_i \sigma_{i,t} = H(t)^{\sum_i tk_{i,t}} (g_1^a)^{sum_t}$. Both sum_t and σ_t are sent to $\mathcal{DA}$.

(3) $\mathcal{DA}$ verifies the aggregation as follows: $e(\sigma_t, g_2) == e(H(t), vk_1) e(g_1^{sum_t}, vk_2)$.

party P_1 constructing a particular graph for its set and of party P_2 tracing the graphs, obliviously, for the items in its set. It is worth to note that this protocol can be instantiated for secure equality check when the cardinality of each party's set is one (see [8]). Similarly, in our protocol construction we convert Ciampi-Orlandi's PSM protocol into a variant of secure comparison protocol and use this comparison protocol to allow the users to get a valid encrypted tag value if their inputs belongs to a legitimate interval.

3 OUR SECURE AGGREGATION PROTOCOL

We propose to design a new secure aggregation protocol which considers users from whom input is collected in a privacy preserving manner, as potentially malicious. With this aim, we propose to extend the PUDA protocol [12] by introducing a preliminary phase where the individual tag is correctly computed only if the input belongs to a legitimate interval. This phase involves the use of a newly designed oblivious programmable pseudo-random function (OPPRF) that is based on the oblivious graph tracing idea proposed in [4]. In this section, we first define an oblivious programmable pseudo-random function and describe a new construction for OPPRF. We further provide the specification of the resulting aggregation scheme.

3.1 Our OPPRF Construction

An oblivious pseudo-random function (OPRF) [5] is a two-party protocol where party P_1 and party P_2 respectively inputs a key K and a string x and P_2 outputs $F_K(x)$ where F is a pseudo-random function family that receives a key K and a string x and outputs a random-looking result (P_1 outputs nothing).

An oblivious *programmable* pseudo-random function (OPPRF) [10] is defined as an extension of OPRF whereby the protocol outputs predefined, specific values for some of the programmed inputs (and random-looking outputs for other values).

For our secure aggregation protocol, we construct a variation of OPPRF such that P_2 corresponding to the user, can learn a specific value only if x is smaller than a predefined threshold value. Otherwise, the user learns a random-looking value (i.e., $F_K(x)$). For this purpose, we introduce our OPPRF protocol (described in Functionality 2) to be run between a user (P_2) holding private input x and an aggregator (P_1) holding secret key g^a and threshold value λ. At the end of the protocol, P_2 learns $k_{opprf}(g^a)^x$, only if x is smaller than or equal to λ and the P_1 learns k_{opprf}. This value helps the user contribute to a valid construction of the validation tag. If, on the other hand, $x > \lambda$, then the user learns a random-looking value and thus the tag verification will later fail. It is worth to note that the aggregator learns nothing about the private input x.

Functionality 2: Oblivious Programmable Pseudo Random Function

Parameters. A generator g of a group $\mathbb{G}$.

Inputs. P_2 inputs x, P_1 inputs a threshold value λ and secret keys g^a.

Outputs. If $x \leq \lambda$ then P_2 learns $k_{opprf}(g^a)^x$, otherwise the user outputs a random value. P_1 outputs k_{opprf}.

For the sake of clarity, the specification of a simple version of our OPPRF protocol is provided in Protocol 3. This protocol can only be executed with positive integer inputs. An improved version also supporting negative integers is described in Appendix A.

More specifically, the user and the aggregator run l OT protocols where l is the bit length of x. In each OT execution, the user learns a masked piece of the information about the result $(g^a)^x$ and a key which is necessary to remove the masks on the result if $x \leq \lambda$. More precisely, in the i-th OT the user learns $(g^a)^{2^{i-1}x_{i-1}} r^{ok_{i-1}}$ where $r^{ok_{i-1}}$ is the mask and x_{i-1} is the $(i-1)$-th right most bit of $x = x_{\ell-1}\|...\|x_1\|x_0$. At the end of ℓ OTs, the aggregator sends the encryption result of $k_{opprf} r^{-\sum_{i=0}^{|x|-1} ok_i}$ and a random value under the different keys. If $x \leq \lambda$ then the user learns the key used for the encryption of $k_{opprf} r^{-\sum_{i=0}^{\ell-1} ok_i}$. Note that the user can extract $k_{opprf}(g^a)^x$ as follows:

$$k_{opprf} r^{-\sum_{i=0}^{\ell-1} ok_i} \times \Pi_{i=1}^{\ell}((g^a)^{2^{i-1}x_{i-1}} \times r^{ok_{i-1}}) = k_{opprf}(g^a)^x$$

Otherwise, the output becomes a random value.

To enable the user to remove the mask from the actual result only when $x \leq \lambda$, we utilize Ciampi-Orlandi's PSM protocol, which is based on oblivious graph tracing. The aggregator assigns three encryption keys for each bit of x and builds a chain of encryption keys to build a secure comparison protocol. Figure 1 shows an example graph for $\lambda = (100)$. Since the leftmost bit of λ is '1', the user learns $k_2^<$ when the leftmost bit of x is '0', otherwise learns $k_2^=$ after the execution of the first OT. Let us assume that $x = (001)$. After the execution of the first OT protocol, the user learns $k_2^<$. After the second execution of OT, since the second bit of x is '0', the user receives $E_{k_2^<}(k_1^<), E_{k_2^=}(k_1^=)$ and $E_{k_2^>}(k_1^>)$. Because the user holds key $k_2^<$ received from the previous OT protocol, can only decrypt $E_{k_2^<}(k_1^<)$[1]. After the decryption operation the user learns $k_1^<$. Since the user only knows $k_1^<$, it can only recover $k_0^<$ that is finally

[1]It is worth to note that the encryption function has the property that the user can validate whether the decryption operation is successful or not.

Protocol 3: (Our OPPRF Protocol)

Inputs. P_2 inputs x of length ℓ, P_1 inputs a threshold λ and secret keys g^a.

Outputs. If $x \leq \lambda$ then P_2 outputs $k_{opprf}(g^a)^x$, otherwise outputs a random value. P_1 outputs k_{opprf}.

Protocol steps:

(1) P_1 randomly chooses $k_{opprf} \in \mathbb{G}$ and prepares (S_i^0, S_i^1) for $1 \leq i \leq \ell$, which are the input message pairs for each OT, as follows:

(a) Chooses four symmetric keys $k_{\ell-1}^{<}$, $k_{\ell-1}^{=}$, $k_{\ell-1}^{>}$, and $ok_{\ell-1}$ randomly and a random value r and then computes S_ℓ^0 and S_ℓ^1 as follows:

$$S_\ell^0 = \begin{cases} \{k_{\ell-1}^{=}, (g^a)^{0\times 2^{\ell-1}} r^{ok_{\ell-1}}\} & \text{if } \lambda_{\ell-1} = 0 \\ \{k_{\ell-1}^{<}, (g^a)^{0\times 2^{\ell-1}} r^{ok_{\ell-1}}\} & \text{otherwise} \end{cases}$$

$$S_\ell^1 = \begin{cases} \{k_{\ell-1}^{>}, (g^a)^{1\times 2^{\ell-1}} r^{ok_{\ell-1}}\} & \text{if } \lambda_{\ell-1} = 0 \\ \{k_{\ell-1}^{=}, (g^a)^{1\times 2^{\ell-1}} r^{ok_{\ell-1}}\} & \text{otherwise} \end{cases}$$

(b) For $i = \ell - 1$ to 1

- Chooses four symmetric keys $k_{i-1}^{<}$, $k_{i-1}^{=}$, $k_{i-1}^{>}$, and ok_{i-1} randomly and computes S_i^0 and S_i^1 as follows:

If $\lambda_{i-1} = 0$,

$$S_i^0 = \{E_{k_i^{<}}(k_{i-1}^{<}), E_{k_i^{=}}(k_{i-1}^{=}), E_{k_i^{>}}(k_{i-1}^{>}), (g^a)^{0\times 2^{i-1}} r^{ok_{i-1}}\}$$

$$S_i^1 = \{E_{k_i^{<}}(k_{i-1}^{<}), E_{k_i^{=}}(k_{i-1}^{>}), E_{k_i^{>}}(k_{i-1}^{>}), (g^a)^{1\times 2^{i-1}} r^{ok_{i-1}}\}$$

Otherwise,

$$S_i^0 = \{E_{k_i^{<}}(k_{i-1}^{<}), E_{k_i^{=}}(k_{i-1}^{<}), E_{k_i^{>}}(k_{i-1}^{>}), (g^a)^{0\times 2^{i-1}} r^{ok_{i-1}}\}$$

$$S_i^1 = \{E_{k_i^{<}}(k_{i-1}^{<}), E_{k_i^{=}}(k_{i-1}^{=}), E_{k_i^{>}}(k_{i-1}^{>}), (g^a)^{1\times 2^{i-1}} r^{ok_{i-1}}\}$$

(c) Permutes the ciphertexts in S_i^0 and S_i^1 for $1 \leq i \leq \ell$, randomly.

(2) P_1 and P_2 run ℓ oblivious transfer protocols where in the i-th OT P_1 inputs the message pair (S_i^0, S_i^1) and P_2 inputs x_{i-1} as the choice bit and P_2 learns $S_i^{x_{i-1}}$.

(3) P_1 chooses a random number r' and a key k_{opprf} to mask the result, computes $E_{k_0^{<}}(k_{opprf} r^{-\sum_{i=0}^{\ell-1} ok_i})$, $E_{k_0^{=}}(k_{opprf} r^{-\sum_{i=0}^{\ell-1} ok_i})$, and $E_{k_0^{>}}(r')$ and sends the encryption results in random order to P_2.

(4) P_2 sets $result = (g^a)^{2^{\ell-1} x_{\ell-1}} r^{ok_{\ell-1}}$ from the received message $S_\ell^{x_{\ell-1}}$ in the ℓ-th OT. P_2 also sets $k'_{\ell-1}$ as the key in the received message.

(5) For $i = \ell - 1$ to 1

(a) P_2 sets $result = result \times (g^a)^{2^{i-1} x_{i-1}} r^{ok_{i-1}}$ from the received message $S_i^{x_{i-1}}$ in the i-th OT. P_2 will also be able to decrypt only one of the ciphertexts in the received message using k'_i and sets k'_{i-1} as the decryption result.

(6) P_2 will be able to decrypt only one of the ciphertexts received in Step 3 using k'_0 and multiplies the $result$ with the decryption result.

(7) P_2 and P_1 respectively output $result$ and k_{opprf}.

used to encrypt the mask needed to hide output $k_{opprf}(g^a)^x$. The

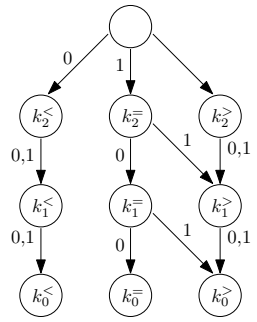


Figure 1: Graph representation of key encryptions executed by the aggregator for $\lambda = (100)$. Note that the path to $k_2^>$ is never used in this example. If the left-most bit of λ was '0', then the path to $k_2^<$ would not be used.

underlying OT protocol for each step of our OPPRF is specified in Protocol 3. In step 1, the aggregator prepares message pairs for OTs (one message for each potential bit value). These messages include the encryption of keys used to encrypt/decrypt the messages for the next step. Hence the chain of OT protocols follow the actual graph of encryption keys illustrated in Figure 1: The keys in the child nodes are encrypted with the keys in the parent nodes. In Step 5 of Protocol 3, the user obliviously traces this graph, which allows the user to learn only one of the keys in the leaves. The execution of our OPPRF protocol for the example where $\lambda = 100$ and $x = 001$ is depicted in Figure 2. As seen from the figure, since the private input of the user is less than the threshold value, the user is able to compute $(g^a)^x$.

3.2 Our Secure Aggregation Protocol

In this section, we introduce our secure aggregation protocol which is secure against honest-but-curious aggregator ($\mathcal{A}$). The aggregator and users respectively input a threshold (λ) and private input (x_i for $\mathcal{U}_i$), and the aggregator outputs the sum of the users' inputs if each user's input is smaller than λ, otherwise the aggregator outputs an error indicating that at least one of the user's input is larger than λ. For the design of our solution we use the idea in the PUDA protocol [12]. In PUDA, the aggregator can be malicious and the users are assured that the sum of their inputs is not altered. In our case, in addition to the correctness of the actual sum, the aggregator ensures that the private input of each participating user is smaller than the threshold λ.

Our protocol steps are given in Protocol 4. Similar to PUDA, a preliminary setup phase involves a trusted key dealer $\mathcal{KD}$ which generates the keys used in the protocol. KD disappears afterwards.

Setup. $\mathcal{KD}$ takes the security parameter κ as input and generates the following parameters as follows:

- Creates cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order p with a bilinear mapping e where g_1 and g_2 are respectively generators of $\mathbb{G}_1$ and $\mathbb{G}_2$.
- Selects a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_1$.
- Selects encryption keys ek_i from $\mathbb{Z}_p$ randomly and sends ek_i to $\mathcal{U}_i$ in a secure way for $1 \leq i \leq n$ where n is the number of users.

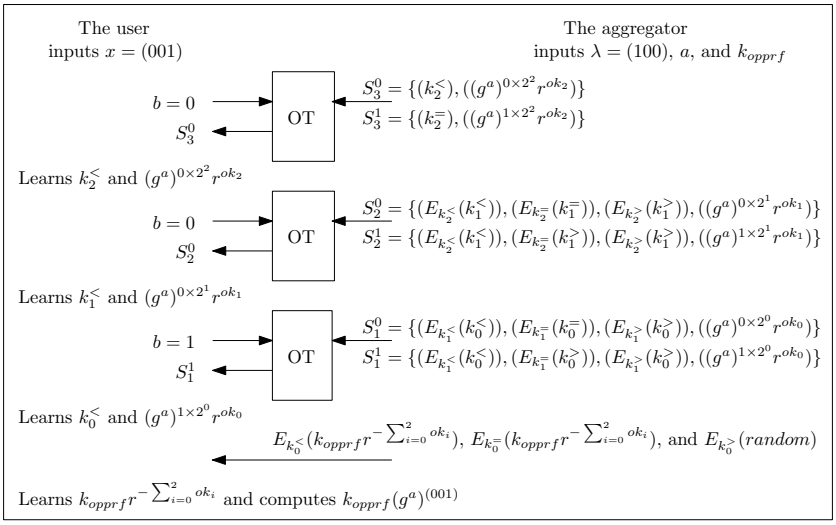


Figure 2: Execution of Protocol 3 for $x = (001)$ and $\lambda = (100)$.

- Computes $sk_A = -\sum_i^n ek_i$ and sends sk_A to $\mathcal{A}$.
- Receives $g_2^{tk_i}$ from $\mathcal{U}_i$ where tk_i is chosen randomly by the user.
- Computes a part of the verification key $vk_1 = g_2^{\sum tk_i}$ and sends vk_1 to $\mathcal{A}$.

Protocol 4: (Our Secure Aggregation Protocol)

Parameters. $H, p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e$

Inputs. $\mathcal{U}_i$ inputs value $x_{i,t}$ for time interval t, tag key tk_i, encryption key ek_i. $\mathcal{A}$ inputs sk_A and vk_1.

Outputs. The $\mathcal{A}$ outputs an alert 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 the ciphertext as $c_{i,t} = H(t)^{ek_i} g_1^{x_{i,t}}$ and sends it to $\mathcal{A}$.
(2) $\mathcal{A}$ chooses a random number a_t, runs Protocol 3 with each $\mathcal{U}_i$. $\mathcal{U}_i$ learns $k_{opprf_{i,t}}(g_1^{a_t})^{x_{i,t}}$ if $x_{i,t} \leq \lambda$, otherwise learns a random number. Let $o_{oprf_{i,t}}$ denote the output of $\mathcal{U}_i$ in Protocol 3.
(3) Each $\mathcal{U}_i$ computes the tag value as $\sigma_{i,t} = H(t)^{tk_i} o_{oprf_{i,t}}$ and sends it to $\mathcal{A}$.
(4) $\mathcal{A}$, computes $V_t = (\prod_i c_{i,t}) H(t)^{sk_A} = g_1^{sum_t}$, extracts sum_t from V_t and computes the aggregated tag value $\sigma_t = \prod_i (\sigma_{i,t}/k_{opprf_{i,t}}) = H(t)^{\sum_i tk_{i,t}} (g_1^a)^{sum_t}$.
(5) $\mathcal{A}$, verifies the aggregation by checking the equation $e(\sigma_t, g_2) == e(H(t), vk_1) e(g_1^{sum_t}, g_2^a)$
(6) If the verification fails, $\mathcal{A}$ discards the sum value.

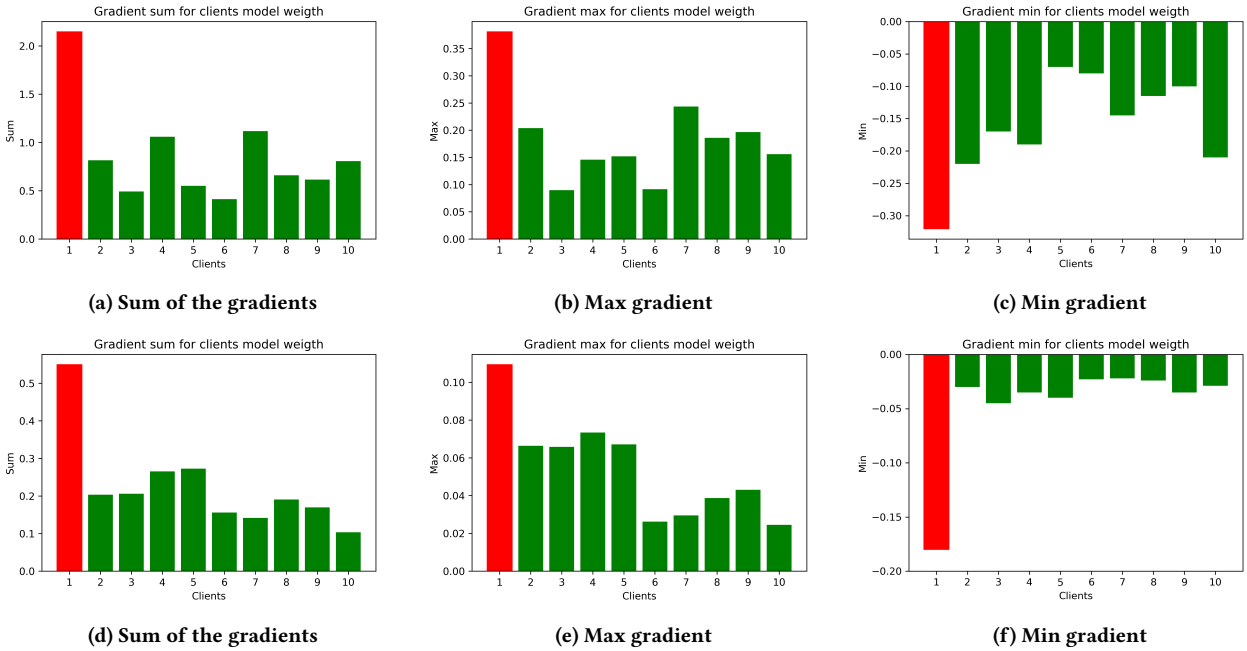
4 APPLICATION TO FEDERATED LEARNING

In this section, we show how to use our proposed secure aggregation protocol in a real case study which consists of a federated learning (FL) scheme. By definition a federated learning scheme allows the training of a model across multiple edge devices holding local data samples, without exchanging them. Particularly, we demonstrate how to prevent backdoor attacks in FL, where a malicious data owner tries to perturb the joint model only for certain inputs with specific characteristics, by using our secure aggregation protocol and therefore without disclosing the local model parameters.

Case Study: Backdoor Attack. In a backdoor attack, any participant in federated learning can replace the joint model with another so that (i) the new model is equally accurate on the federated-learning task, yet (ii) the malicious participant manages how the model performs on an attacker-chosen backdoor subtask [1]. For example, as demonstrated in [1], a backdoored image-classification model misclassifies images with certain features to an attacker-chosen class. More precisely, it was shown in [1] that a malicious participant can manage the model to misclassify all cars with a racing stripe as birds on CIFAR-10 dataset while preserving the model accuracy on other inputs. Figure 3 shows some car images that are used for the mentioned backdoor attack.

Anomalies in a Backdoor Attack: To investigate and reveal potential anomalies caused by a backdoor attack, we implemented the backdoor attack demonstrated in [1] on CIFAR-10, and analysed the local model weights for both malicious and benign participants in a comparative manner. In our experimental setting, we used 10 clients, one of which is malicious and the others are benign. Figure 4 shows some statistics of the gradients in the bias values of the last layer for 10 local models (i.e the difference between the bias values of the joint model and the local model). Figure 4a shows the absolute sum of the bias gradients for 10 clients at training round 1, where the malicious client (i.e. client 1) is indicated with red color. According to the figure, the malicious client's bias values deviate from the joint model more with respect to the benign clients' ones, which could be a sign of anomaly for backdoor attack. Similarly, Figure 4b and Figure 4c respectively show the maximum and minimum values for the model bias gradients, where the malicious client has the greatest absolute values. Figure 4d, 4e and 4f

Figure 3: Sample car images from CIFAR-10, with racing stripe, used for backdoor injection [1].



(a) Sum of the gradients **(b) Max gradient** **(c) Min gradient**

(d) Sum of the gradients **(e) Max gradient** **(f) Min gradient**

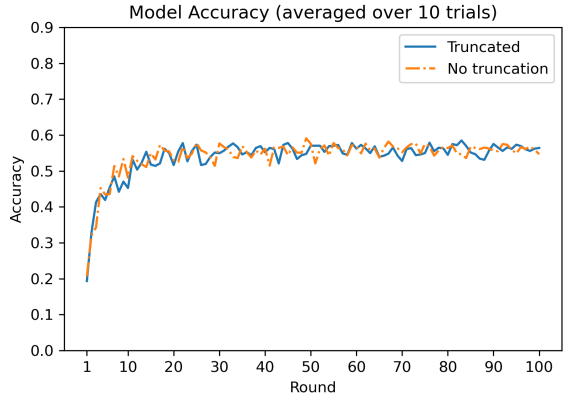
Figure 4: (a), (b), (c) at round 1 and (d), (e), (f) at round 30

depict similar graphs for round 30, which display identical pattern[2]. From these experimental results, we observe that a backdoor attack in federated learning setting creates anomaly in bias values. This could be used to detect this type of attacks. Therefore, our secure aggregation protocol enabling the blind verification of each user's input can be executed for this layer only and the aggregation of other parameters in previous layers can be performed with simpler secure aggregation solutions (such as [17]) that do not consider users as malicious.

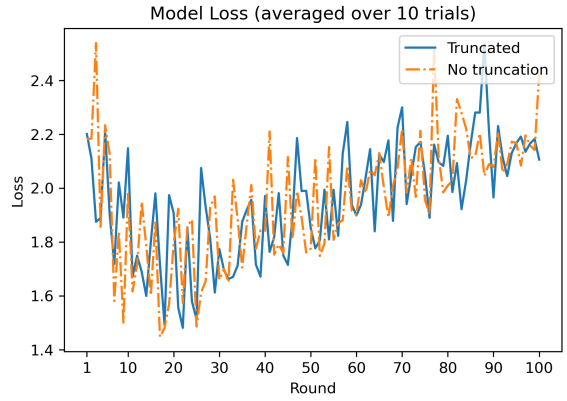
Impact of Truncation on FL Performance. To perform the actual operations of the proposed secure aggregation protocol over the model weights, these values need to be transformed into integers. A truncation process would result in speeding up the execution of our secure aggregation protocol. Therefore, we investigate the impact of truncation on the accuracy of the joint model. We conducted some experiments by applying truncation on model weights, and compared the accuracy and loss of the joint model with the non-truncated case. Figures 5a and 5b respectively show the averaged joint model accuracy and loss over 10 trials at each training round for both cases: where weights are truncated to 3 significant digits; where there is no truncation. We observe that truncation does not have a significant impact on both the model accuracy and the loss as the corresponding curves overlap to a large extend, while minor differences between the curves may be due to the stochastic nature of model training.

Impact of Multiple Malicious Users. We have also investigated the case where multiple malicious users could collaboratively perturb the model. In this new experiment, we consider that the malicious data is uniformly distributed among malicious users. We studied three main scenarios: (i) 1 malicious user with malicious data rate of 1/3, (ii) 2 malicious users with malicious data rate of 1/6 for each, and finally (iii) 5 malicious users with malicious data rate of 1/15 for each. Figure 6 exhibits the absolute sum of the bias gradients for 10 clients at training round 30, where the malicious clients are illustrated with red color. We observe that with the increase of number of malicious users, it is getting more difficult to differentiate them from legitimate ones. Therefore our solution is effective up to a certain number of malicious users (such as in [15]

[2]We do not display the parameters for each round because they show similar behaviour as the 1-st and 30-th rounds.



(a) Accuracy



(b) Loss

Figure 5: Impact of weight truncation on model accuracy and loss

where the maximal rate of malicious users is set to 50%). Further investigation on such sophisticated attacks can be performed as part of future work. Furthermore, the number of malicious users does not increase the computational and communication cost of our protocol since one OPPRF is run between each user and the aggregator.

5 SECURITY ANALYSIS

We present a security proof sketch for our secure aggregation protocol. Similarly to [12], we first consider the aggregator as an adversary and show that the proposed protocol ensures aggregator obliviousness, i.e. that the aggregator only learns the sum of users' inputs and nothing more. We further discuss the security of Protocol 2 against users following the simulation proof technique.

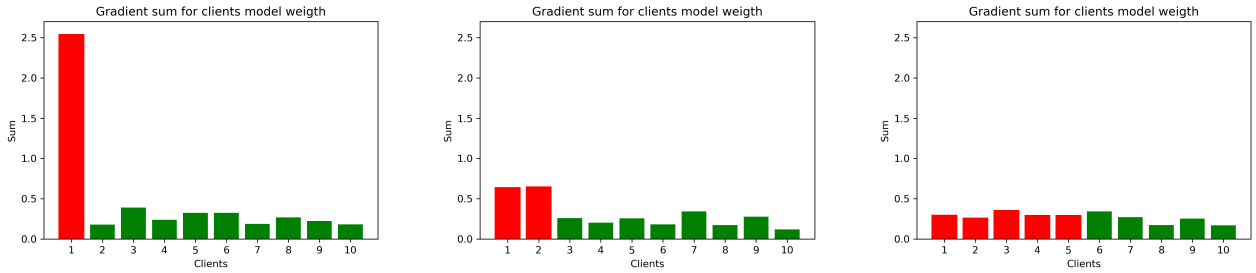
Aggregator obliviousness: The formal definition of *aggregator obliviousness* is provided in [12]. Intuitively, by receiving the encrypted inputs from users and the corresponding tags, aggregator $\mathcal{A}$ should not discover any additional information than the actual sum. Since the proposed solution basically implements PUDA with no modification, we can deduce that the protocol ensures *aggregator obliviousness*.

Secure computation: To prove the security of the computation, we need to compare the distribution of the outputs of all parties to an ideal model where a trusted third party is given the inputs from the parties, computes the function and returns the outputs and show that the view of the adversary in the real world is computationally indistinguishable from the simulated view (see [13] for definitions on computational indistinguishability and security).

- **Case when user U is corrupted**: Consider that the simulator of U, denoted as Sim_U, is given input x and output $k_{opprf}g^{ax}$ if $x < \lambda$ or random value r otherwise. In simulation, we need to show that Sim_U can generate the view of incoming messages to the user. Namely, we would like to show that the following condition holds:

$$\{Sim_U(x, OPPRF_U(x, g^a, \lambda)), OPPRF(x, g^a, \lambda)\}$$
$$\approx \{view_U^{OPPRF}(x, g^a, \lambda), output^{OPPRF}(x, g^a, \lambda)\}$$

  Sim_U first needs to run a simulator for OT that we call Sim_{UOT} for the ℓ OT executions. Further on, Sim_U executes the protocol as if user U and aggregator A would do.
  Similar to Lemma 1 in [4], the proof uses hybrid arguments starting from the real execution of OPPRF.
  – Hybrid experiment $\mathcal{H}_0$ is identical to the real execution of OPPRF.
  – $\mathcal{H}_i$ proceeds according to $\mathcal{H}_0$ with the difference that in the i-th OT execution Sim_{UOT} is run on input $(x_{\ell-i}, S_{\ell-i+1}^{x_{\ell-i}})$.
  As proved in [4], the views of Sim_U in the hybrid experiments $\mathcal{H}_i$ are indistinguishable with each other.
  We can conclude that Sim_U can generate the view of incoming messages to U and the indistinguishability guarantees that a corrupted user has no advantage on differentiating the view of U and the output of OPPRF in the real world and the view generated by Sim_U and the output of OPPRF during simulation.
- **Case where aggregator A is corrupted**: The simulator of the aggregator Sim_A is provided with the inputs to OPPRF, namely: the secret keys g^a, the threshold value λ and the output k_{opprf}. To simulate A, Sim_A acts as a sender for all OTs and sends to U $\{S_i^{x_{i-1}}\}$ for all $1 \leq i \leq l$. Sim_A further follows the execution of Protocol 2.
  Similar to Lemma 2 in [4], the proof uses hybrid arguments starting from the real execution of Protocol 2. As a first step, Sim_A randomly generates $4 \times \ell$ symmetric keys, one additional random value, and computes (S_i^0, S_i^1) according to Step 1. Sim_A runs the simulator of ℓ OT protocols. Finally Sim_A chooses a random number r', a random key k_{opprf}, and computes three values following Step 2.
  We can observe that the indistinguishability between each experiment derives from the security of OT, and the encryption algorithms. We can conclude that Sim_A can generate the view of incoming messages to the aggregator and a corrupted aggregator has no advantage on differentiating the view of the aggregator in the real world and the view generated by Sim_A during simulation.



(a) 1 malicious user **(b) 2 malicious users** **(c) 5 malicious users**

Figure 6: Sum of the bias gradients at round 30 for different number of malicious users cases

6 PERFORMANCE ANALYSIS

In this section, we analyze the complexity of our secure aggregation protocol and validate our scheme through the implementation of a proof of concept and its instantiation in a backdoor detection scenario under federated learning.

6.1 Complexity Analysis

Asymptotic complexity of our OPPRF and secure aggregation protocols. In our OPPRF protocol (Protocol 3), the aggregator (P_1) executes $O(\ell)$ OTs as the sender, $O(\ell)$ symmetric key encryptions and $O(\ell)$ exponentiations and multiplications in $\mathbb{G}_1$. Similarly, the user runs $O(\ell)$ oblivious transfer operations as the receiver, $O(\ell)$ symmetric key decryptions and $O(\ell)$ multiplication operations in $\mathbb{G}_1$. Thus, the computation complexity of our OPPRF protocol is $O(\ell)$. It is trivial to see that the communication complexity of the protocol is also $O(\ell)$. Our secure aggregation protocol (Protocol 4) requires the users to run a constant number of symmetric key operations, exponentiation and multiplication operations in $\mathbb{G}_1$ and one OPPRF protocol execution. Thus, the cost on users becomes $O(\ell)$ because of the OPPRF protocol. On the aggregator side, $O(n)$ multiplication operations in $\mathbb{G}_1$ are executed to compute V_t and σ_t value and $O(2^\ell)$ multiplication operations in $\mathbb{G}_1$ are performed to extract the aggregation result from V_t. These come in addition to $O(n)$ OPPRF executions where n is the number of users. As a result, the computation complexity of the aggregator becomes $O(2^\ell + n\ell)$. The total communication cost becomes $O(n\ell)$ because of running n OPPRF protocols in total.

Concrete complexity of our OPPRF protocol. The aggregator prepares ℓ message pairs (S_i^0, S_i^1) by executing six symmetric key encryptions, one exponentiantion and two multiplication operations in $\mathbb{G}_1$. The total number of operations to produce these message pairs approximately equals to 6ℓ symmetric key encryptions, ℓ exponentiation and 2ℓ multiplication operations. Note that the exponentiation operations (r^{ok_i}) can be done offline to lower the online computation cost. The aggregator also performs one exponentiation, one multiplication and three symmetric key encryptions in Step 3 of Protocol 3. In addition to producing these message pairs and ciphertexts, the aggregator plays the sender role and needs to perform 2ℓ asymmetric key decryptions for the ℓ OT operations. When an OT extension method [6] is used , the cost of one OT becomes 3ℓ symmetric key encryption operations under the assumption that the base OT executions are performed offline. Thus, the total cost of running our OPPRF protocol for the aggregator is $9\ell + 3$ symmetric key encryptions, $\ell + 1$ exponentiations and $2\ell + 1$ multiplications in $\mathbb{G}_1$ when OT extension is used. The user (P_2) performs 3ℓ symmetric key operations while executing ℓ oblivious transfer if OT extension is used (otherwise, ℓ asymmetric key encryptions), 3ℓ symmetric key operations to decrypt the messages and $\ell - 1$ multiplications in $\mathbb{G}_1$ in Step 5 of Protocol 3. Finally, the user performs three symmetric key decryptions and one multiplication to reach the output of the OPPRF protocol. Thus, the user performs $6\ell + 3$ symmetric key operations and ℓ multiplications in total when OT extension is used. When we look at the concrete communication complexity, we see that the total amount of data approximately equals to $2\ell(3\ell_s + \ell_p)$ where ℓ_s is the ciphertext length of the symmetric key encryption and ℓ_p is the item's length in $\mathbb{G}_1$, under the assumption that the communication cost of OT extension is approximately equal to the length of the inputs.

Concrete complexity of our secure aggregation protocol. The aggregator performs n multiplications and one exponentiation in $\mathbb{G}_1$ and one hash computation to compute V_t, 2^ℓ multiplications to extract sum_t from V_t, n inverse operations and $n-1$ multiplications in $\mathbb{G}_1$ to compute σ_t, and finally one hash, one exponentiation in $\mathbb{G}_2$, three pairing operations and one multiplication in $\mathbb{G}_T$ for validation in addition to the OPPRF executions. Thus the total computation cost on the aggregator is two hash computations, $9n\ell + 3n$ symmetric key encryptions, $3n + 2^\ell + 2n\ell - 1$ multiplications in $\mathbb{G}_1$, n inverse operations in $\mathbb{G}_1$, $n\ell + n + 1$ exponentiations in $\mathbb{G}_1$, one exponentiation in $\mathbb{G}_2$, three pairing operations and one multiplication in $\mathbb{G}_T$. When we look at the computation cost on one user, we see that the user needs to perform one hash operation, two exponentiations and one multiplication in $\mathbb{G}_1$ to compute the ciphertext and one exponentiation and one multiplication in $\mathbb{G}_1$ to compute the tag in addition to the execution of OPPRF as the receiver. Thus the total computation complexity of the user becomes one hash operation, three exponentiations and $\ell + 2$ multiplications in $\mathbb{G}_1$ and $6\ell + 3$ symmetric key operations. As a result, the total computation cost of our secure protocol considering the costs on the users and the aggregator is $n + 2$ hash computations, $15n\ell + 6n$ symmetric key encryptions, $5n + 2^\ell + 3n\ell - 1$ multiplications in $\mathbb{G}_1$, n inverse operations in $\mathbb{G}_1$, $n\ell + 4n$ exponentiations in $\mathbb{G}_1$, one exponentiation

Table 1: Execution times in seconds of our secure aggregation protocol.

# of users (n)	2^6			2^8			2^{10}		
sum_t	2^{12}	2^{14}	2^{16}	2^{12}	2^{14}	2^{16}	2^{12}	2^{14}	2^{16}
ℓ	12	14	16	12	14	16	12	14	16
Aggregation	2.2	5.9	18.0	6.2	9.7	21.3	24.2	31.0	38.4
OPPRF	58.5	67.5	76.7	238.3	272.4	308.4	1006.4	1213.1	1209.1
Verification	2.8	3.0	2.9	10.6	10.6	10.2	44.4	43.6	37.1
Total time	63.5	76.4	97.6	255.1	292.7	339.9	1075.0	1287.7	1284.6

in $\mathbb{G}_2$, three pairing operations and one multiplication in $\mathbb{G}_T$. The communication cost mainly results from sending the ciphertexts and tags by users to the aggregator and from the OPPRF protocol executed between each user and the aggregator. Thus, the total communication complexity of our secure aggregation protocol becomes $2n\ell_p + 2n\ell(3\ell_s + \ell_p)$.

6.2 Experimental Verification

We implemented our secure aggregation protocol using python and run the implementation on a standard user laptop with Intel Core i7 CPU and 32 GB RAM. We use RSA 2048 and BN256 [2] respectively as asymmetric encryption algorithm in the oblivious transfer protocol and elliptic curve for the bilinear pairing needed for verification. Instead of running every entity as a separate program thread, we simulated them as functions to be called sequentially. To simulate the communication, we used files where the message sender party writes a file and the receiving party reads that file. While measuring the computation time, we removed the overhead caused by file read/write operation costs and initial key set-up procedures. We did not use any OT extension method and also included computation cost related to r^{ok_i} in the OPPRF protocol. Note that the usage of OT extension, the execution of base OTs and computation of r^{ok_i} offline would remarkably increase the speed of our OPPRF protocol. We choose the AES encryption algorithm with 128-bit key length and the statistical correctness parameter as 48 bits in the validation of the symmetric key decryption operations.

Table 1 presents execution times of our protocol with the granularity of main functionalities (secure aggregation, OPPRF and verification) to show the dependency of their performances with respect to the number of users, the aggregated sum value and the length of the input that is aggregated. Note that the execution times presented in the table are the total execution times at the users and at the aggregator. The aggregation time includes the time for the encryption of inputs by the users, the aggregation of ciphertexts and the extraction of the sum by the aggregator. The verification time includes the time for the computation of tag values by the users, the aggregation of the tags and the verification by the aggregator. The table reports the average execution times of 20 executions of the protocol. Note that the implementation is not speed optimized and is just a proof of concept. With several optimizations and with the usage of OT extension methods, we believe that the execution times will significantly decrease.

We also integrated our secure aggregation protocol (involving negative inputs as well) into the federated learning setting described in Section 4 with the CIFAR dataset. We used our protocol for the last bias values where these values for malicious users diverge from the legitimate users' bias values (see Section 4 for details). These values are scaled and truncated in order for them to belong to the $\{-999, .., 999\}$ interval. The experimental results show that indeed our protocol can help the detection of backdoor attacks.

7 CONCLUSION

We introduced a new secure aggregation solution for a particular setting where users may provide malicious inputs to change the aggregation result on purpose. To prevent such attacks, our solution allows the aggregator to detect inputs larger than a predetermined threshold value and this without learning any information about the inputs. To develop such a solution, we proposed a new oblivious programmable pseudo-random function which can also be of independent use for further designs. This OPPRF is based on a secure comparison protocol and is executed between the aggregator and each user. Thanks to this OPPRF, the aggregator only sums legitimate values that are under/below a certain threshold and this, without discovering users' inputs. This solution only supports interval-based verification. The protocol is proved being secure under the assumption that the aggregator is honest-but-curious. We have also evaluated the performance of our solution. Finally, we have instantiated our secure aggregation protocol into a federated learning scenario with the existence of an adversary who launches a backdoor attack. In this particular scenario, the data consists of model parameters and the aggregator compares the last layer bias values with a pre-determined threshold in a privacy preserving manner. Our protocol can also be used in other use cases for anomaly detection such as the ones mentioned in [17]: sensor network aggregation, smart metering, population monitoring and sensing, etc. Future work can focus on investigating the impact of the number of malicious users in the context of the federated learning case-study. Another direction of potential research work could be to study a stronger adversary model whereby the aggregator is malicious.

ACKNOWLEDGMENT

This work was funded by the Scientific and Technological Research Council of Turkey (TUBITAK), under 1515 Frontier RD Laboratories Support Program with project no:5169902. This work has also been partially supported by the 3IA Côte d'Azur program (reference number ANR19-P3IA-0002). The authors thank Shagufta Mehnaz for her valuable comments.

REFERENCES

[1] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2018. How To Backdoor Federated Learning. *CoRR* abs/1807.00459

(2018). arXiv:1807.00459 http://arxiv.org/abs/1807.00459

[2] Paulo S. L. M. Barreto and Michael Naehrig. 2005. Pairing-Friendly Elliptic Curves of Prime Order. In *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers (Lecture Notes in Computer Science)*, Bart Preneel and Stafford E. Tavares (Eds.), Vol. 3897. Springer, 319–331. https://doi.org/10.1007/11693383_22

[3] 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, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1175–1191. https://doi.org/10.1145/3133956.3133982

[4] Michele Ciampi and Claudio Orlandi. 2018. Combining Private Set-Intersection with Secure Two-Party Computation. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings (Lecture Notes in Computer Science)*, Dario Catalano and Roberto De Prisco (Eds.), Vol. 11035. Springer, 464–482. https://doi.org/10.1007/978-3-319-98113-0_25

[5] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword Search and Oblivious Pseudorandom Functions. In *TCC*. 303–324. https://doi.org/10.1007/978-3-540-30576-7_17

[6] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently. In *CRYPTO*. 145–161. https://doi.org/10.1007/978-3-540-45146-4_9

[7] M. Joye and Libert B. 2013. A scalable scheme for privacy-preserving aggregation of time-series data. In *Financial Cryptography*.

[8] Ferhat Karakoç, Majid Nateghizad, and Zekeriya Erkin. 2019. SET-OT: A Secure Equality Testing Protocol Based on Oblivious Transfer. In *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019*. ACM, 12:1–12:9. https://doi.org/10.1145/3339252.3339264

[9] Youssef Khazbak, Tianxiang Tan, and Guohong Cao. 2020. MLGuard: Mitigating Poisoning Attacks in Privacy Preserving Distributed Collaborative Learning. In *29th International Conference on Computer Communications and Networks, ICCCN 2020, Honolulu, HI, USA, August 3-6, 2020*. IEEE, 1–9. https://doi.org/10.1109/ICCCN49398.2020.9209670

[10] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. 2017. Practical Multi-party Private Set Intersection from Symmetric-Key Techniques. In *ACM CCS*. 1257–1272. https://doi.org/10.1145/3133956.3134065

[11] I. Leontiadis, K. Elkhiyaoui, and R. Molva. 2014. Private and Dynamic Time-Series Data Aggregationwith Trust Relaxation. In *CANS*.

[12] Iraklis Leontiadis, Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva. 2015. PUDA - Privacy and Unforgeability for Data Aggregation. In *Cryptology and Network Security - 14th International Conference, CANS 2015, Marrakesh, Morocco, December 10-12, 2015, Proceedings (Lecture Notes in Computer Science)*, Michael Reiter and David Naccache (Eds.), Vol. 9476. Springer, 3–18. https://doi.org/10.1007/978-3-319-26823-1_1

[13] Yehuda Lindell. 2017. How to simulate it - A tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography. CoRR* (2017).

[14] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA (Proceedings of Machine Learning Research)*, Aarti Singh and Xiaojin (Jerry) Zhu (Eds.), Vol. 54. PMLR, 1273–1282. http://proceedings.mlr.press/v54/mcmahan17a.html

[15] Thien Duc Nguyen, Phillip Rieger, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Ahmad-Reza Sadeghi, Thomas Schneider, and Shaza Zeitouni. 2021. FLGUARD: Secure and Private Federated Learning. *IACR Cryptol. ePrint Arch.* 2021 (2021), 25. https://eprint.iacr.org/2021/025

[16] Michael O. Rabin. 1981. *How to Exchange Secrets by Oblivious Transfer*. Technical Report. Harvard Aiken Computation Laboratory Technical Report TR-81. http://eprint.iacr.org/2005/187.pdf

[17] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. 2011. Privacy-Preserving Aggregation of Time-Series Data. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*. The Internet Society. https://www.ndss-symposium.org/ndss2011/privacy-preserving-aggregation-of-time-series-data

[18] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica. 2019. Helen: Maliciously Secure Coopetitive Learning for Linear Models. In *IEEE Security and Privacy (S&P)*.

A OUR OPPRF PROTOCOL FOR NEGATIVE NUMBERS

In this appendix, we propose a solution to make our OPPRF protocol work for negative inputs, also. The aggregator can provide two encryption keys k_n and k_p (one used for negative values and one for positive values) and put these keys as a message pair to one additional OT: P_2 sets $b = 0$ or $b = 1$ if the input x is negative or positive, respectively. By running the OT, P_2 learns only one of the keys depending on the sign of the input. The remaining part of the protocol is run with the input $abs(x)$ instead of the x itself. Another modification to be applied to our protocol is that in the message pairs instead of putting $(g^a)^{0,1\times 2^i r^{ok_i}}$ P_2 puts two possible results encrypted with the sign related keys as follows: $E_n((g^a)^{-0,1\times 2^i r^{ok_i}})$ and $E_p((g^a)^{0,1\times 2^i r^{ok_i}})$. Then, P_2 will be able to have only one of the results $k_{opprf}(g^a)^{abs(x)}$ or $k_{opprf}(g^a)^{-abs(x)}$.