SoK: Secure Aggregation for Federated Learning

Mohamad Mansouri\textsuperscript{1,2} - Melek Önen\textsuperscript{2} - Wafa Ben Jaballah\textsuperscript{1} – Mauro Conti\textsuperscript{3}
Who am I?

**3rd Year PhD Student**

**CIFRE: Between** Thales **and Eurecom**

**Supervisors:** Wafa Ben Jaballah and Melek Önen

**PhD topic:** IoT Security
Outline

Secure Aggregation (SA)
- Definitions
- Historical Background
- Systematization and Categorization
  - Protocol Phases
  - Encryption-based vs MPC

Secure Aggregation in Federated Learning
- What is Federated Learning?
- Inference Attacks and solutions
- Challenges in integrating SA for FL
- Categorization and analysis of the 38 published solutions

Final Observations and Take-Aways
Defining Secure Aggregation

What is Secure Aggregation?

\[ x_\tau = \sum_{1}^{n} x_{i,\tau} \]

- \( x_{i,\tau} \) is a **private** input of user \( i \) at time period \( \tau \)
- The goal is to compute the **sum** \( X_\tau \) of all \( n \) users inputs

Example:

- Government collecting statistical information from smart electricity meters

Parties

- **Users**: Hold the private inputs
- **Aggregators**: Compute the sum (aggregate)
Security Definitions

HBC Threat Model:

➢ The basic threat model for SA

- **Honest-but-curious parties**: Users and Aggregators follow correctly the protocol but tries to learn information about the private inputs of other users.

- **Colluding parties**: Subset of the users (corrupted users) share private information with each others and with the aggregators

➢ Security Requirement:

- **Aggregator Obliviousness**: The aggregator cannot learn anything about the non-corrupted users inputs except their sum

Malicious Threat Model:

➢ A stronger threat model that is studied more recently for SA
Security Definitions - Continue

<table>
<thead>
<tr>
<th>HBC Threat Model:</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ The basic threat model for SA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Malicious Threat Model:</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ A stronger threat model that is studied more recently for SA</td>
</tr>
<tr>
<td>- <strong>Malicious Aggregator</strong>: Aggregator can manipulate the final aggregated value</td>
</tr>
<tr>
<td>- <strong>Malicious Users</strong>: Users can manipulate their own inputs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security Requirement:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- <strong>Aggregator Obliviousness</strong></td>
</tr>
<tr>
<td>- <strong>Aggregate Un-forgeability</strong>: This notion guarantees that:</td>
</tr>
<tr>
<td>1) <strong>The malicious aggregator cannot forge a false aggregate of the inputs without being detected.</strong></td>
</tr>
<tr>
<td>2) <strong>A set of malicious users cannot significantly drift the aggregation result without being detected.</strong></td>
</tr>
</tbody>
</table>
History of Secure Aggregation

Alternative names:
- Privacy-preserving aggregation
- Privacy-friendly aggregation
- Private-stream aggregation

First appeared around 2003 [HE03]

Applications
- 2003 → 2016: WSN / Smart Meters
- 2017 → now: Federated Learning

Fig. 1: The publications that used terms "secure aggregation" and "privacy-preserving aggregation" in their title or abstract.
Secure Aggregation – Systemization

Three Phases:

- **SA.Setup:** The users and the aggregator get the secret keys. Keys are generated by either a trusted party or through a distributed mechanism.

- **SA.Protect:** A user locally executes a protection algorithm to protect its input $x_{i,\tau}$ of time period $\tau$.

- **SA.Agg:** The aggregators collaboratively execute an aggregation algorithm to retrieve the sum of user inputs for time period $\tau$. In case a single aggregator exists, the aggregation algorithm is locally executed by the aggregator.
Secure Aggregation - Categorization

- Based on the **cryptographic** technology used

- For each category, we describe the three phases (Setup, Protect, Agg)

- We show the pros and cons of each

```
Secure Aggregation - Categorization

- Based on the cryptographic technology used

- For each category, we describe the three phases (Setup, Protect, Agg)

- We show the pros and cons of each
```

```
Encryption-based

- Masking-based
- AHE-based
- FE-based
- MPC-based
```
Masking-based SA

What is Masking?

- Masking uses one-time pad encryption

- It ensures perfect security if the keys is used once

Consists of two deterministic algorithms:

- \( c \leftarrow \text{Mask}(k, m) \): masks an input \( m \) using the key \( k \) \( (c = m + k \mod r) \)

- \( m \leftarrow \text{UnMask}(k, c) \): unmask the cipher text \( c \) using the same key \( k \) \( (m = c - k \mod r) \)
Masking-based SA

**SA.Setup**: each user perform DH key agreement with other user \((k_{(i,j),\tau})\) and with the aggregator \((k_{(i,0),\tau})\)

- The user key is: \(k_{i,\tau} \leftarrow \sum_{j=1}^{i-1} k_{(i,j),\tau} - \sum_{j=l+1}^{n} k_{(i,j),\tau} - k_{(i,0),\tau}\)

  The aggregator key is: \(k_{0,\tau} \leftarrow \sum_{j=1}^{n} k_{(0,j),\tau}\)

**SA.Protect**: User computes: \(c_{i,\tau} \leftarrow \text{Mask}(k_{i,\tau}, x_{i,\tau})\)

**SA.Agg**: Aggregator adds all ciphers and unmaskes: \(X_{\tau} \leftarrow \text{Unmask}(k_{0,\tau}, \sum_{1}^{n} c_{i,\tau})\)
Masking-based SA

Pros

- No need for a trusted party
- No need for pre-established secure communication channels

Cons

- Costly in terms of computation and communication (Because \textbf{SA.Setup} should be repeated for each time period)
- Cannot support dynamic users.
AHE-based SA

Multi-user AHE schemes consists of 3 PPT algorithms.

- \((k_0, \{k_i\}_{i \in U}, pp) \leftarrow AHE.\text{Setup}(\lambda)\): Generate keys and public parameters
- \(y_{i,\tau} \leftarrow AHE.\text{Enc}(pp, k_i, \tau, x_{i,\tau})\): Encrypts a message \(x_{i,\tau}\) using key \(k_i\) for time period \(\tau\)
- \(X_\tau \leftarrow AHE.\text{Agg}(pp, k_0, \{y_{i,\tau}\}_{i \in U})\): Evaluates the homomorphic operation on the \(n\) ciphertexts then decrypts the result using the decryption key \(k_0\)

Not all AHE schemes are multi-user AHE

Example of multi-user AHE - Joye-Libert Scheme [JL13]

- \(AHE.\text{Setup}(\lambda)\): choose modulus \(N\), hash \(H\) and key \(k_i\) for each user s.t. \(\sum_i k_i = -k_0\)
- \(AHE.\text{Protect}(pp, k_i, \tau, x_{i,\tau})\): \(y_{i,\tau} = (1 + x_{i,\tau}N)H(\tau)^{k_i} \mod N^2\)
- \(AHE.\text{Agg}(pp, k_i, \{y_{i,\tau}\}_{i \in U})\): \(V_\tau = H(\tau)^{k_0} \prod_i y_{i,\tau}\)

\[X_\tau = \frac{V_\tau^{-1}}{N}\]
AHE-based SA

> **SA.Setup:** A trusted party runs the *AHE.Setup* algorithm and distributed the keys (this is only executed once)

> **SA.Protect:** User runs *AHE.Enc* and sends the protected input to the aggregator

> **SA.Agg:** Aggregator runs *AHE.Agg*
AHE-based SA

Pros

- Long-term keys (no need to re-setup)
- No need for pre-established secure communication channels

Cons

- Costly in terms of computation
- Cannot support dynamic users.
### What is FE?

- Encryption scheme that enables the server to learn a function on a user data
- A special type is FE for Inner Product (IP)
  
  \[
  IP(x, y) = \sum_i x[i]y[i]
  \]

- A variant of FE is multi-input FE (MIFE) where inputs are provided by \( n \) users.
- MIFE consists of 4 PPT algorithms:
  - \( \{msk, \{k_i\}_{\forall i}, pp\} \leftarrow MIFE.\text{Setup}(\lambda) \): Generates a master key and all user keys
  - \( c_{i,\tau} \leftarrow MIFE.\text{Enc}(pp, k_i, x_{i,\tau}) \): Encrypts a message \( x_{i,\tau} \) using key \( k_i \)
  - \( dk_{\tau} \leftarrow MIFE.\text{DKGen}(pp, msk, y_{\tau}) \): Generate a decryption key from the master key
  - \( IP([x_1, ..., x_n], y_{\tau}) \leftarrow MIFE.\text{Dec}(pp, dk_{\tau}, [c_{1,\tau}, ..., c_{n,\tau}], y_{\tau}) \): Computes the inner product of all users inputs with the vector \( y_{\tau} \) using the decryption key
**Functional-Encryption-based SA**

**SA.Setup:** A trusted party runs the $\textit{MIFE.Setup}$ algorithm and distributed the keys (this is only executed once)

**SA.Protect:** User runs $\textit{MIFE.Enc}$ and sends the protected input to the aggregator

**SA.Agg:** The trusteeed party runs $\textit{MIFE.DKGen}$ by setting the vector $\gamma_T = [1, 1, 1, 1, 1]$ and sends the decryption key to the aggregator which runs Aggregator runs $\textit{MIFE.Dec}$
Functional-Encryption-based SA

Pros
- Light-weight operations
- No need for pre-established secure communication channels

Cons
- Require the trusted dealer to stay online
Multi-party-Computation-based SA

What is MPC ?

- No need for user keys
- private messages are split into shares and distributed to multiple servers
- $t$ of the servers can collaborate to reconstruct the private message
- a.k.a. t-out-of-n sharing

Consists of two algorithms:
- $\{[s]_i\}_{i\in[1,\ldots,n]} \leftarrow \text{MPC.Share}(s, t, n)$: It splits the secret $s$ to $n$ shares
- $s \leftarrow \text{MPC.Recon}([s]_{i\in U'\subset[1,\ldots,n]})$: It reconstructs the secret $s$ from a subset of more than $t$ shares
Multi-party-Computation-based SA

**SA.Setup:** Not required

**SA.Protect:** User runs *MPC.Share* and sends each share to a different aggregator

**SA.Agg:** Each aggregator sums locally the shares. Then, one of the aggregator collects all the summed shares and execute *MPC.Recon*
Multi-party-Computation-based SA

**SA.Setup**

- Share
- $U_i\rightarrow A_1\rightarrow \ldots \rightarrow A_m$

**SA.Protect**

**SA.Aggregate**

- Requires pre-established secure communication channels
- High communication overhead

**Pros**

- No need for trusted dealer
- Light operations
### Table 1. Table comparing the different categories of secure aggregation. $TP$ stands for trusted third party. SC stands for pre-established secure channels. Dynamic users property shows wether the aggregation can be performed with only a subset of the users. Comp. stands for computation cost on users and aggregators. Comm. stands for communication cost between users and aggregators. • means that the property is attained.
Federated Learning (FL)

- A technique to train a ML model on multiple private datasets without sharing the data

- Consists of \( n \) FL clients and a FL server

Fig. 2: One FL round with 3 FL clients
Inference Attacks

- An HBC aggregator
- Leak information about the private data set of a client from its locally trained model

Solutions

Differential Privacy
- Affects Accuracy

Trusted Execution Environment
- Requires Trust

Secure Aggregation

- Train ML model on private data
- Send agg. model to clients
- Send ML models to server
- Aggregate models
SA to Mitigate Inference Attack on FL

- FL clients are users
- FL server is the aggregator (or set of aggregators)
- Is this solution practical?

Fig. 3: SA integrated in FL. One FL round with 3 FL clients
SA for FL: Challenges

- Chall1: Clients Dropouts
- Chall2: Inputs have high dimension
- Chall3: Huge number of clients
- Chall4: Privacy Attacks that Bypass SA
- Chall5: Malicious Users
- Chall6: Malicious Aggregator
If users drop, aggregation cannot be performed

- Masking-based SA
- AHE-based SA
- Sum of the keys equals to aggregator key

Solutions: Fault-Tolerant SA

- Masking-based SA: [BIK+17], [SSV+21], [YSH+21]
- Use secret sharing to share the DH secret keys of users
- If user failed, generate its masks
The inputs are the trained model parameters

- Vectors of high dimension
- Leads to large communication overhead

Solutions:

- AHE-based SA: packing and batch encryption [PAH+18], [LCV19], [ZLX+20]
- Masking-based SA: Auto-tuned quantization by changing the modulo [BSK+19], [EA20]
- FE-based SA: All or nothing transformation [WPX+20]
- All category techs: Ternary FL [DCSW20]
SA for FL - Chall3: Huge number of clients

- New large scale apps: Gboard [YAE+18]
- Tens of thousands of clients
- How to synchronize the aggregation

Solutions:

➤ Group clients into multiple groups and run multiple SA instances [BEG+19], [BBG+20], [SGA21b], [SMH21], [JNMALC22]

➤ Adapt SA for asynchronous FL [SAGA21]
SA for FL – Chall4: Privacy Attacks that Bypass SA

The aggregated model is public information!
SA is not designed to protect the aggregate
Adversaries can infer information from the aggregate

- Inference attacks: infer data samples
- Multi-round attacks: Bypasses SA

Solution

- Distributed Differential Privacy (DDP) [TBA+19], [SSV+21]
- Batch Partitioning [SAG+21]
Malicious users perform poisoning attacks

- Manipulates the input to install a backdoor [STS16]

Solution

- Verify user input before including it
- But isn’t it protected?!
- MPC-based SA: computed and compare cosine distance over protected inputs [KTC20], [NRY+21]
- Masking-based SA: Hierarchal aggregation to verify intermediate results [ZLYM21],[VXK21]
- AHE-based SA:
  - Using OPPRF to compare with threshold[KOB21]
  - Using commitments scheme to compute the distance [BLV+21]
A malicious aggregator can send a false aggregation results

- Force users to learn back doored ML model
- Impact privacy also: bypass SA [PFA21]

Solutions: Aggregator generate a proof of the result using:

- Homomorphic hash functions (HHF) [ZFW+20], [XLL+20]
- Commitment scheme [GLL+21]
### SA for FL – Systematization

<table>
<thead>
<tr>
<th>FT</th>
<th>Comm. C2</th>
<th>Scale C3</th>
<th>Privacy C4</th>
<th>Mal. users C5</th>
<th>Mal. agg. C6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>[BIK+17]</td>
<td>[BSK+19]</td>
<td>[BEG+19]</td>
<td>[KLS21]</td>
<td>[ZLYM21]</td>
</tr>
<tr>
<td></td>
<td>Google</td>
<td></td>
<td>[BBG+19]</td>
<td></td>
<td>[GLL+21]</td>
</tr>
<tr>
<td>Masking</td>
<td>[SSV+21]</td>
<td>[SMH21]</td>
<td>[FMLF21]</td>
<td>[VML21]</td>
<td>[VXK21]</td>
</tr>
<tr>
<td></td>
<td>University of Southern California</td>
<td>[JNMLAC22]</td>
<td>[SSV+21]</td>
<td></td>
<td>[XLL+20]</td>
</tr>
<tr>
<td></td>
<td>[YGSH+21]</td>
<td>[SGA21b]</td>
<td>[SAG+21]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FE</td>
<td>[XBZ+19]</td>
<td></td>
<td></td>
<td>[SGA21a]</td>
<td></td>
</tr>
<tr>
<td>IBM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AHE</td>
<td>[LCV+19]</td>
<td></td>
<td>[TBA+19]</td>
<td>[BLV+21]</td>
<td>[ZFW+20]</td>
</tr>
<tr>
<td></td>
<td>[PAH+18]</td>
<td></td>
<td></td>
<td>[KOB21]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ZEX+20]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPC</td>
<td>[KRRK20]</td>
<td>[BT20]</td>
<td></td>
<td>[KTC20]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[DCSW20]</td>
<td></td>
<td>[NRY+21]</td>
<td></td>
</tr>
</tbody>
</table>
SA for FL – Our observations

- **O1**: 20 masking-based SA solutions that tackles different challenges. Combining them?
- **O2**: AHE-based SA is not well explored
- **O3**: Non-secure aggregation AHE-based solutions [PAH+18, LCV19, ZLX+20, ZFW+20]
- **O4**: SA need additional privacy mechanism (DDP, multi-round privacy)
- **O5**: Scalable (w.r.t. # users) solutions in the malicious user model are open problem
- **O6**: Scalable (w.r.t. the #of model parameters) solutions in the malicious aggregator model are open problem
Extended definition of Secure Aggregation for FL

The extended SA.Protect phase:

- First users add DP noise to the input
- Then perform the basic protection algorithm
- Finally, generate a proof of the input

The extended SA.Agg phase:

- First, aggregator verifies the protected input and decide whether to accept it
- Then performs the basic aggregation algorithm
- Finally, generate a proof of the aggregation

A new SA.Verify phase:

- Users verify the aggregated value and decide weather to accept it
Question ?