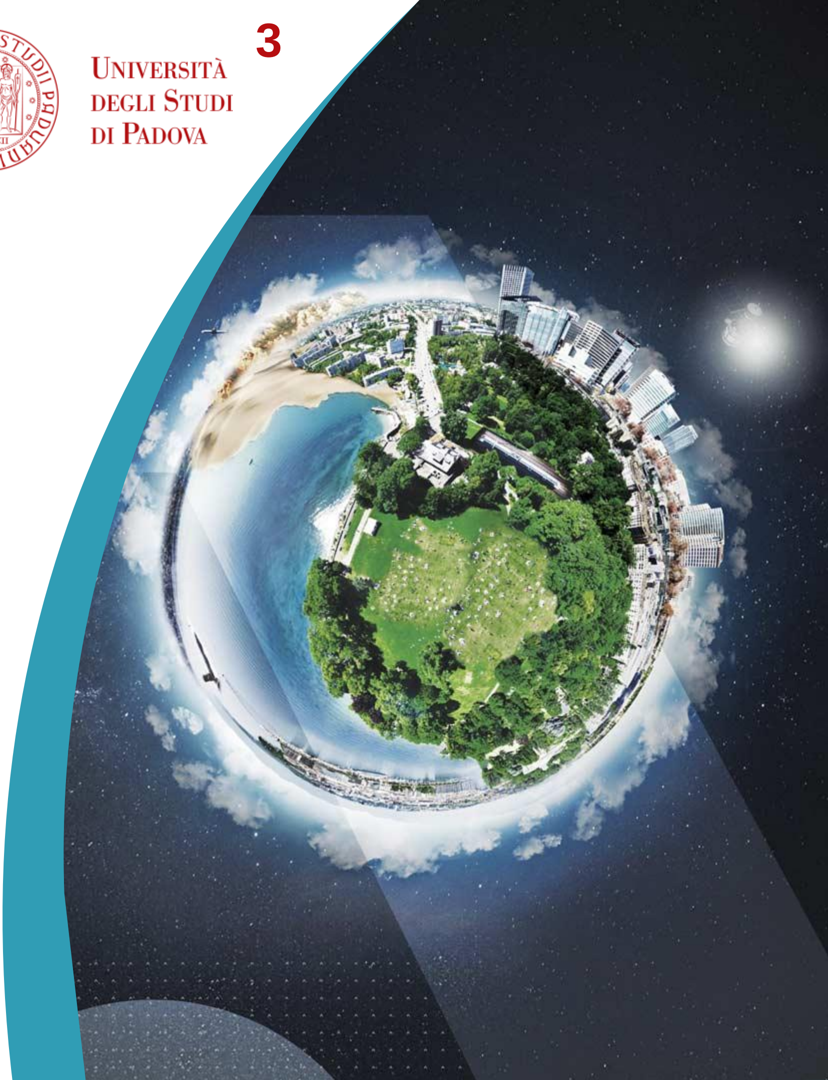
THALES[1]
Building a future we can all trust

EURECOM[2]
Sophia Antipolis

Università degli Studi di Padova[3]

SoK: Secure Aggregation for Federated Learning

***Mohamad Mansouri*[1,2]** - *Melek Önen*[2] - *Wafa Ben Jaballah*[1] – *Mauro Conti*[3]



Who am I?

- 3rd Year PhD Student
- CIFRE: Between THALES and EURECOM Sophia Antipolis
- 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 τ
- The goal is to compute the **sum** X_τ 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

HBC Threat Model:

- The basic threat model for SA

Malicious Threat Model:

- A stronger threat model that is studied more recently for SA
  - **Malicious Aggregator**: Aggregator can manipulate the final aggregated value
  - **Malicious Users**: Users can manipulate their own inputs
- **Security Requirement:**
  - ***Aggregator Obliviousness***
  - ***Aggregate Un-forgeability:*** *This notion guarantees that:*

1) *The malicious aggregator cannot forge a false aggregate of the inputs without being detected.*
2) *A set of malicious users cannot significantly drift the aggregation result without being detected.*

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

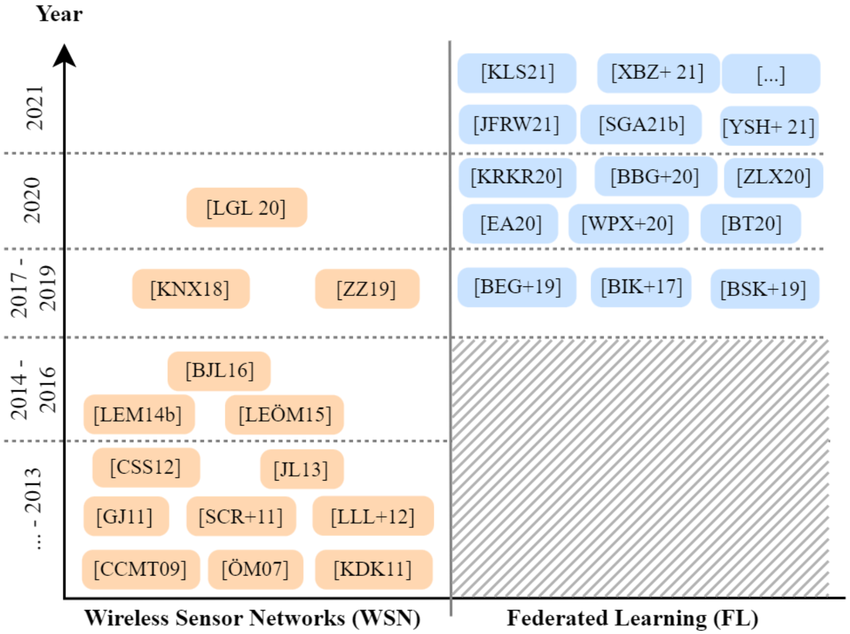


Fig. 1: The publications that used terms “secure aggregation” and “privacy-preserving aggregation” in their title or abstract.

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 τ.

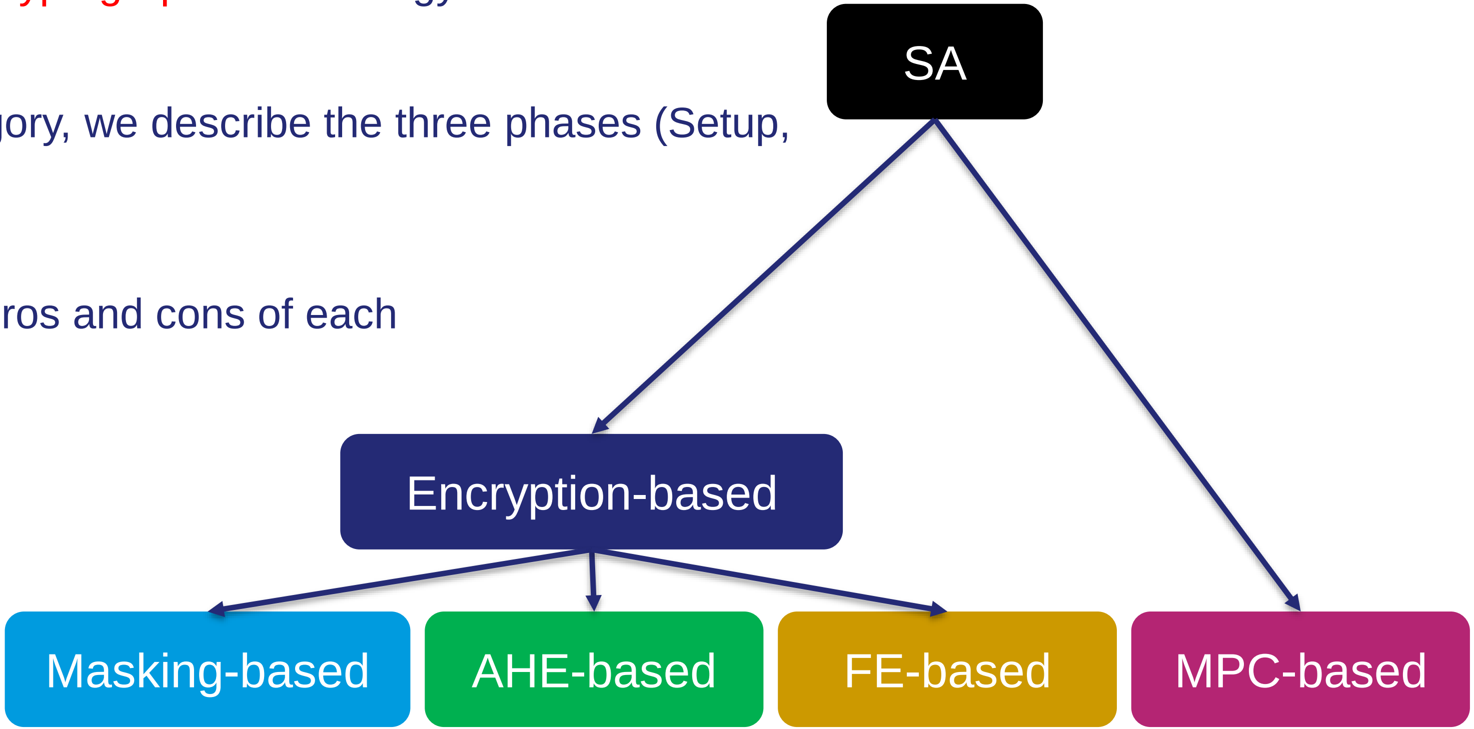
- **SA.Agg**: The aggregators collaboratively execute an aggregation algorithm to retrieve the sum of user inputs for time period τ. In case a single aggregator exists, the aggregation algorithm is locally executed by the aggregator.

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



Masking-based SA

What is Masking?

- Masking uses one-time pad encryption

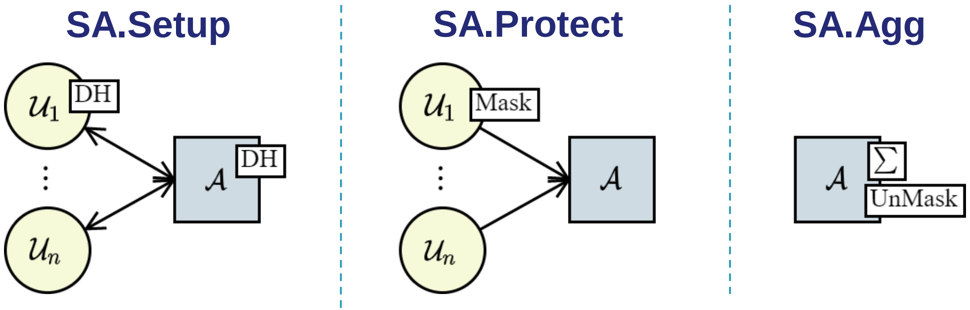
- It ensures perfect security if the keys is used once

- Consists of two deterministic algorithms:

  - $\boldsymbol{c} \leftarrow \boldsymbol{Mask}(\boldsymbol{k}, \boldsymbol{m})$: masks an input m using the key k ($c = m + k \bmod r$)
  - $\boldsymbol{m} \leftarrow \boldsymbol{UnMask}(\boldsymbol{k}, \boldsymbol{c})$: unmasks the cipher text c using the same key k ($m = c - k \bmod r$)

Masking-based SA

Masking



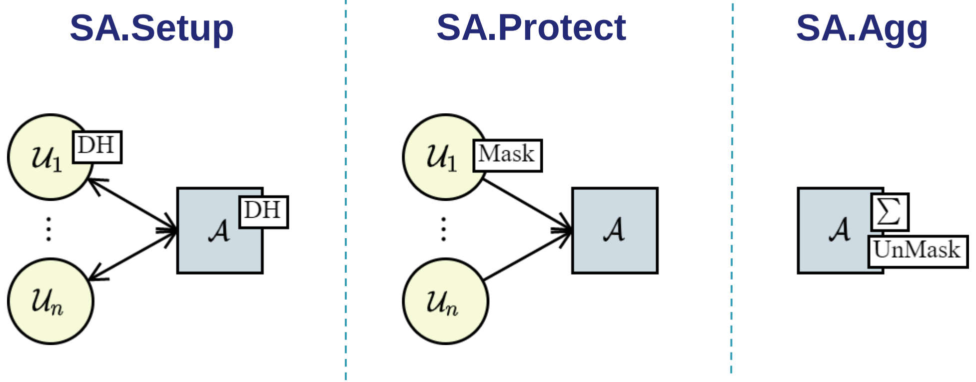
- **SA.Setup:** each user perform DH key agreement with other user ($k_{(i,j),\tau}$) and with the aggregator ($k_{(i,0),\tau}$)

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

- **SA.Protect:** User computes: $c_{i,\tau} \leftarrow Mask(k_{i,\tau}, x_{i,\tau})$
- **SA.Agg:** Aggregator adds all ciphers and unmasks: $X_\tau \leftarrow Unmask(k_{0,\tau}, \sum_1^n c_{i,\tau})$

Masking-based SA

Masking



Pros

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

Cons

- Costly in terms of computation and communication (Because **SA.Setup** should be repeated for each time period)
- Cannot support dynamic users.

AHE-based SA

AHE-based

Multi-user AHE schemes consists of 3 PPT algorithms.

- $(k_0, \{k_i\}_{\forall i \in U}, pp) \leftarrow AHE.Setup(\lambda)$: Generate keys and public parameters
- $y_{i,\tau} \leftarrow AHE.Enc\left(pp, k_i, \tau, x_{i,\tau}\right)$: Encrypts a message $x_{i,\tau}$ using key k_i for time period τ
- $X_\tau \leftarrow AHE.Agg\left(pp, k_0, \{y_{i,\tau}\}_{\forall i \in U}\right)$: 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]

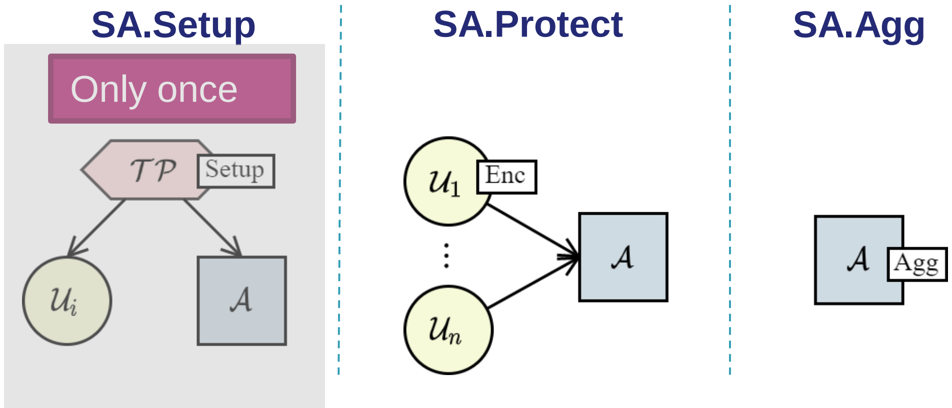
$AHE.Setup(\lambda)$: choose modulus N, hash H and key k_i for each user s. t. $\sum_i k_i = -k_0$

$AHE.Protect\left(pp, k_i, \tau, x_{i,\tau}\right)$: $y_{i,\tau} = \left(1 + x_{i,\tau} N\right) H(\tau)^{k_i}$ mod N^2

$AHE.Agg\left(pp, k_i, \{y_{i,\tau}\}_{\forall i \in U}\right)$: $V_\tau = H(\tau)^{k_0} \prod_i y_{i,\tau}$ $\quad X_\tau = \frac{V_\tau - 1}{N}$

AHE-based SA

AHE-based

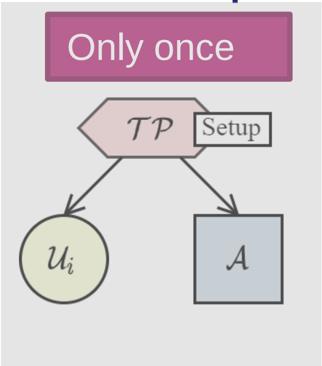


- **SA.Setup:** A trusted party runs the $AHE.Setup$ algorithm and distributed the keys (this is only executed once)
- **SA.Protect:** User runs $AHE.Enc$ and sends the protected input to the aggregator
- **SA.Agg:** Aggregator runs $AHE.Agg$

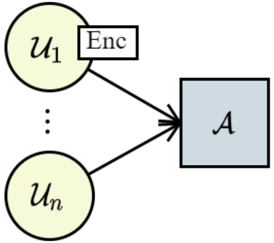
AHE-based SA

AHE-based

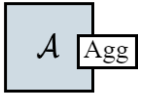
SA.Setup



SA.Protect



SA.Agg



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.

Functional-Encryption-based SA

What is FE ?

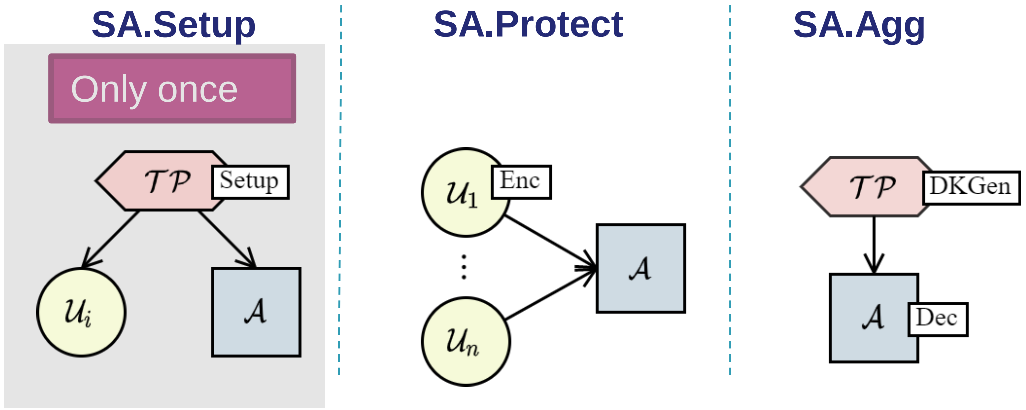
- Encryption scheme that enables the server to learn a function on a user data
- A special type is FE for Inner Product (IP)

$$IP(x,y) = \sum_i x[i]y[i]$$

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

Functional-Encryption-based SA

FE-based

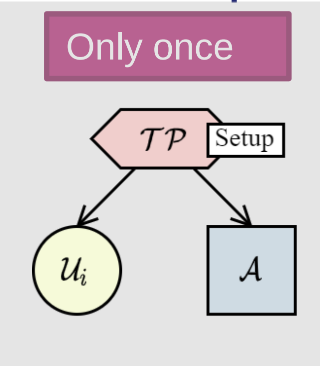


- **SA.Setup:** A trusted party runs the $MIFE.Setup$ algorithm and distributed the keys (this is only executed once)
- **SA.Protect:** User runs $MIFE.Enc$ and sends the protected input to the aggregator
- **SA.Agg:** The trusteed party runs $MIFE.DKGen$ by setting the vector y_τ =[1,1,1,1,1] and sends the decryption key to the aggregator which runs Aggregator runs $MIFE.Dec$

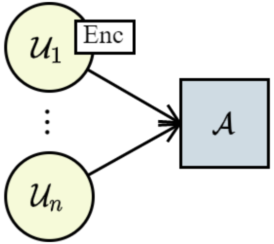
Functional-Encryption-based SA

FE-based

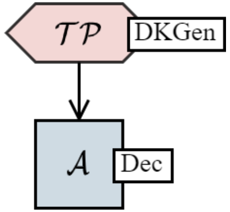
SA.Setup



SA.Protect



SA.Agg



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

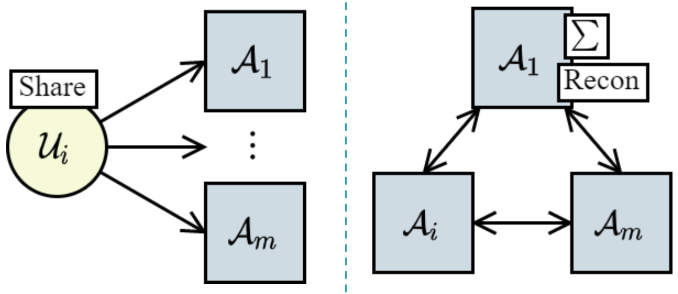
What is MPC ?

- 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:
  - $\{[s]_i\}_{\forall i \in [1,..,n]} \leftarrow MPC.Share(s,t,n)$: It splits the secret s to n shares
  - $s \leftarrow MPC.Recon\left(\{[s]_i\}_{i \in U' \subset [1,..,n]}\right)$: It reconstructs the secret s from a subset of more than t shares

Multi-party-Computation-based SA

MPC-based

SA.Setup | SA.Protect | SA.Agg

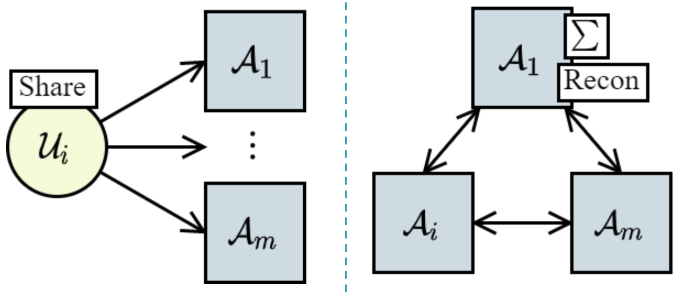


- **SA.Setup:** Not required
- **SA.Protect:** User runs $MPC.Share$ and sends each share to a different aggregator
- **SA.Agg:** Each aggregator sums locally the shares. Then, one of the aggregator collects all the summed shares and execute $MPC.Recon$

Multi-party-Computation-based SA

MPC-based

SA.Setup | SA.Protect | SA.Agg



Pros

- No need fro trusted dealer
- Light operations

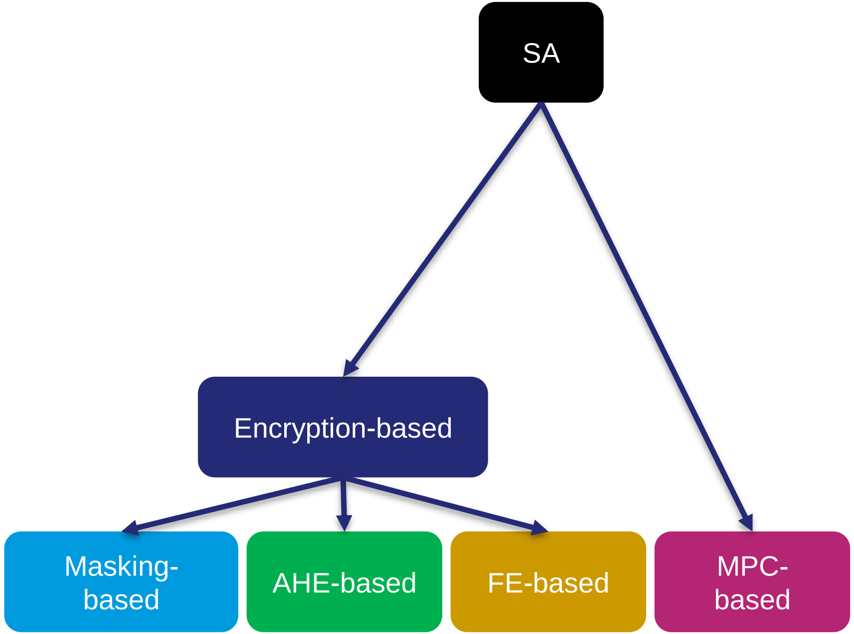
Cons

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

SA Categorization - Summary

	Schemes	No $\mathcal{TP}$ required	No SC required	Dynamic users	Comp.	Comm.
Encryption	Masking (DC-net)	●	●	○	○	○
	AHE	◐	●	○	○	◐
	FE	○	●	●	●	●
MPC	n-out-of-n SS t-out-of-n SS	●	○	●	●	○

Table 1. Table comparing the different categories of secure aggregation. $\mathcal{TP}$ 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**

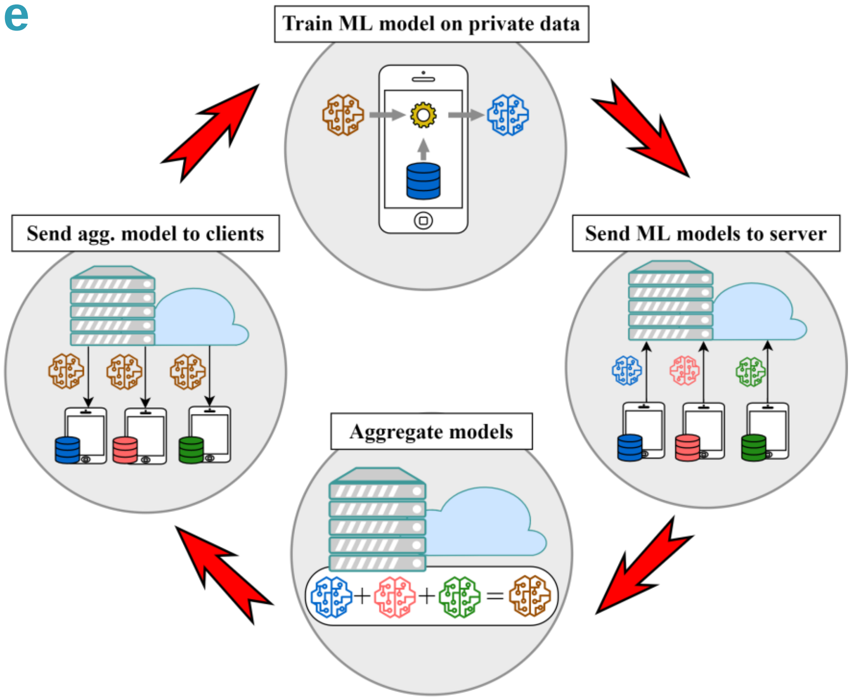


Fig. 2: One FL round with 3 FL clients

Inference Attacks

An HBC aggregator

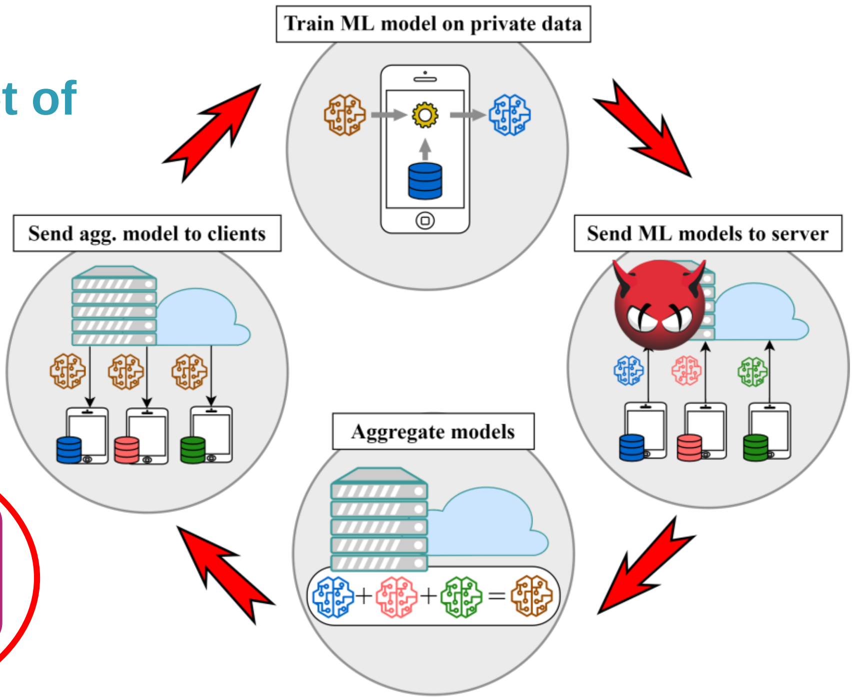
Leak information about the private data set of a client from its locally trained model

Solutions

Differential Privacy — Affects Accuracy

Trusted Execution Environment — Requires Trust

Secure Aggregation



SA to Mitigate Inference Attack on FL

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

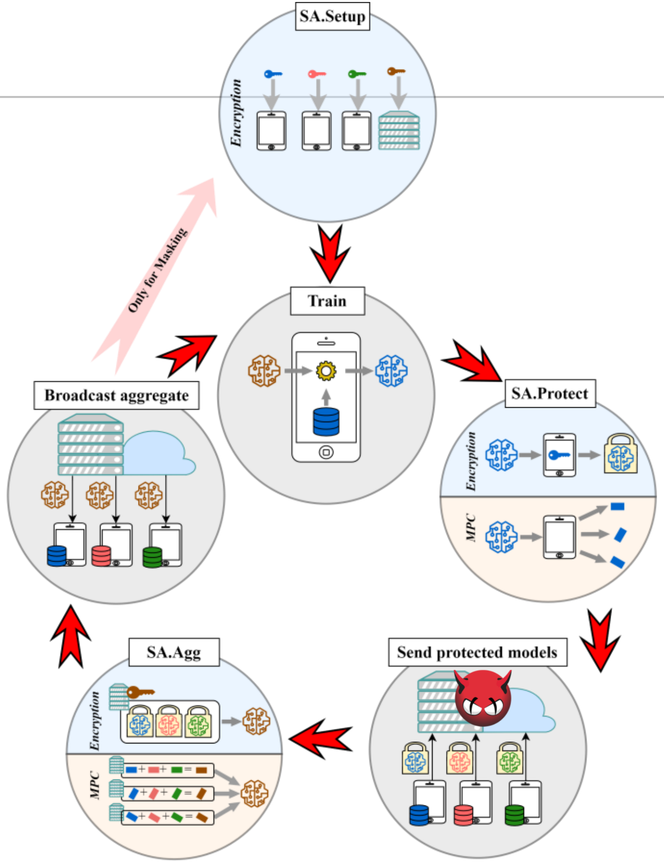
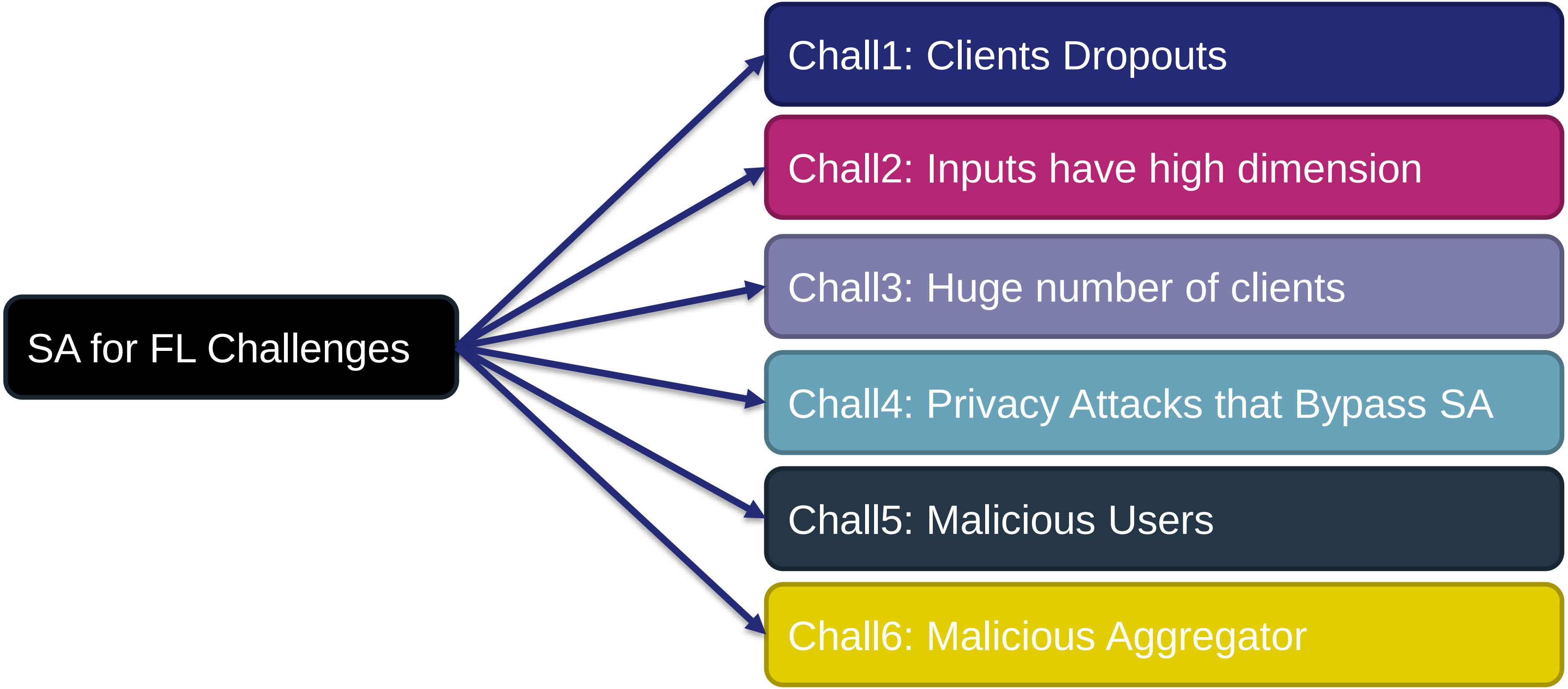


Fig. 3: SA integrated in FL. One FL round with 3 FL clients

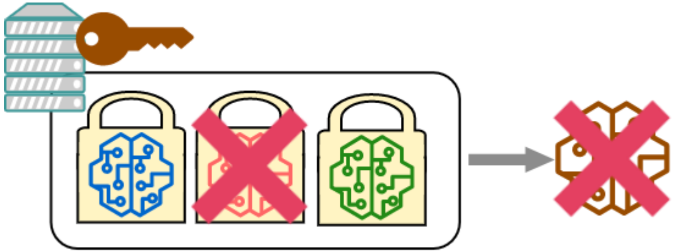
SA for FL: Challenges



Chall1: Clients Dropouts

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

SA for FL – Chall2: Inputs have high dimension

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]

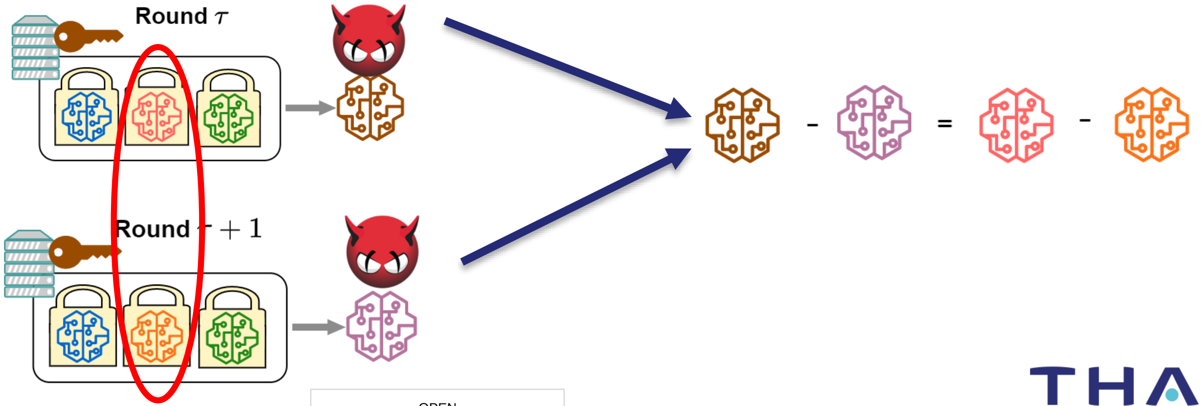
SA for FL - Chall3: Huge number of clients

- **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

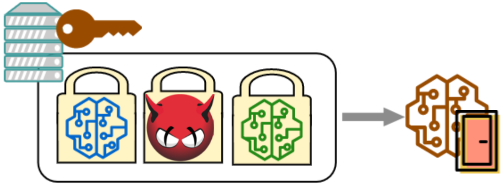


Solution

- Distributed Differential Privacy (DDP) [TBA+19], [SSV+21]
- Batch Partitioning [SAG+21]

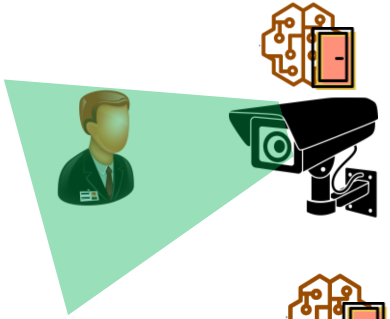
SA for FL – Chall5: Malicious Users

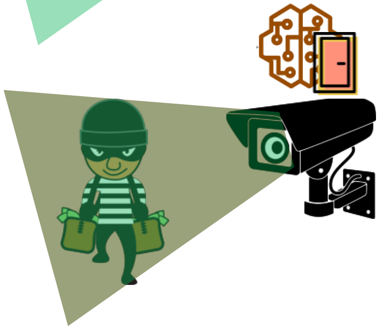
Malicious users perform poisoning attacks



- Manipulates the input to install a backdoor [STS16]

Solution



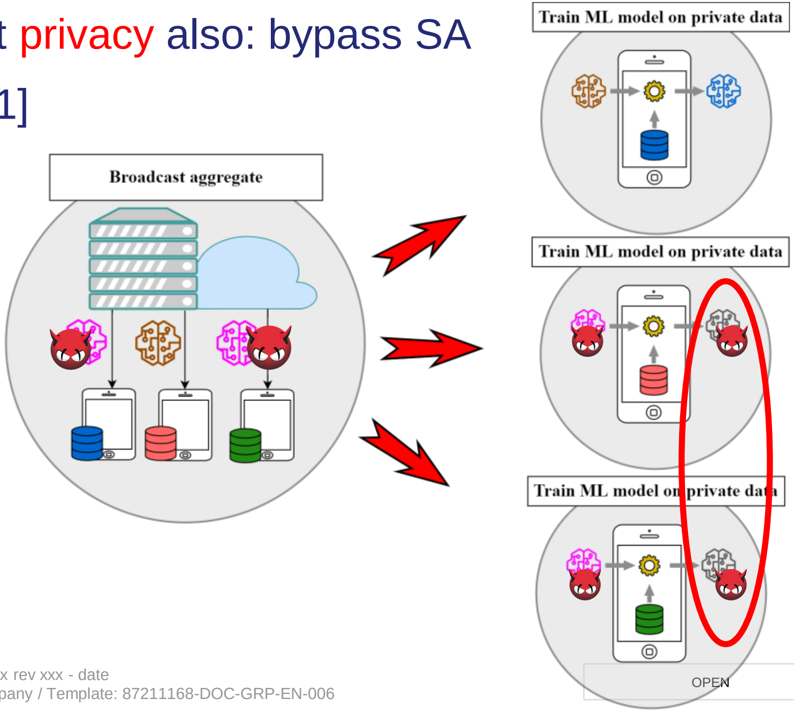


- Verify user input before including it
- But isn’t it protected?!
- MPC-based SA: computed and compare cosine distance over protected inputs [KTC20], [NRY+21]
- Masking-based SA: Hierarchal aggregation to verify intermediate results [ZLYM21],[VXK21]
- AHE-based SA:

  - Using OPPRF to compare with threshold[KOB21]
  - Using commitments scheme to compute the distance [BLV+21]

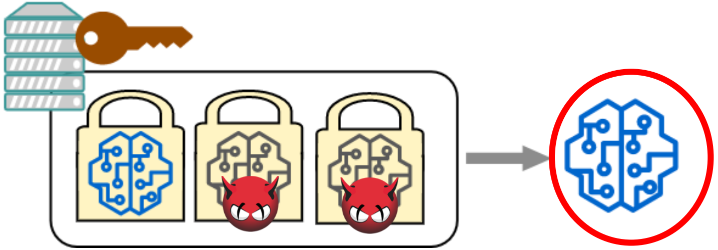
A malicious aggregator can send a false aggregation results

- 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

	FT $\mathbb{C}_1$	Comm. $\mathbb{C}_2$	Scale $\mathbb{C}_3$	Privacy $\mathbb{C}_4$	Mal. users $\mathbb{C}_5$	Mal. agg. $\mathbb{C}_6$
Masking	[BIK+17] **Google** [SSV+21] [YSH+21] **University of Southern California**	[BSK+19]	[BEG+19] [BBG+19] [SMH21] [JNMALC22] [SGA21b] [SAGA21]	[KLS21] [FMLF21] [SSV+ 21] [SAG+21]	[ZLYM21] [VXK21] [SGA21a]	[GLL+21] [XLL+20]
FE	[XBZ+19] **IBM**					
AHE		[LCV19] [PAH+ 18] [ZLX+ 20]		[TBA+19]	[BLV+ 21] [KOB21]	[ZFW+20]
MPC	[KRKR20]	[BT20] [DCSW20]			[KTC20] [NRY+ 21]	

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:

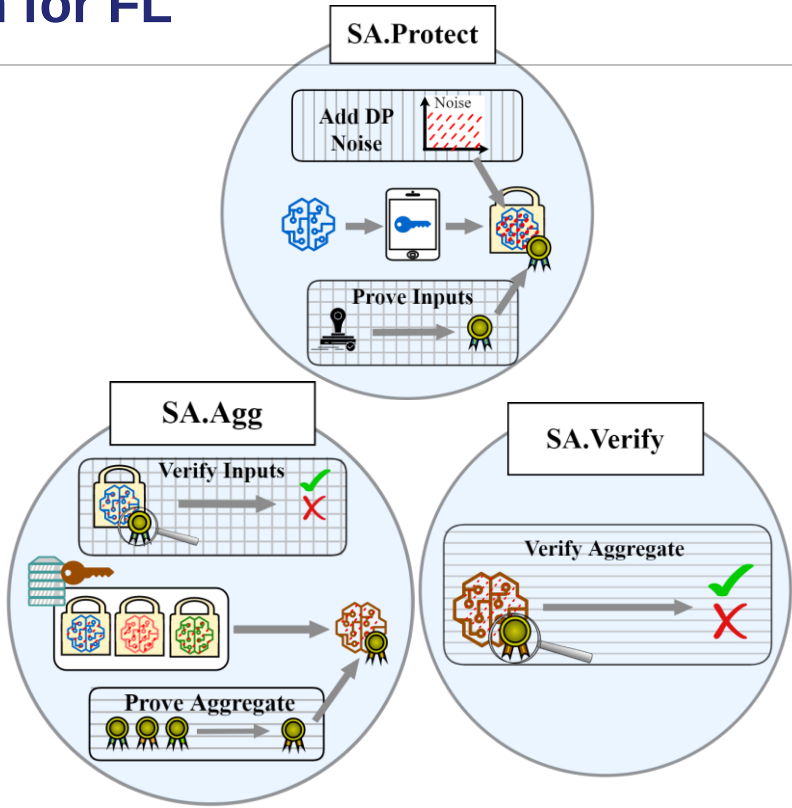
- First users add DP noise to the input
- Then perform the basic protection algorithm
- Finally, generate a proof of the input

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 ?