THALES
Building a future we can all trust

EURECOM
Sophia Antipolis

Secure and Fault-Tolerant Aggregation for FL

Mohamad Mansouri
Melek Önen
Wafa Ben Jaballah

Outline

Overview of Secure Aggregation

- Definition and Applications
- Threat Model
- Integration in Federated Learning

Fault-Tolerant Secure Aggregation

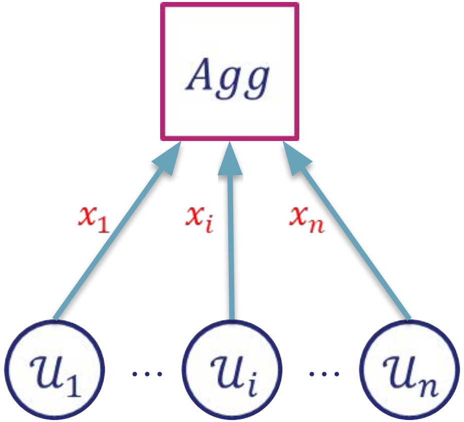
- Joye-Libert (JL) SA scheme
- User drop problem
- Our Threshold JL scheme
- Our protocol

Comparison with SOTA

Conclusion

What is Secure Aggregation?

$$X = \sum_{i=1}^{n} x\_i$$



- x_i is a private user input
- How to compute X without trusting Agg ?

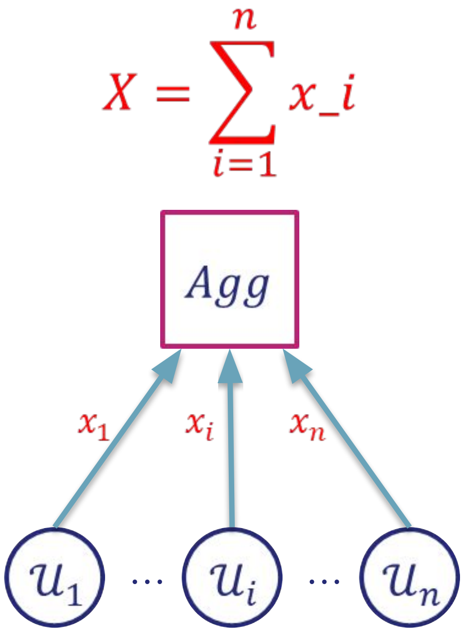
Secure Aggregation Threat Model

Threat Model

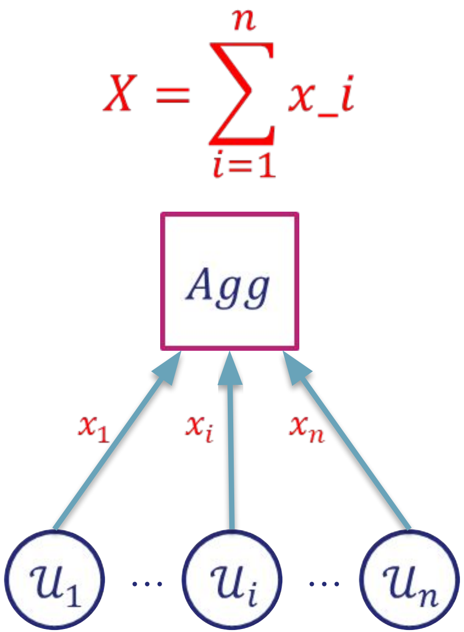
- **Malicious Aggregator colluding with users**

Security Requirement:

- **Aggregator Obliviousness** : The aggregator cannot learn more than the **sum** of the honest users inputs



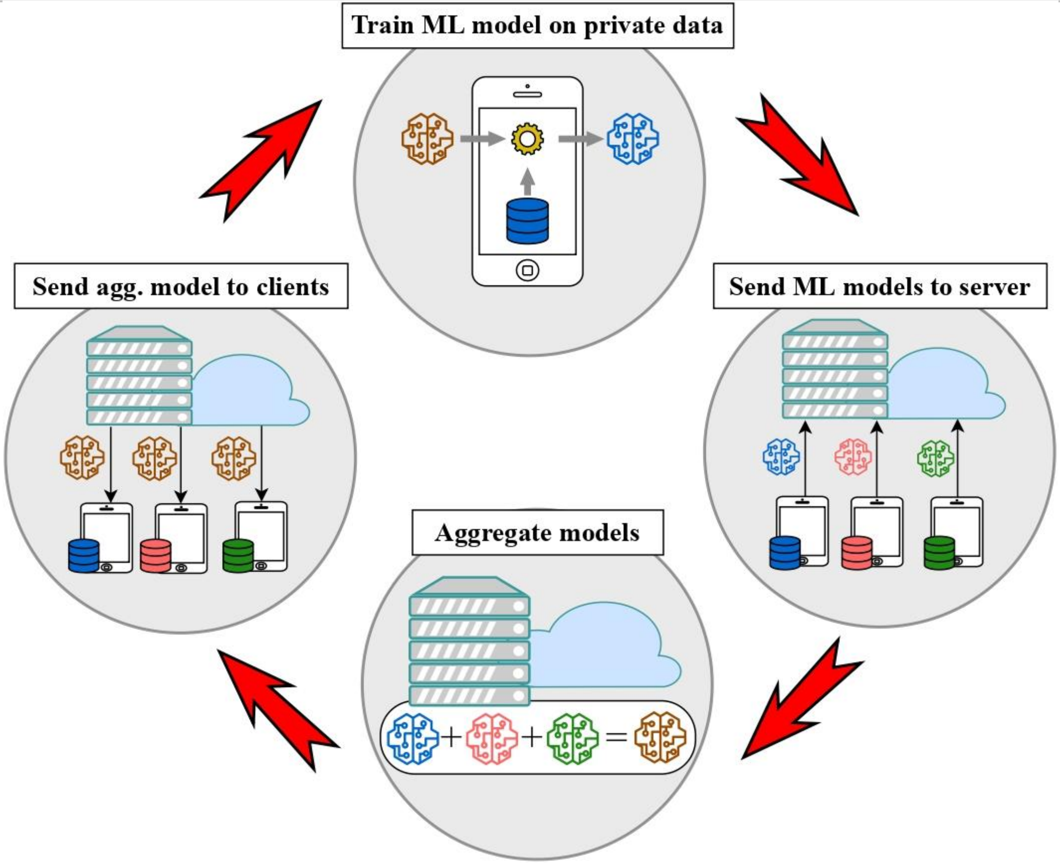
Application of Secure Aggregation



- Voting Systems: $x_i \in \{0, 1\}$
- Rating Systems: $x_i \in \{0, 1, \ldots, 10\}$
- Smart Meters: $x_i \in [0, N_{max}]$
- Federated Learning: $x_i \in [0, N_{max}]^m$

Federated Learning

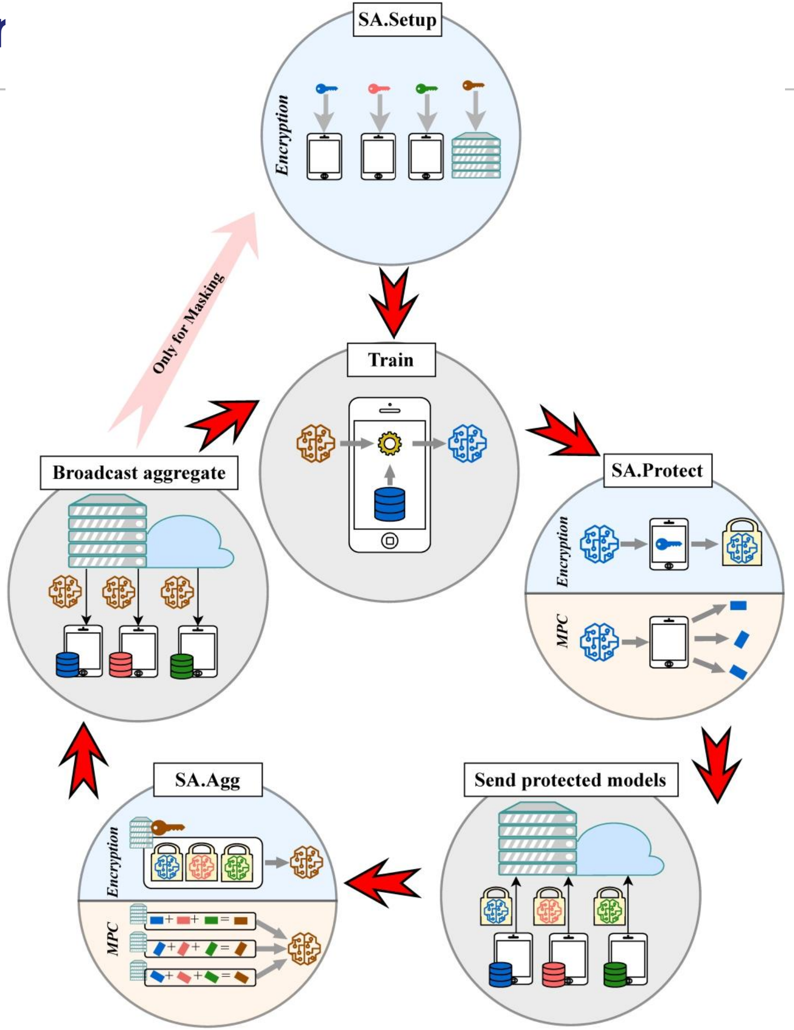
What is FL?



Secure Aggregation for Federated Learnir

Inference Attacks

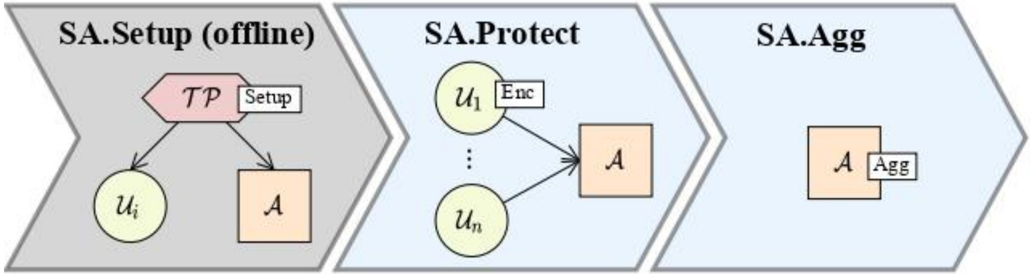
- Leak information about the private data set from the trained model
- Membership Inference Attacks
- Reconstruction Attacks
- Data Properties Inference Attacks

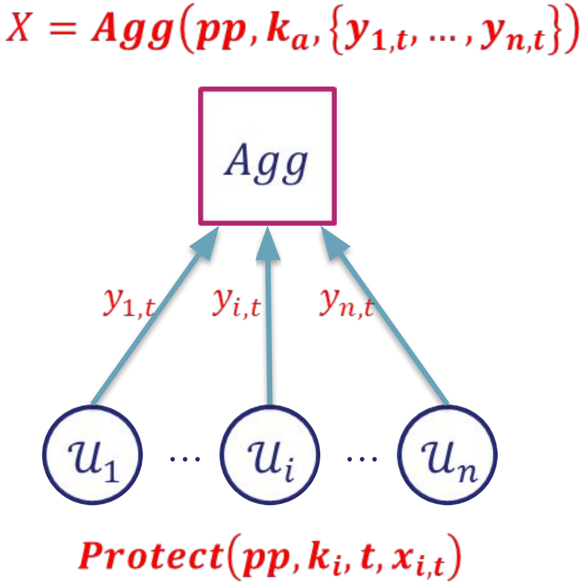


AHE-Based Secure Aggregation

Joye-Libert Scheme

- ***Setup*(λ)**: generate modulus N, hash H and key k_i for each user s.t. $\sum_i k_i = -k_a$
- ***Protect*$(pp, k_i, t, x_{i,t})$**: $y_{i,t} = (1 + x_{i,t}N)H(t)^{k_i} \bmod N^2$
- ***Agg*$\left(pp, k_a, \{y_{i,t}\}_{\forall i}\right)$**: $X = \frac{H(t)^{k_a}\prod_i y_{i,t} - 1}{N}$



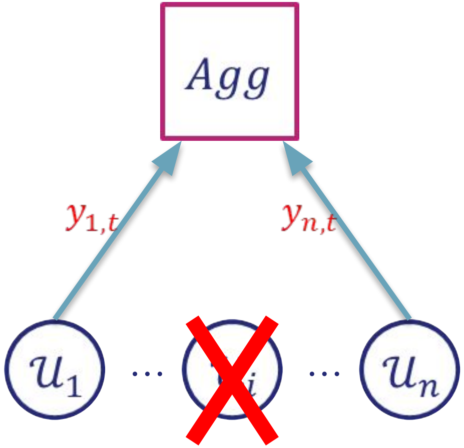


Threshold Joye-Libert Secure Aggregation

Problem of clients dropout

JL scheme is not fault-tolerant

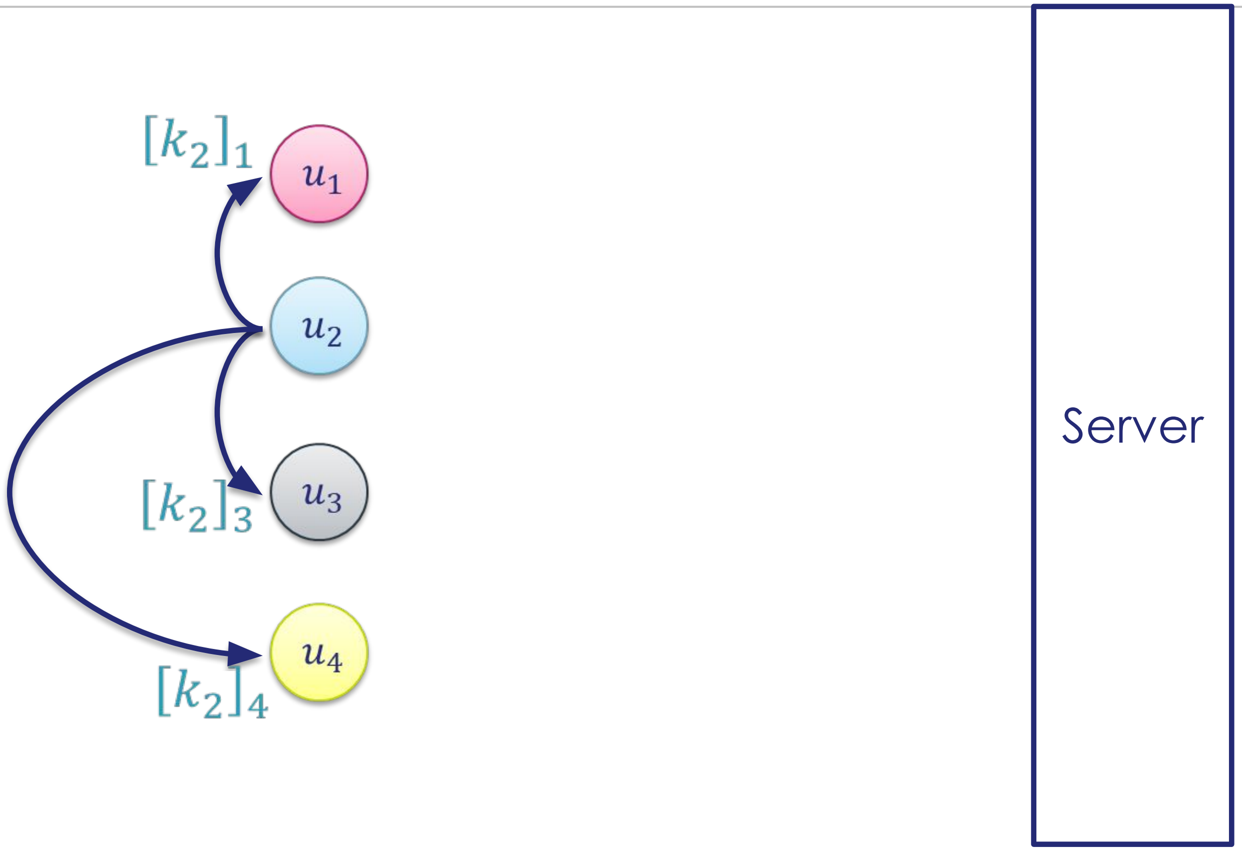
- If one or more clients dropped: $\sum_i k_i \neq -k_a$
- Agg cannot be computed



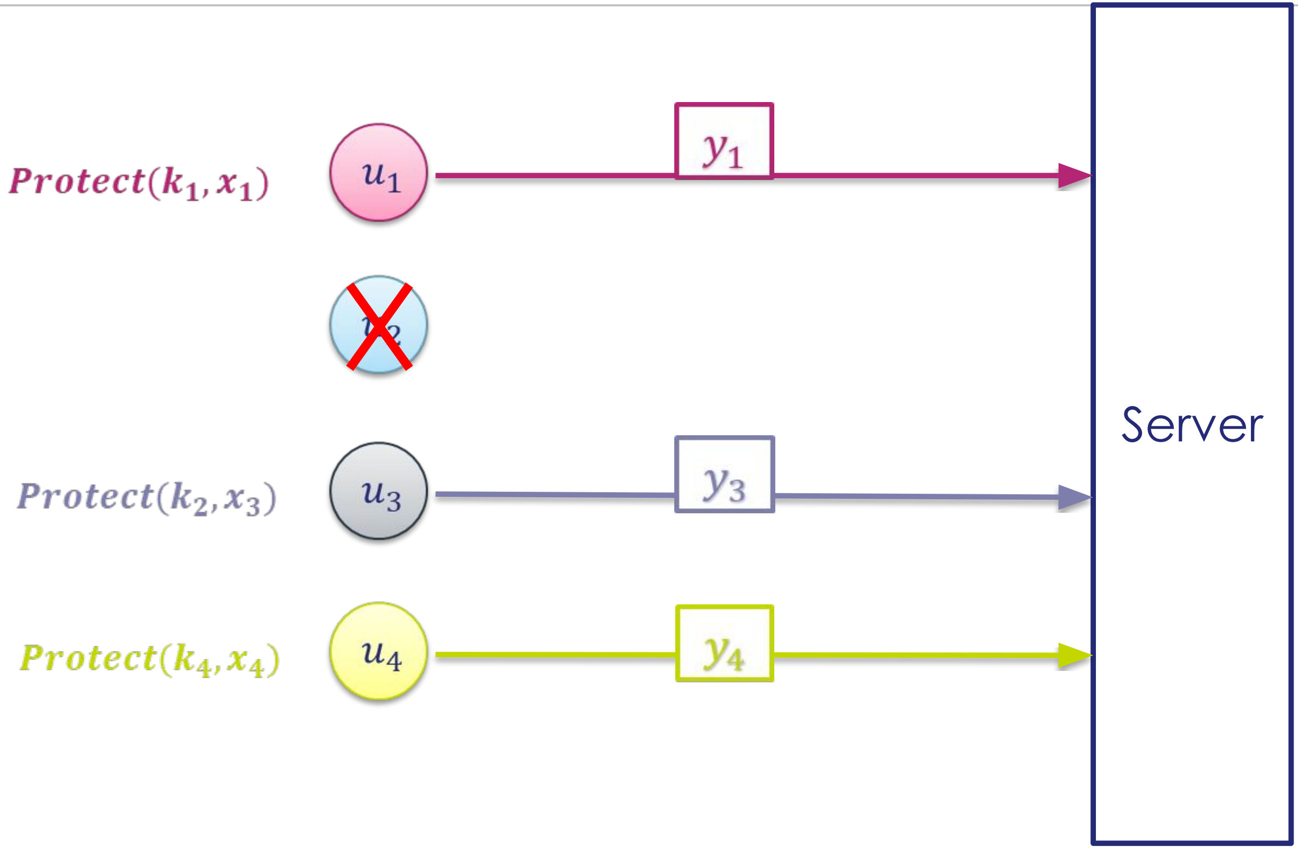
Threshold Joye-Libert scheme (TJL):

- Using Shamir's secret sharing,
  - Each client shares its protection key k_i
- When client $\mathcal{U}_i$ drops:
  - Any t-out-of-n clients encrypt on behalf $\mathcal{U}_i$ using the shares of k_i
- The server aggregate the protected inputs
- Minimum number of honest users t $> \frac{2n}{3}$ honest users

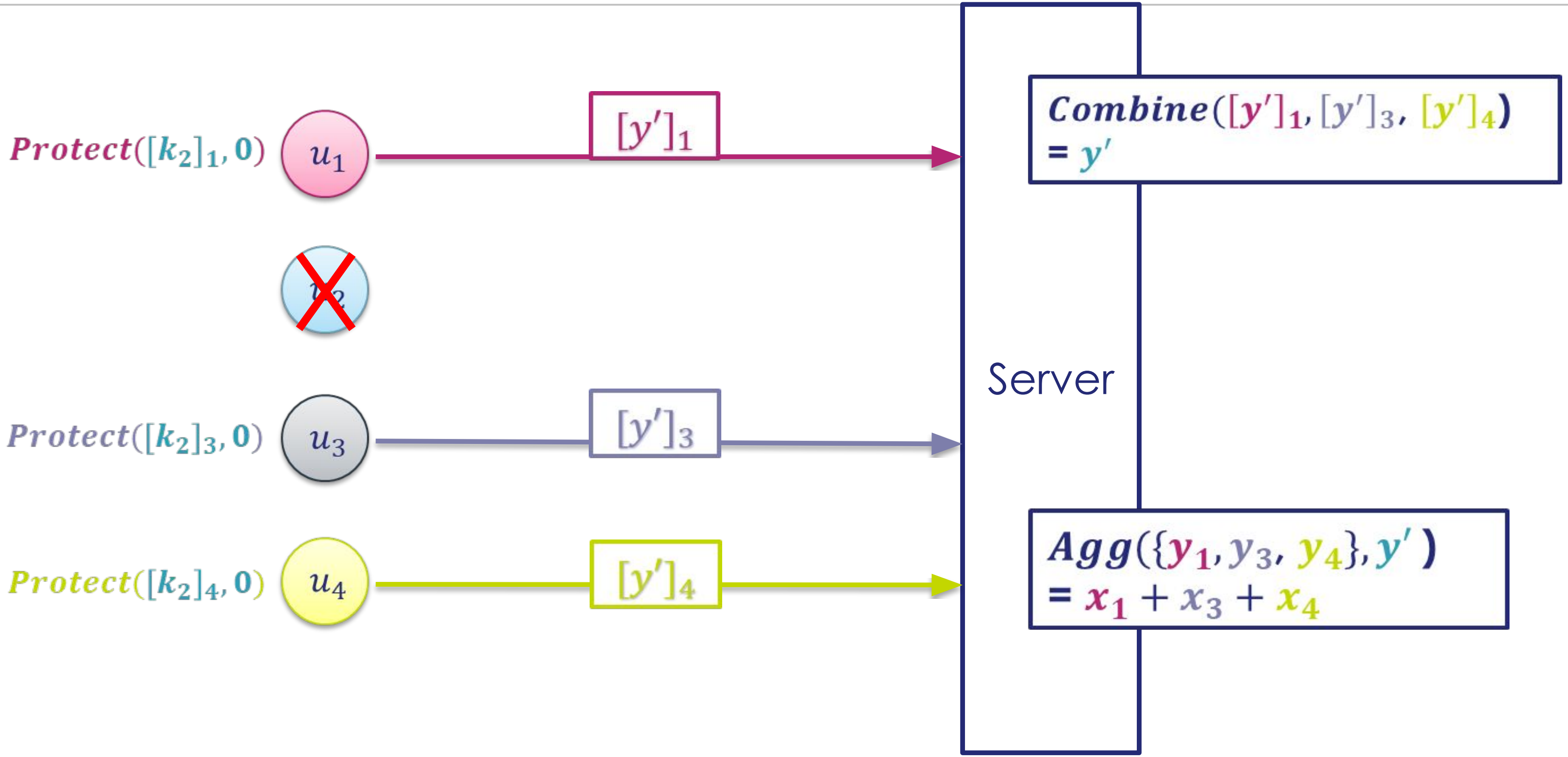
Fault Tolerant Secure Aggregation- Setup



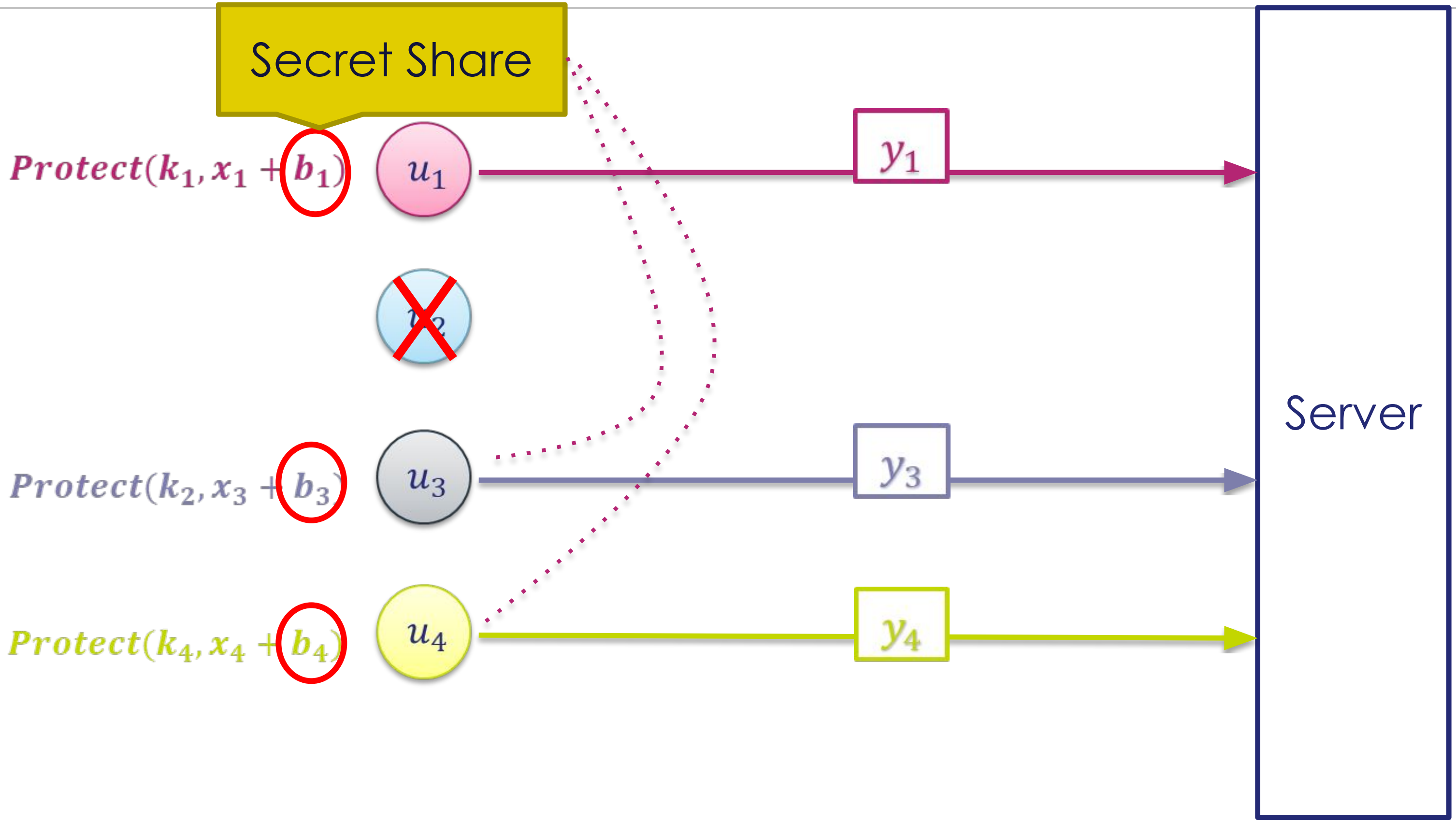
Fault Tolerant Secure Aggregation- Online - Encrypt



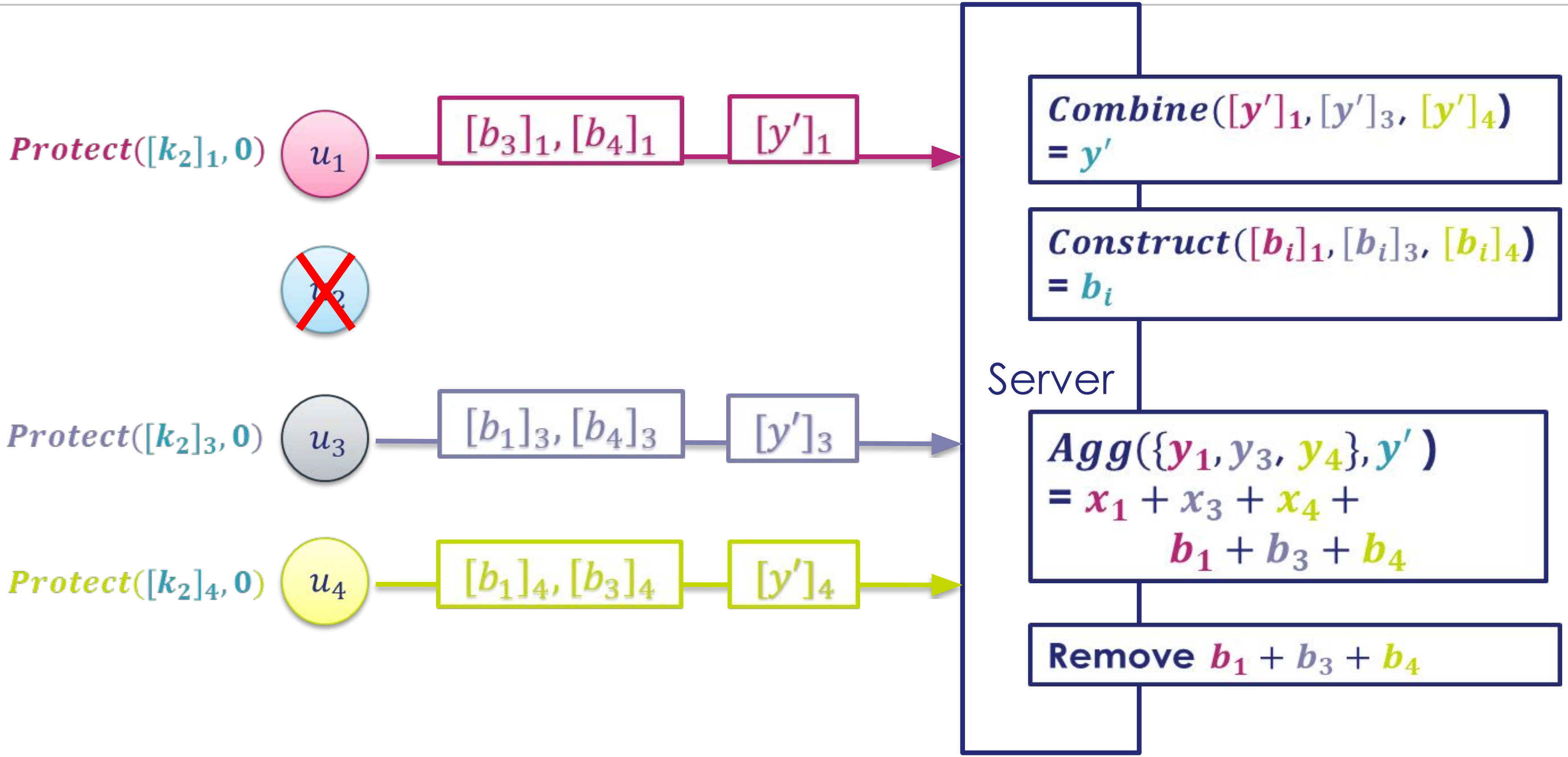
Fault Tolerant Secure Aggregation- Online - Construct



Fault Tolerant Secure Aggregation- Online - Encrypt



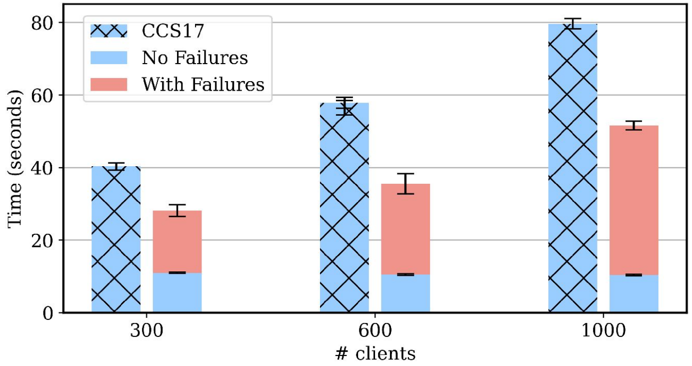
Fault Tolerant Secure Aggregation- Online - Construct

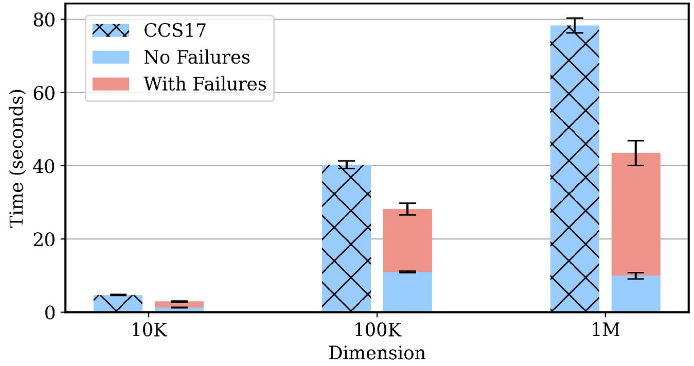


Our Solution vs [2] : Client computation

CCS17 [2]
Our Solution

- **Better scalability of the client's computation**
  - Our solution is ×2.7 faster with 100 clients and ×4.6 faster with 600
- **Same scalability w.r.t input dimension (m)**





Conclusion and Future Work

Conclusion

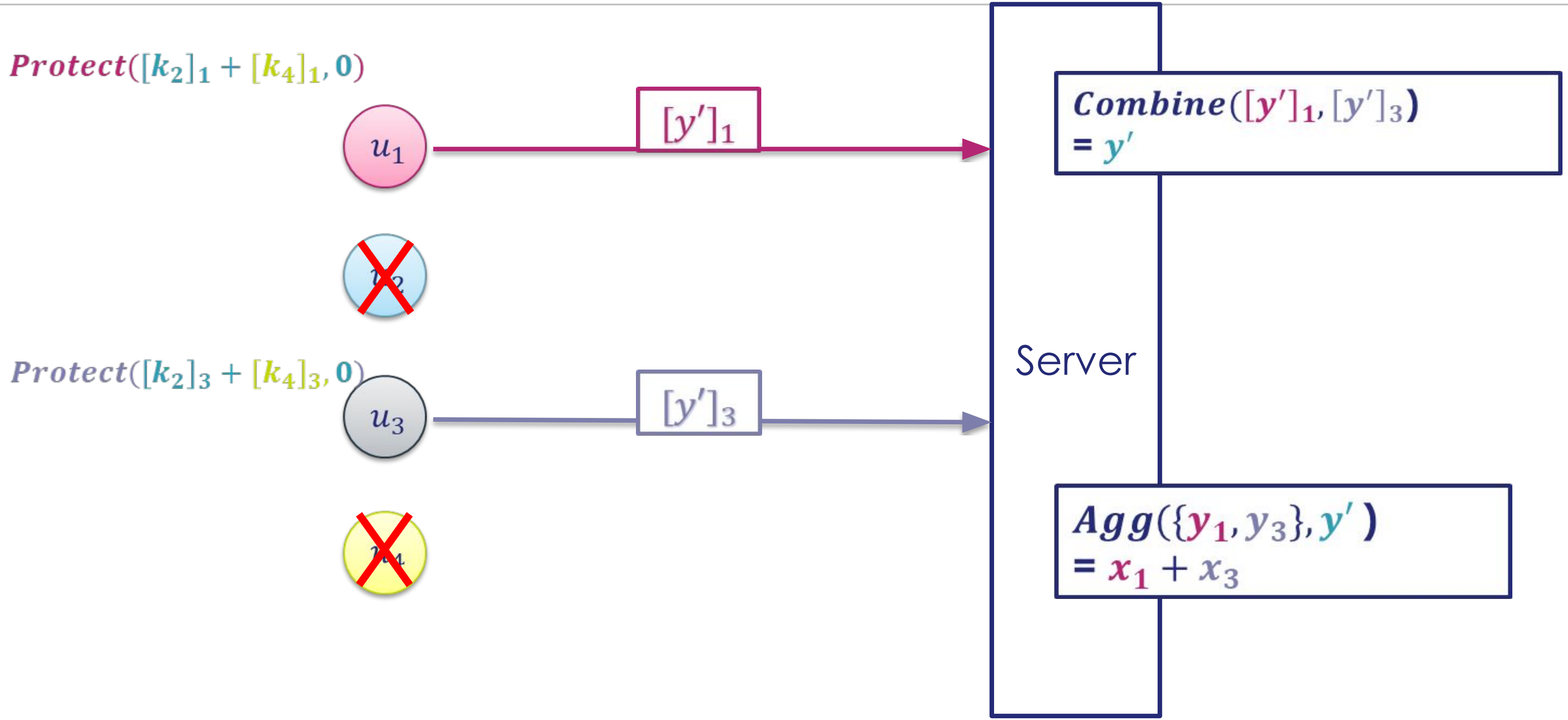
- We presented TJL scheme
- We presented FTSA (first Fault-Tolerant SA based on AHE)
- We compared to the SoTA (better scalability for the clients)

Future Work

- Optimizing the scalability
- Achieving Aggregate Unforgeability

Thanks!

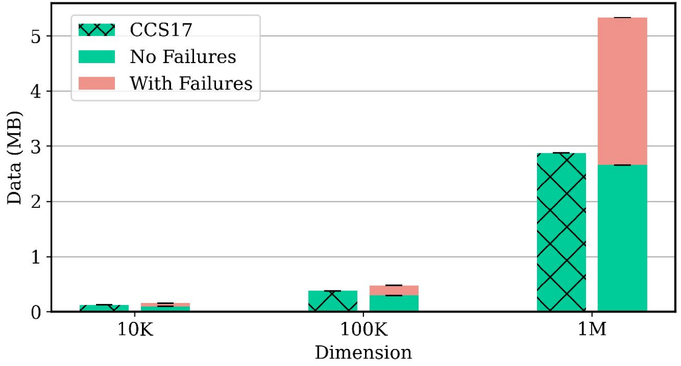
Fault Tolerant Secure Aggregation- Online - Construct



Our Solution vs [2] : Client comunication

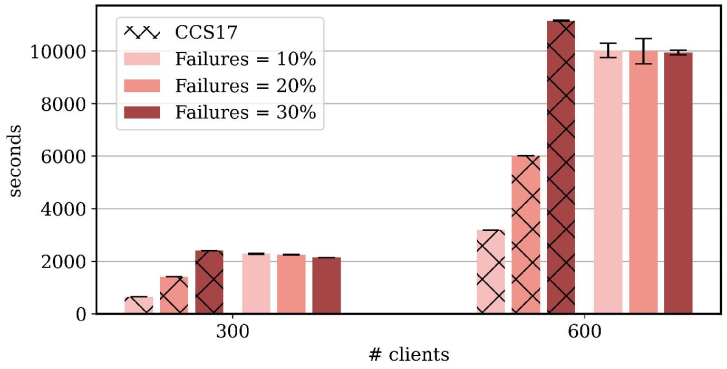
CCS17 [2]
Our Solution

- Better scalability w.r.t # clients (n)
- Worst scalability w.r.t input dimension (m)

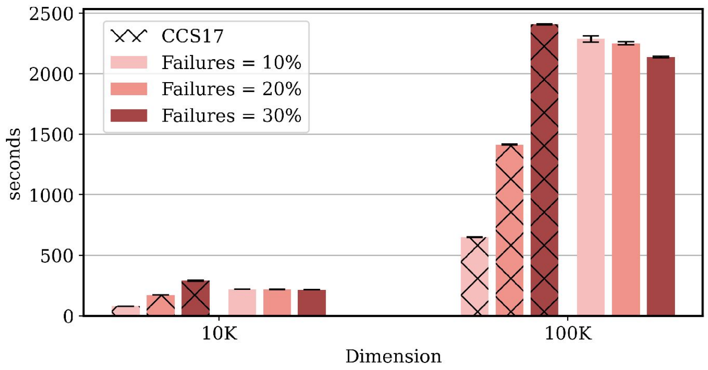


Our Solution vs [2]: Server computation

CCS17 [2]
Our Solution



- **Better scalability w.r.t the number of dropped clients (d)**
  - Better performance on high 30% dropouts



Fault Tolerant Secure Aggregation using AHE

Threshold Joye-Libert scheme (TJL)

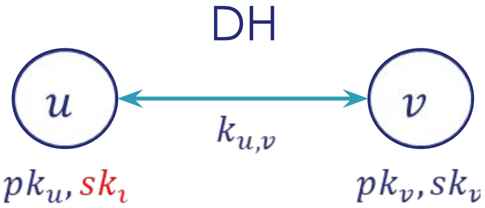
- $(k_a, \{k_i\}_{\forall i \in [n]}, pp) \leftarrow Setup(\lambda)$: generates keys an public parameters
- $\{j, [k_i]_j\} \leftarrow SKShare(k_i, t, n)$: share the secret key of user $\mathcal{U}_1$
- $[y'_t]_i \leftarrow ShareProtect\left(pp, \{[k_j]_i\}_{\forall u \in U''}, t\right)$: protect a zero-value using the shares of dropped clients
- $y'_t \leftarrow ShareCombine(pp, \{[y'_t]_i\}_{\forall i \in U''}, t)$: combine the protected zero-values shares of dropped clients
- $y_{i,t} \leftarrow Protect\left(pp, k_i, t, x_{i,t}\right)$: protect an input $x_{i,t}$ of online client
- $\sum_{\forall i} x_{i,t} \leftarrow Agg\left(pp, k_a, \{y_{i,t}\}_{\forall i \in U'}, y'_\tau\right)$: aggregate all protected input of online clients and zero-values of dropped ones

Fault Tolerant Secure Aggregation

State of the Art: Fault Tolerant Masking [2]

- Use DH key agreement between two clients to agree on masking keys k_u

$$k_u = \sum_{u<v} \boldsymbol{PRG}(k_{u,v}) - \sum_{u>v} \boldsymbol{PRG}(k_{u,v})$$



- Notice that: $\sum_{u} k_u = 0$
- $Protect: y_u = x_u + k_u + \boldsymbol{PRG}(b_u) \bmod R$
- b_u is a random generated seed
- To make the scheme Fault-Tolerant, use Shamir's secret sharing to share sk_u and b_u of each user
- If user u is online: The server collects shares of b_u and reconstruct it
- If user u dropped: The server collects shares of sk_u and reconstruct it
  It then re-computes all DH agreements and recover the mask k_u