LightSecAgg: a Lightweight and Versatile Design for Secure Aggregation in Federated Learning

Jinhyun So, Ramy E. Ali, Chaoyang He, Salman Avestimehr

University of Southern California

Joint work with

Chien-Sheng Yang (USC), Songze Li (HKUST), Qian Yu (USC), and Basak Guler (UCR)

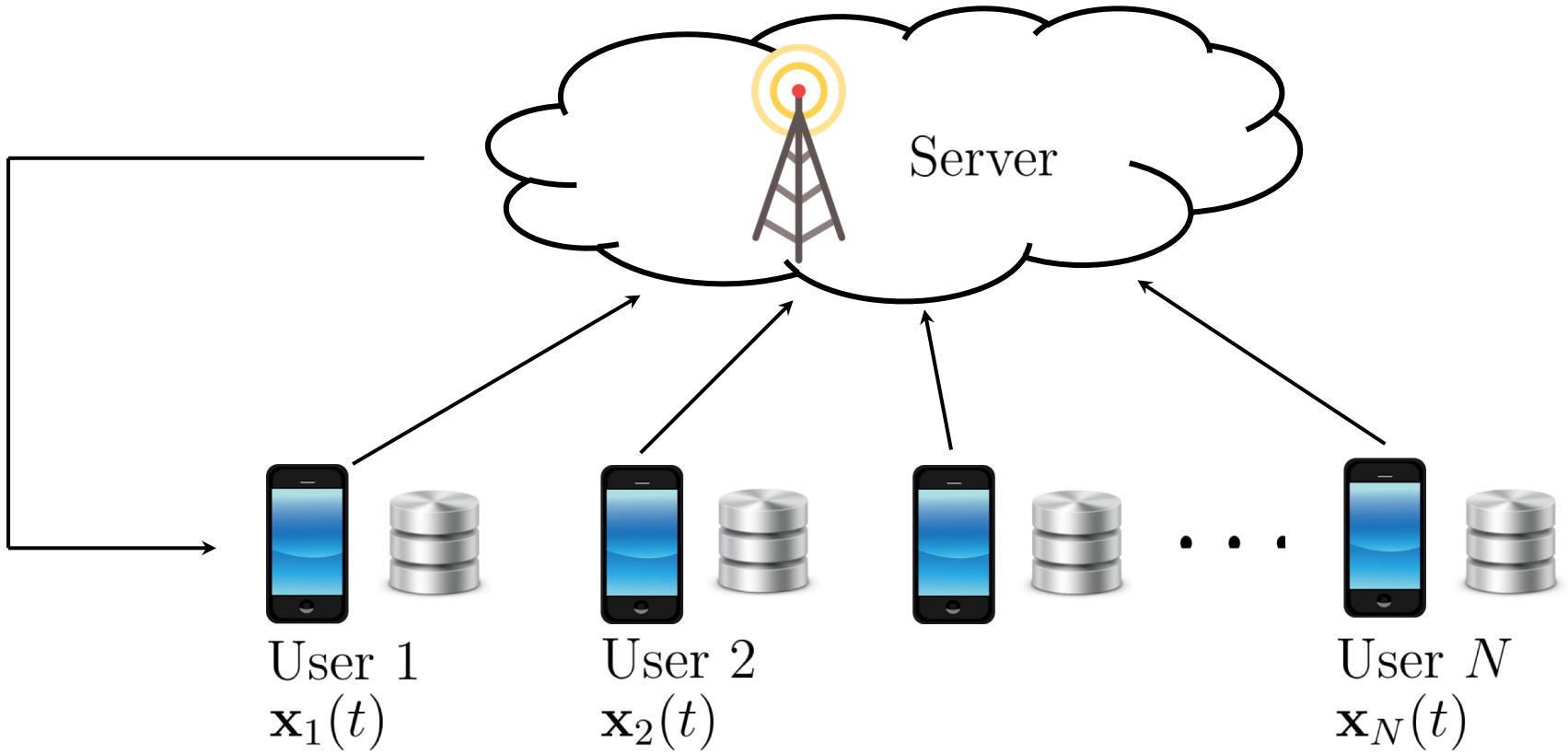
Executive Summary

- Part 1. General description of LightSecAgg
- Part 2. LightSecAgg for Asynchronous Federated Learning
- Part 3. System Design and Experiments

Part 1. General description of LightSecAgg

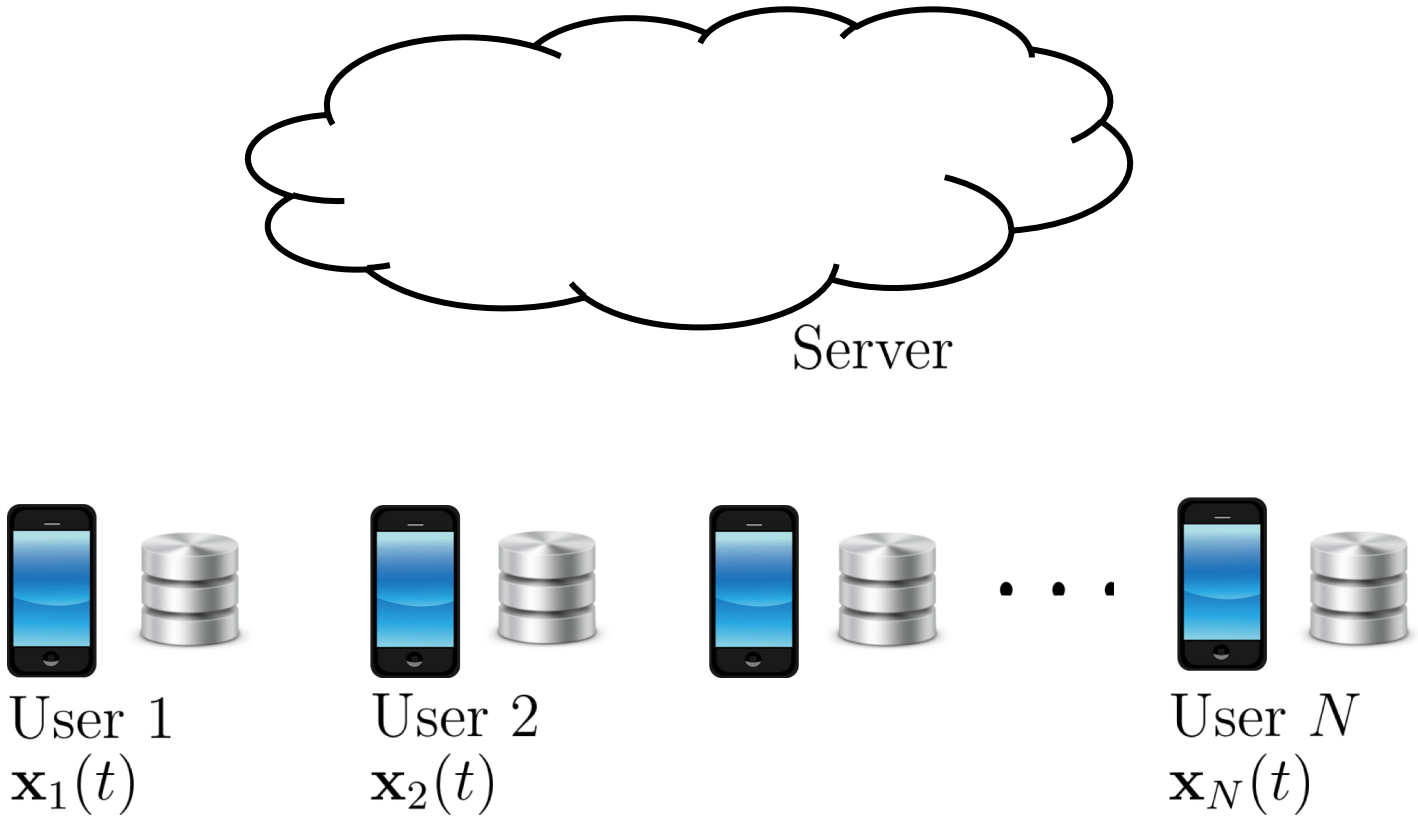
Federated Learning

Machine learning on massive amount of data collected on many users/mobile devices



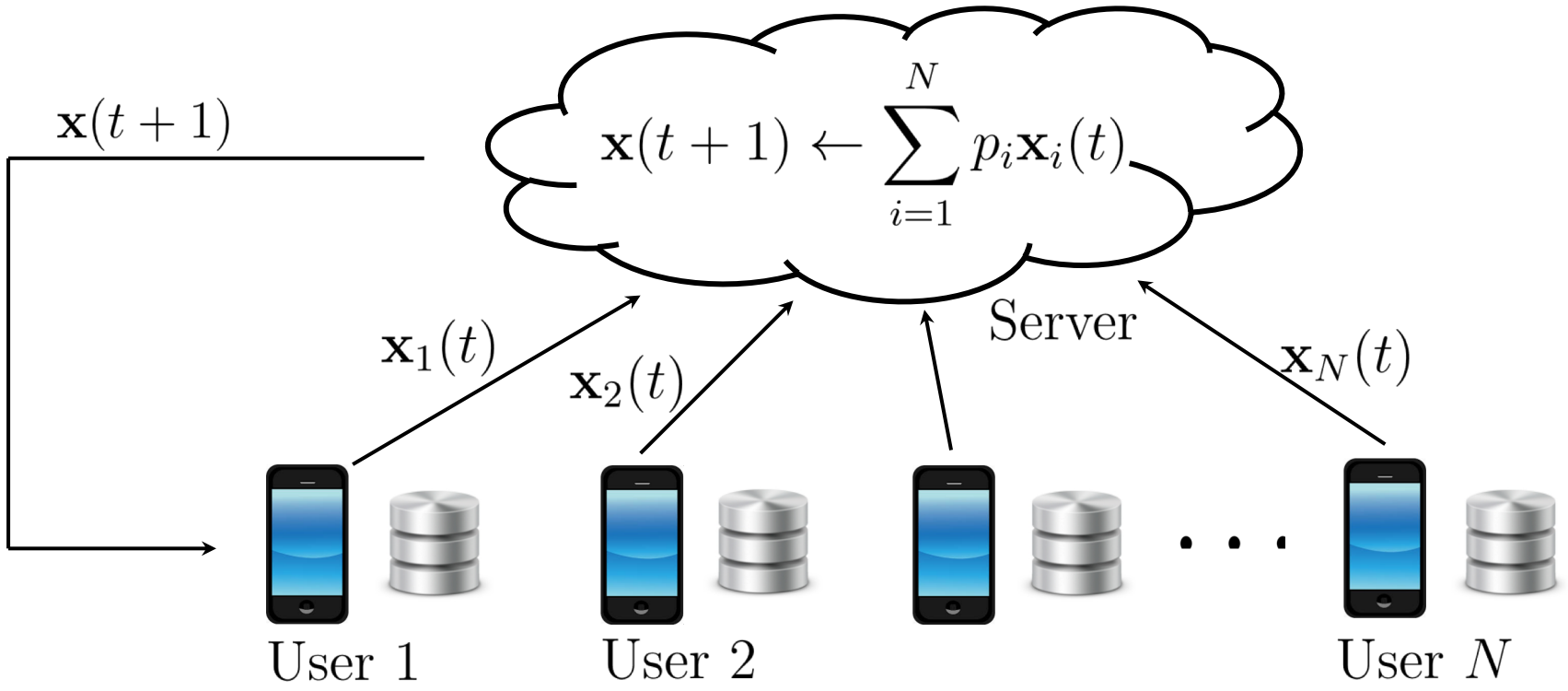
Federated Learning

Step 1: Train locally at each user on the available data



Federated Learning

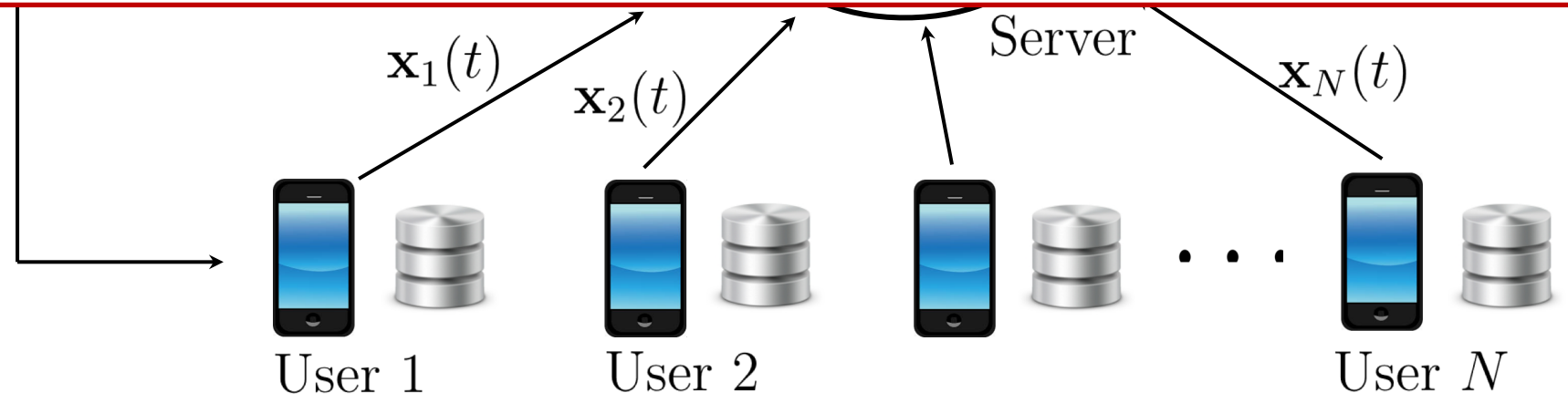
Step 2: Aggregate the local models at the server, update the mode, and send the new global model back to the users



Federated Learning

Step 2: Aggregate the local models at the server, update the mode, and send the new global model back to the users

- Key Design Principles
  1. Privacy
  2. Communication Efficiency



Model Inversion Attack

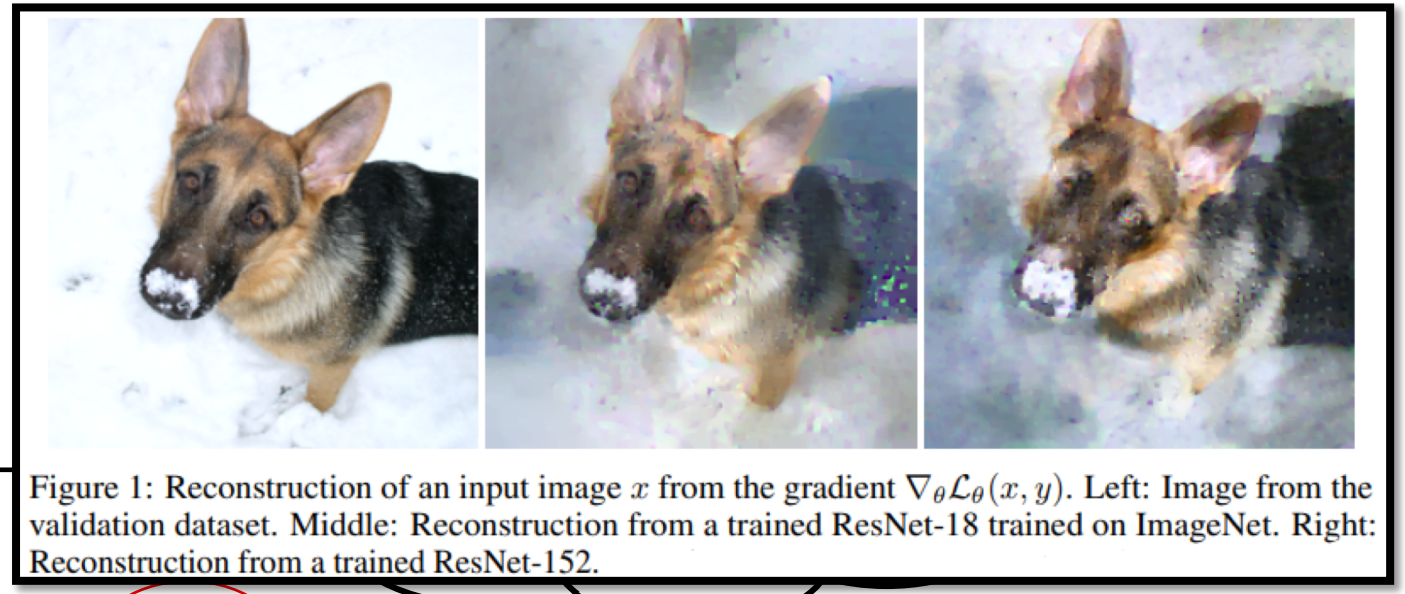
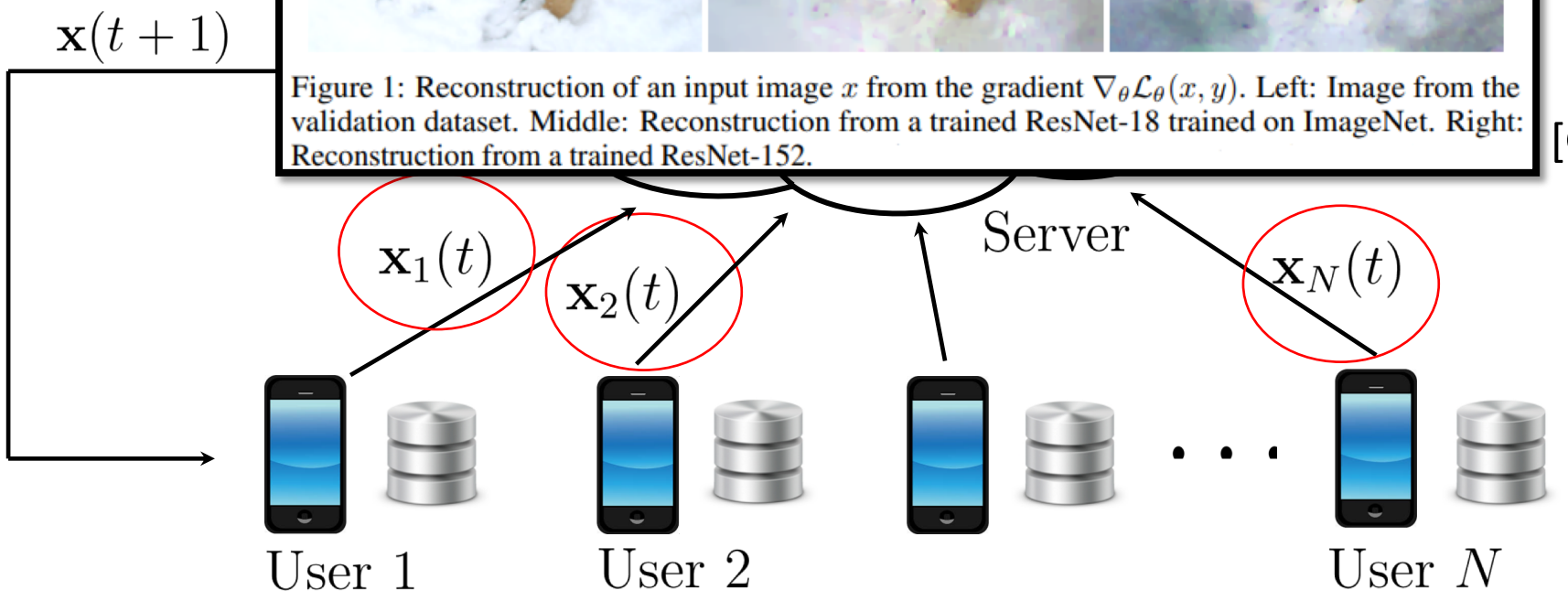


Figure 1: Reconstruction of an input image x from the gradient $\nabla_\theta \mathcal{L}_\theta(x, y)$. Left: Image from the validation dataset. Middle: Reconstruction from a trained ResNet-18 trained on ImageNet. Right: Reconstruction from a trained ResNet-152.

[Geiping et al]

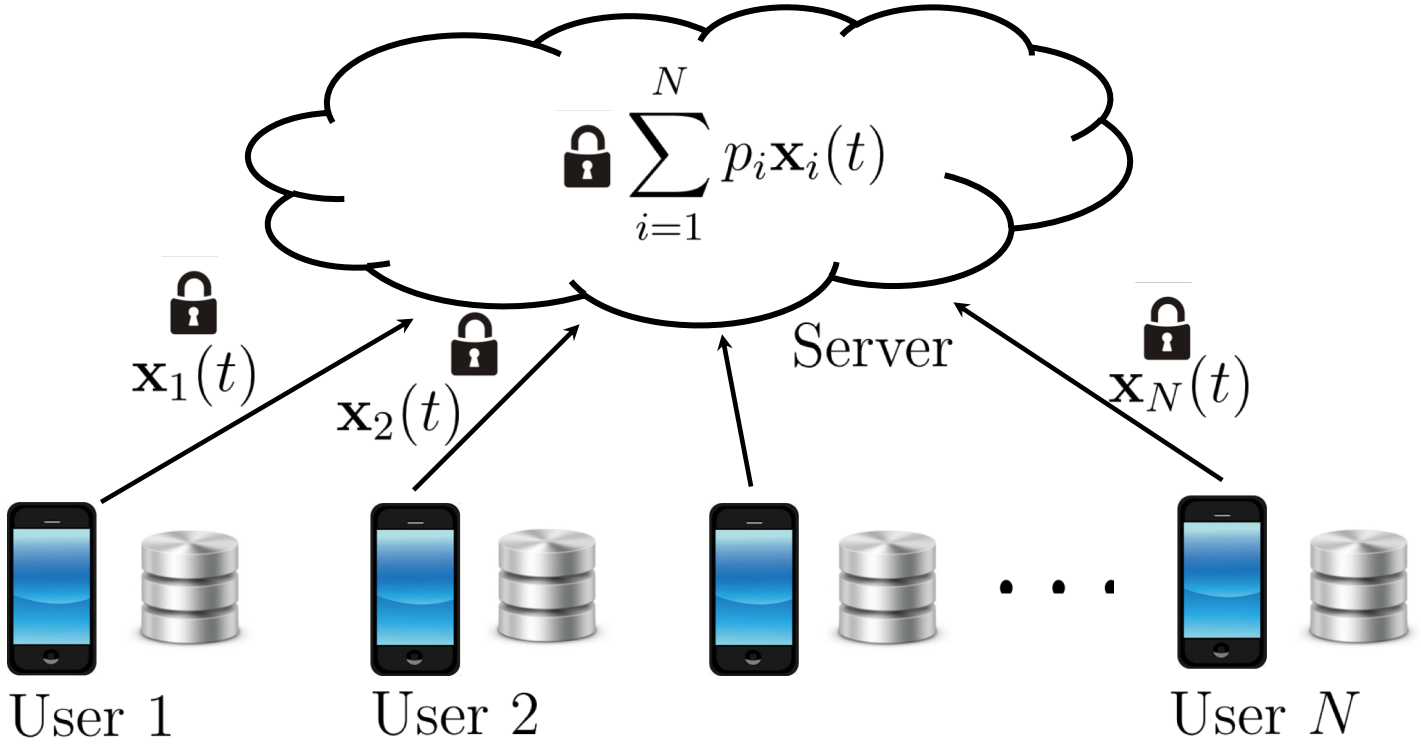


Problem: Individual model update can leak sensitive data.

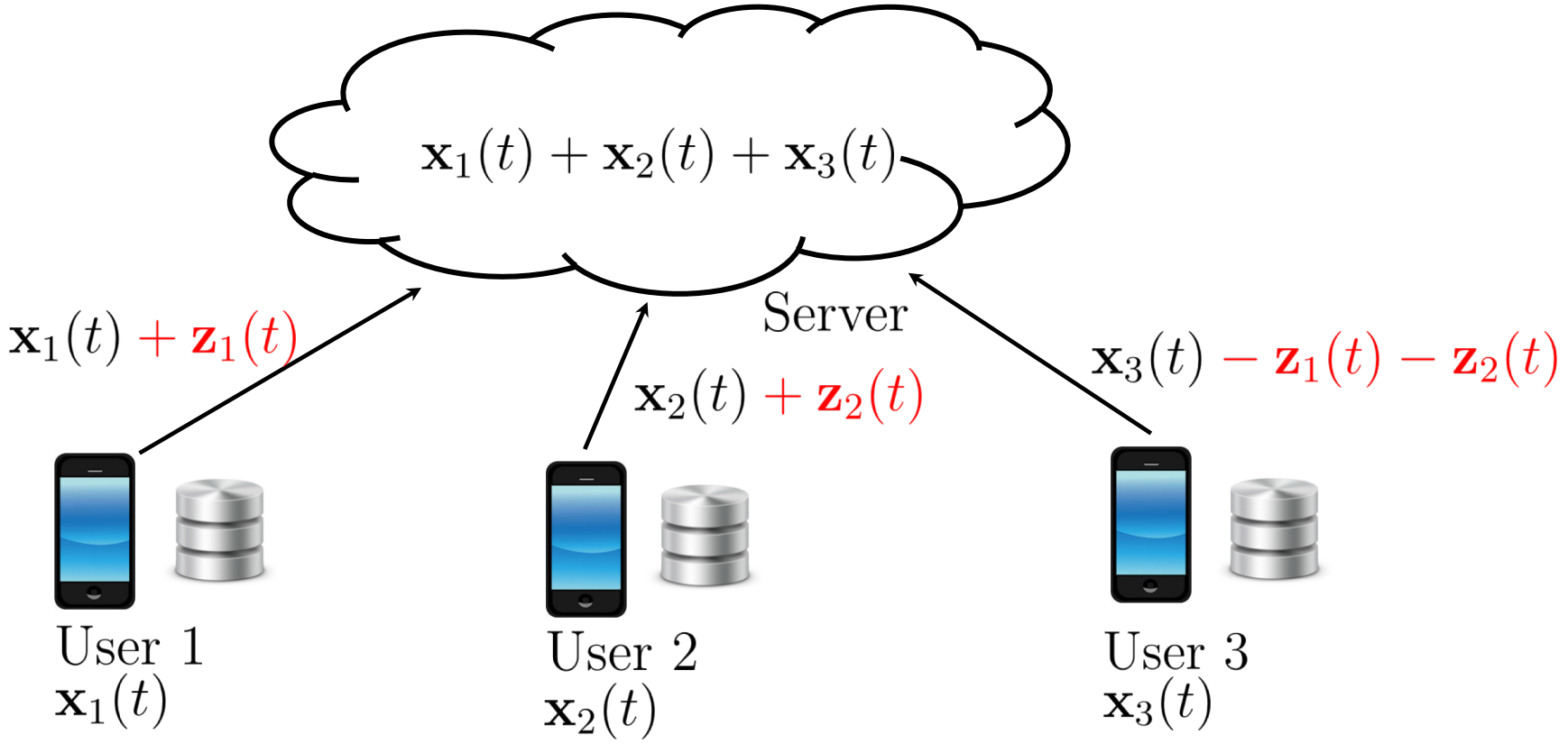
Key component: Secure Aggregation

Threat model:

- honest, but curious
- up to a fraction of users and/or server colluding

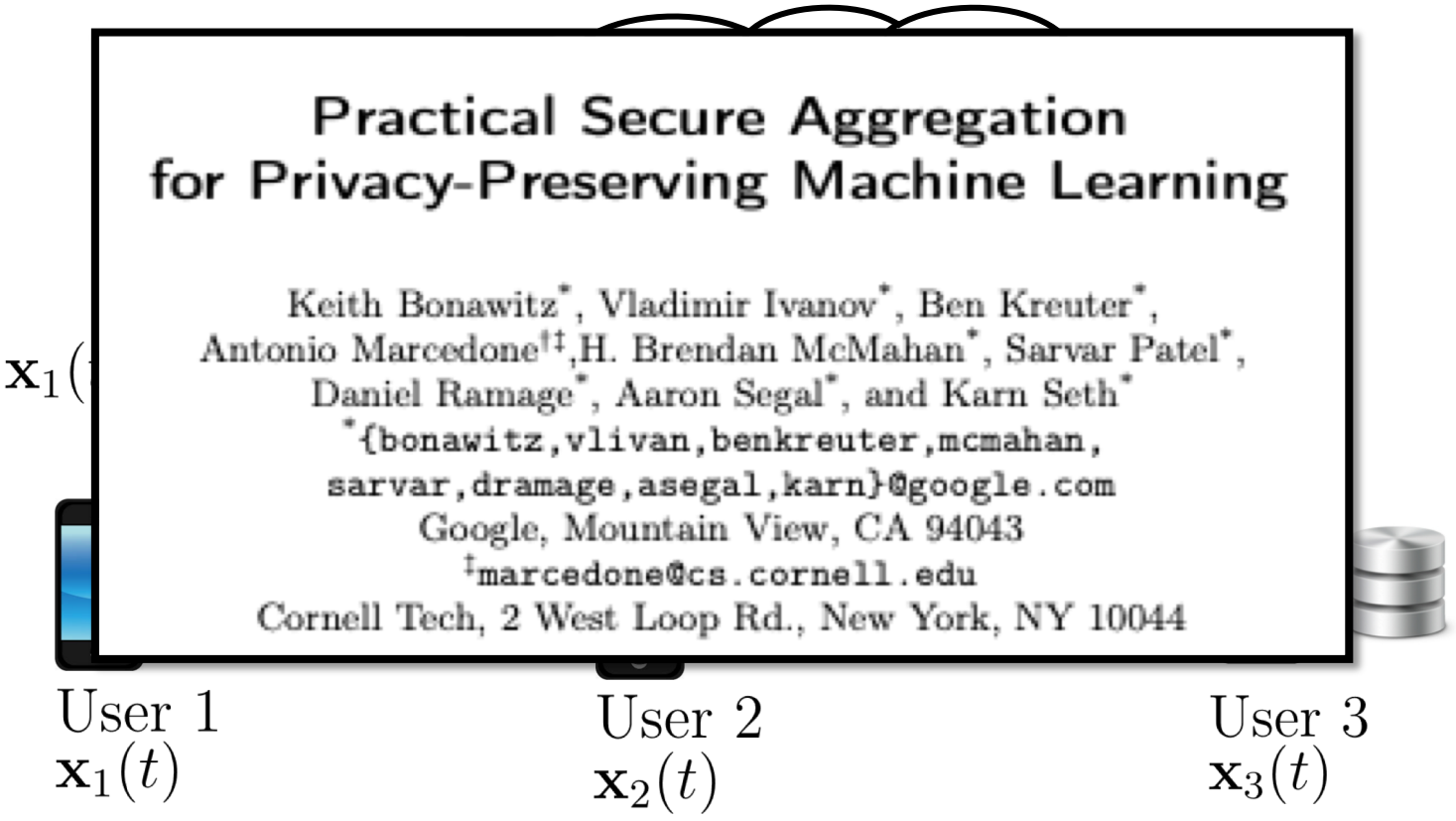


Key component: Secure Aggregation



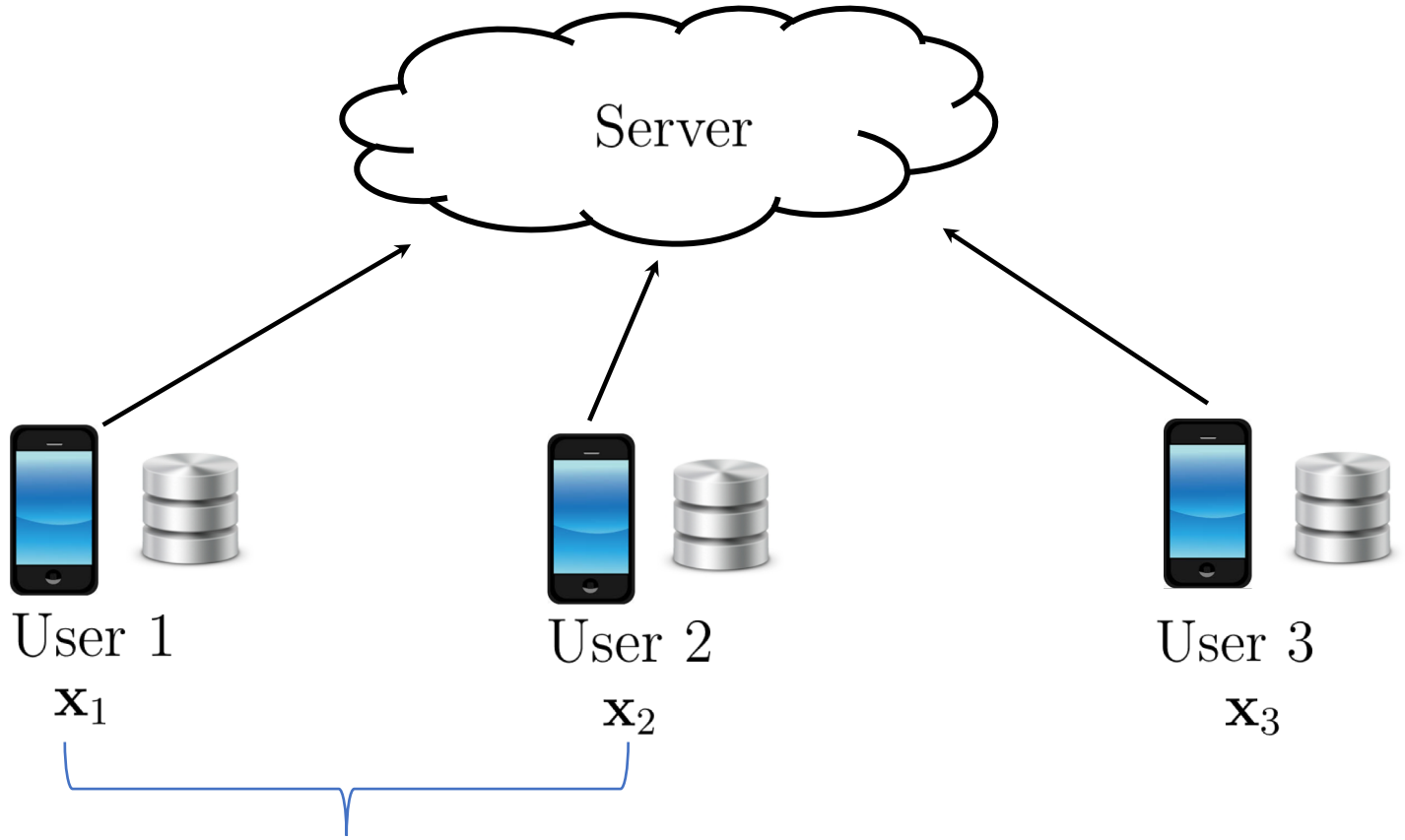
Perfectly secure: the server only learns the aggregated model.

Key component: Secure Aggregation



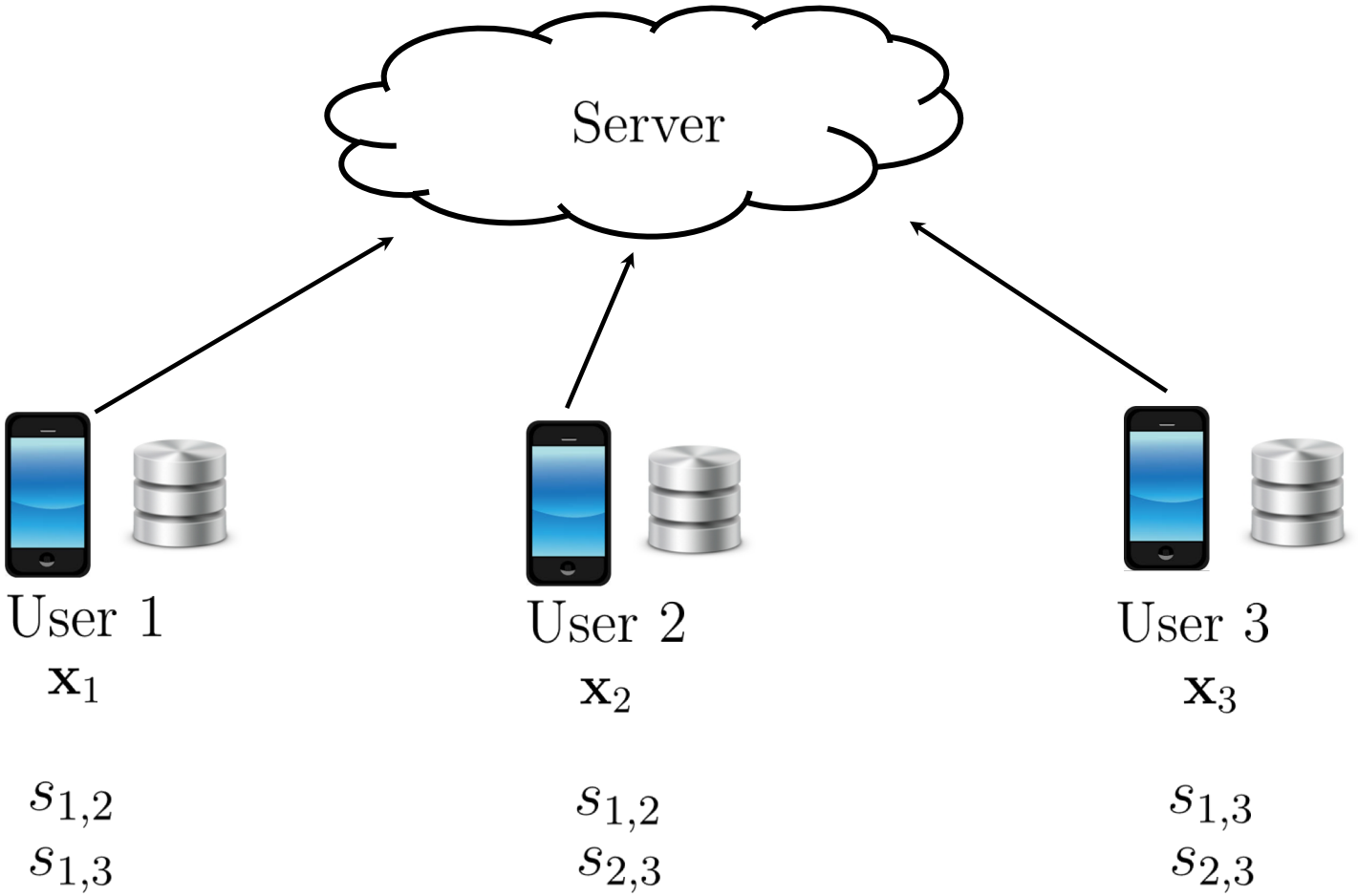
How to create an additive structure allowing global aggregation amidst possible user dropouts?

Example (SecAgg)



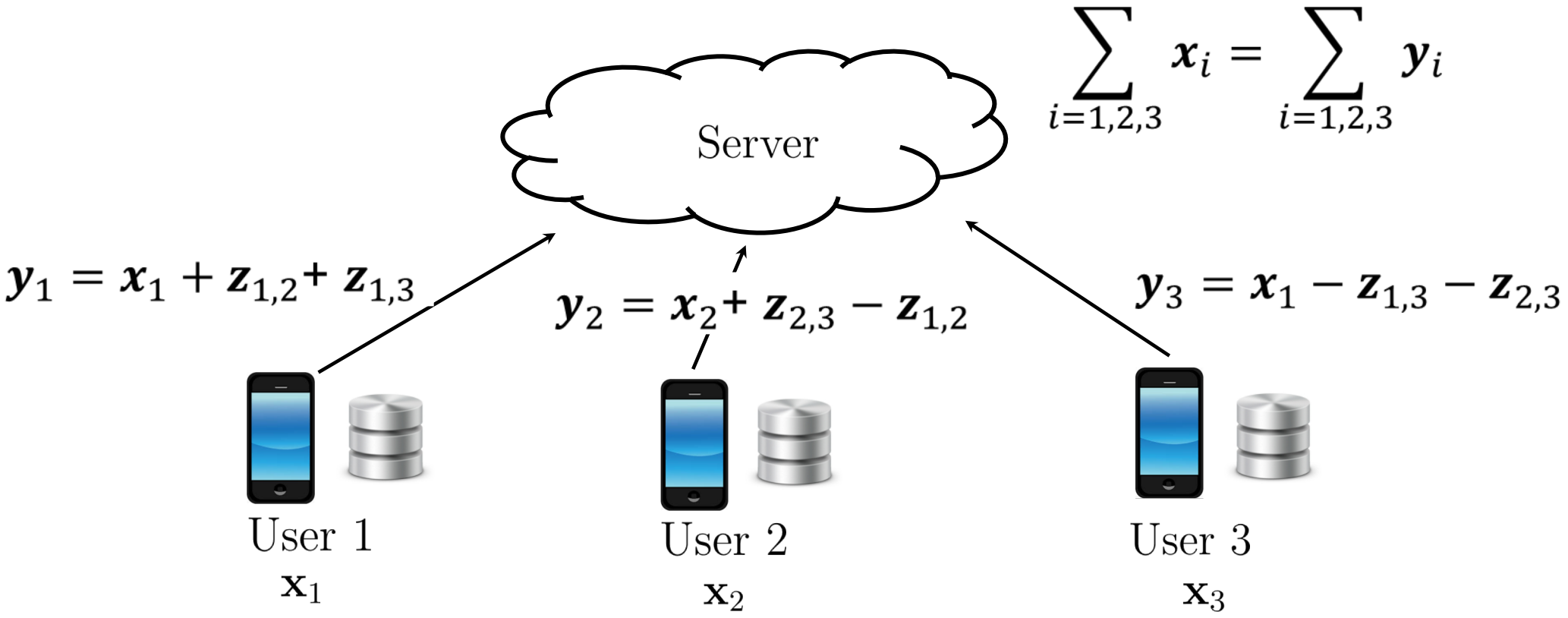
agree on pairwise random seed $s_{1,2}$

Example (SecAgg)



Pairwise random seed

Example (SecAgg)



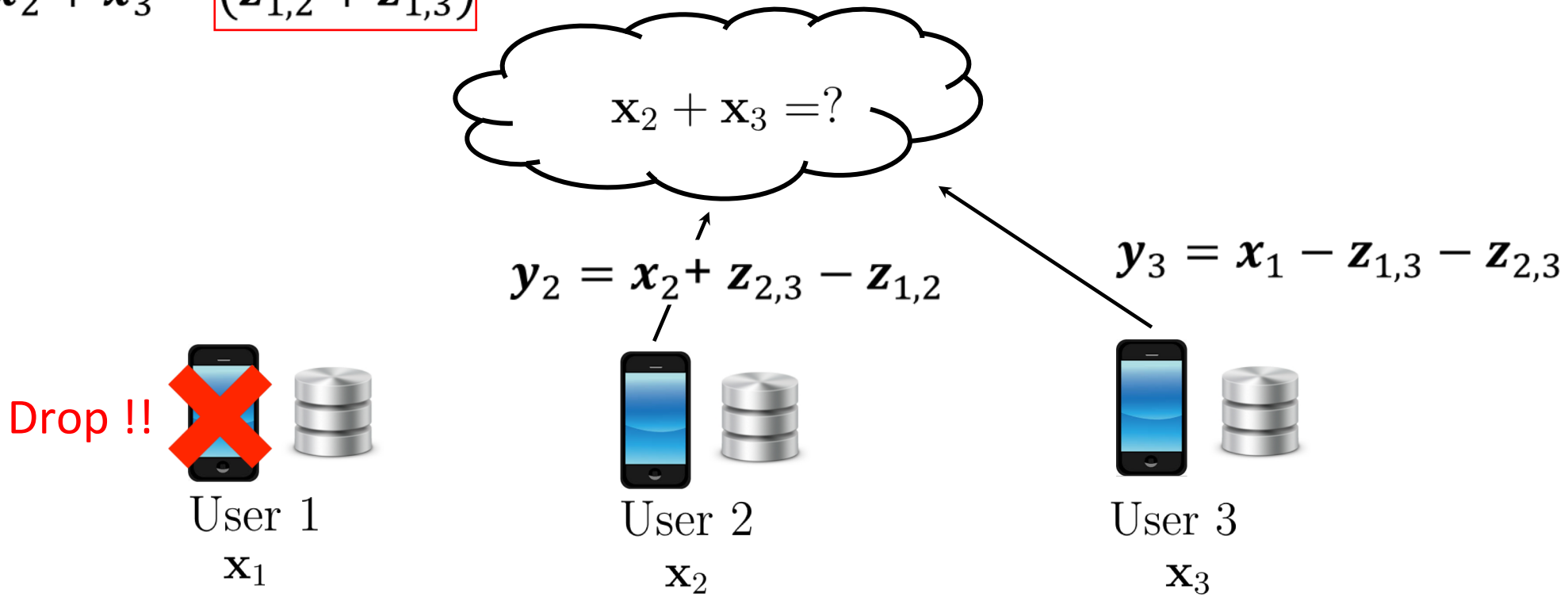
$\mathbf{z}_{1,2} = \mathrm{PRG}(s_{1,2})$ $\quad$ $\mathbf{z}_{1,2} = \mathrm{PRG}(s_{1,2})$ $\quad$ $\mathbf{z}_{1,3} = \mathrm{PRG}(s_{1,3})$

$\mathbf{z}_{1,3} = \mathrm{PRG}(s_{1,3})$ $\quad$ $\mathbf{z}_{2,3} = \mathrm{PRG}(s_{2,3})$ $\quad$ $\mathbf{z}_{2,3} = \mathrm{PRG}(s_{2,3})$

Example (SecAgg)

Remaining random vectors

$$\boldsymbol{y}_2 + \boldsymbol{y}_3 = \boldsymbol{x}_2 + \boldsymbol{x}_3 - \boxed{(\boldsymbol{z}_{1,2} + \boldsymbol{z}_{1,3})}$$

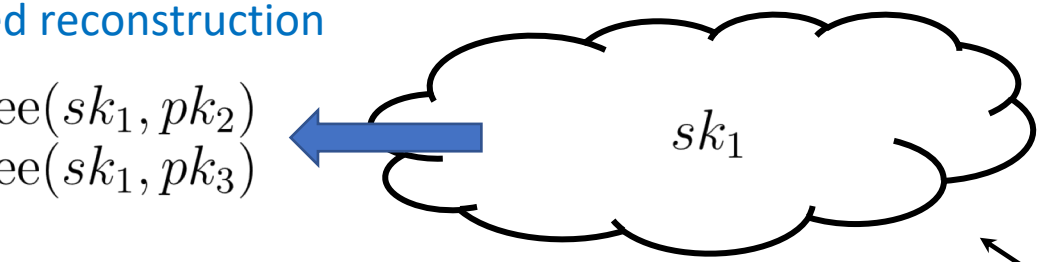


User 1	User 2	User 3
$\mathbf{z}_{1,2} = \mathrm{PRG}(s_{1,2})$	$\mathbf{z}_{1,2} = \mathrm{PRG}(s_{1,2})$	$\mathbf{z}_{1,3} = \mathrm{PRG}(s_{1,3})$
$\mathbf{z}_{1,3} = \mathrm{PRG}(s_{1,3})$	$\mathbf{z}_{2,3} = \mathrm{PRG}(s_{2,3})$	$\mathbf{z}_{2,3} = \mathrm{PRG}(s_{2,3})$

Example (SecAgg)

Pairwise random seed reconstruction

$$s_{1,2} = \text{Key.Agree}(sk_1, pk_2)$$
$$s_{1,3} = \text{Key.Agree}(sk_1, pk_3)$$



$$\boldsymbol{x_2 + x_3 = y_2 + y_3 + (z_{1,2} + z_{1,3})}$$

Secret shares of sk_1

User 1

$\mathbf{x}_1$

$$\mathbf{z}_{1,2} = \text{PRG}(s_{1,2})$$
$$\mathbf{z}_{1,3} = \text{PRG}(s_{1,3})$$

User 2

$\mathbf{x}_2$

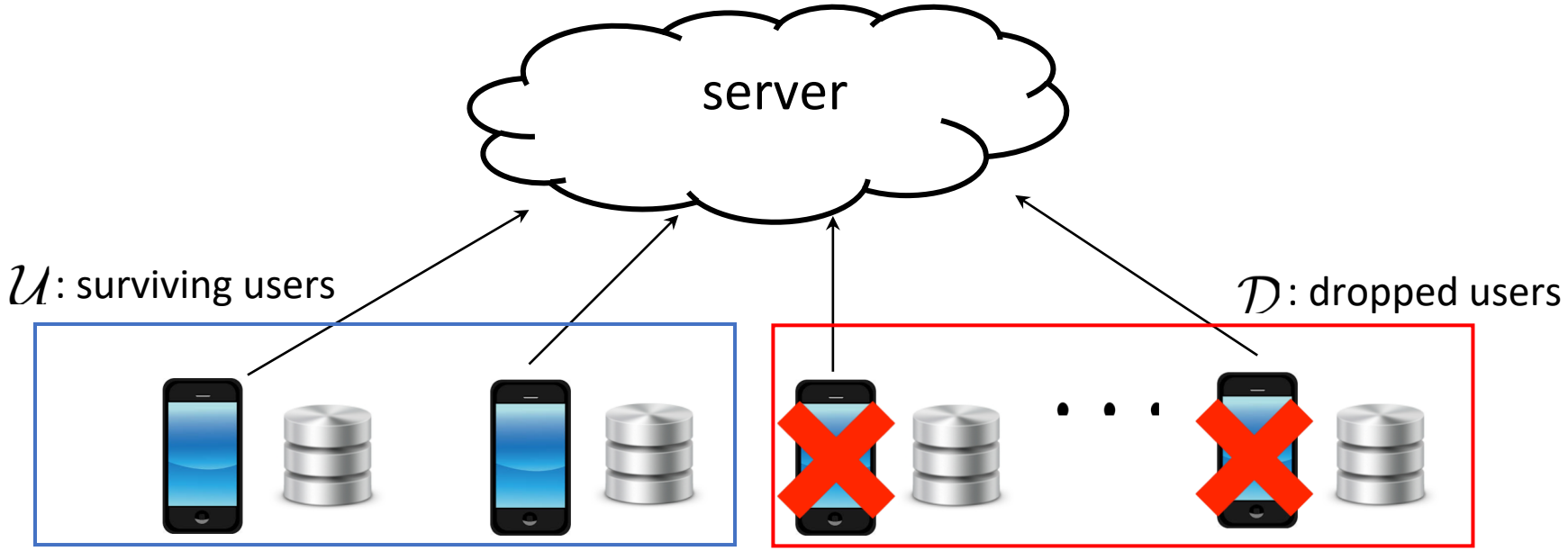
$$\mathbf{z}_{1,2} = \text{PRG}(s_{1,2})$$
$$\mathbf{z}_{2,3} = \text{PRG}(s_{2,3})$$

User 3

$\mathbf{x}_3$

$$\mathbf{z}_{1,3} = \text{PRG}(s_{1,3})$$
$$\mathbf{z}_{2,3} = \text{PRG}(s_{2,3})$$

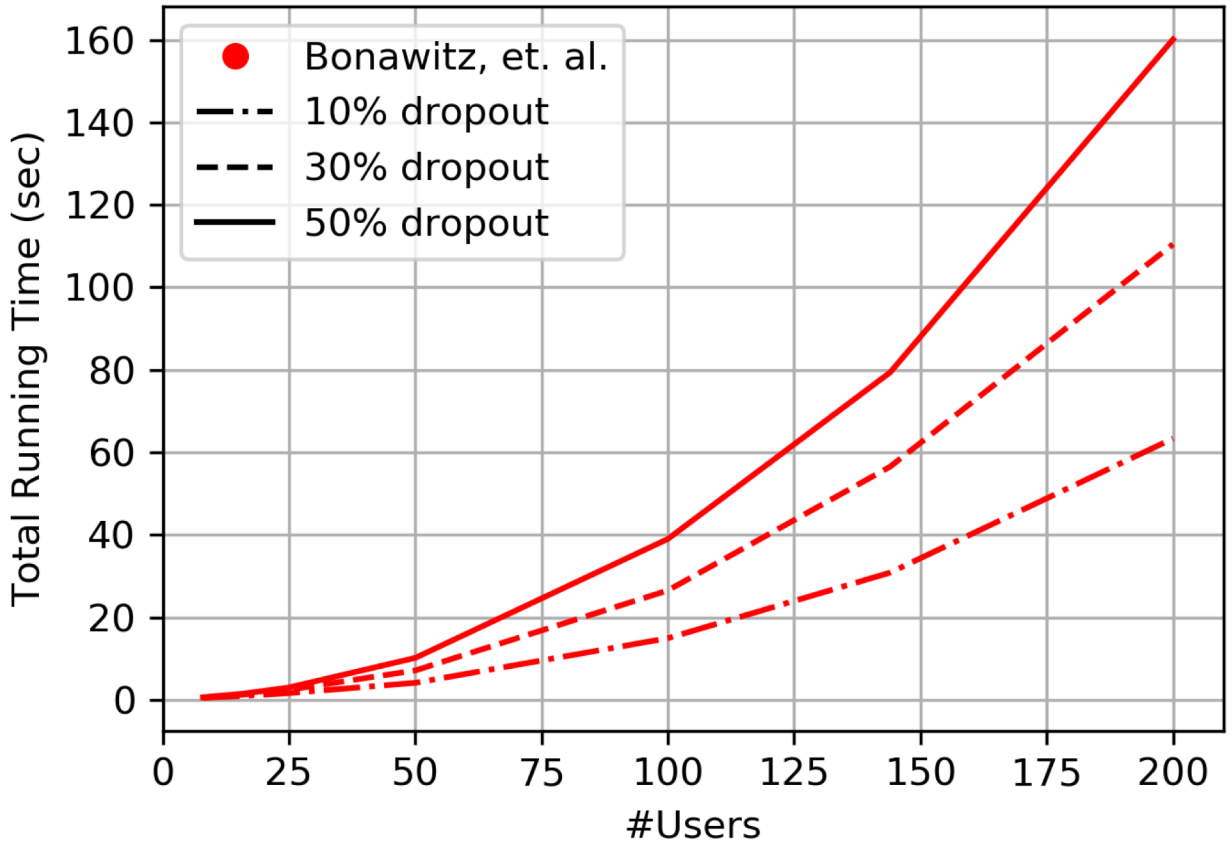
State-of-the-Art: SecAgg



$$\mathbf{z} = \sum_{u \in \mathcal{U}} (\mathbf{y}_u - \mathrm{PRG}(b_u)) + \sum_{u \in \mathcal{D}} \left(\sum_{v:u<v} \mathrm{PRG}(s_{u,v}) - \sum_{v:u>v} \mathrm{PRG}(s_{v,u}) \right)$$

The number of mask reconstructions at the server substantially grows as more users are dropped, causing a major computational bottleneck.

State-of-the-Art: SecAgg



Individual model size of 100,000 with 32 bits entries- experiments over Amazon EC2

New Perspective

State-of-the-art:

$$\mathbf{z} = \sum_{u \in \mathcal{U}} (\mathbf{y}_u - \mathrm{PRG}(b_u)) + \sum_{u \in \mathcal{D}} \left(\sum_{v:u<v} \mathrm{PRG}(s_{u,v}) - \sum_{v:u>v} \mathrm{PRG}(s_{v,u}) \right)$$

New Perspective:

$$\mathbf{z} = \sum_{u \in \mathcal{U}} \mathbf{x}_u + \sum_{u \in \mathcal{U}} \mathbf{z}_u$$

Our focus

We turn the focus from “random-seed reconstruction of the dropped users” to “one-shot aggregate-mask reconstruction of the surviving users”.

LightSecAgg: Key Components

Main components

1) Offline encoding and sharing of local masks

enables one-shot aggregate-mask reconstruction

2) Masking and uploading of local models

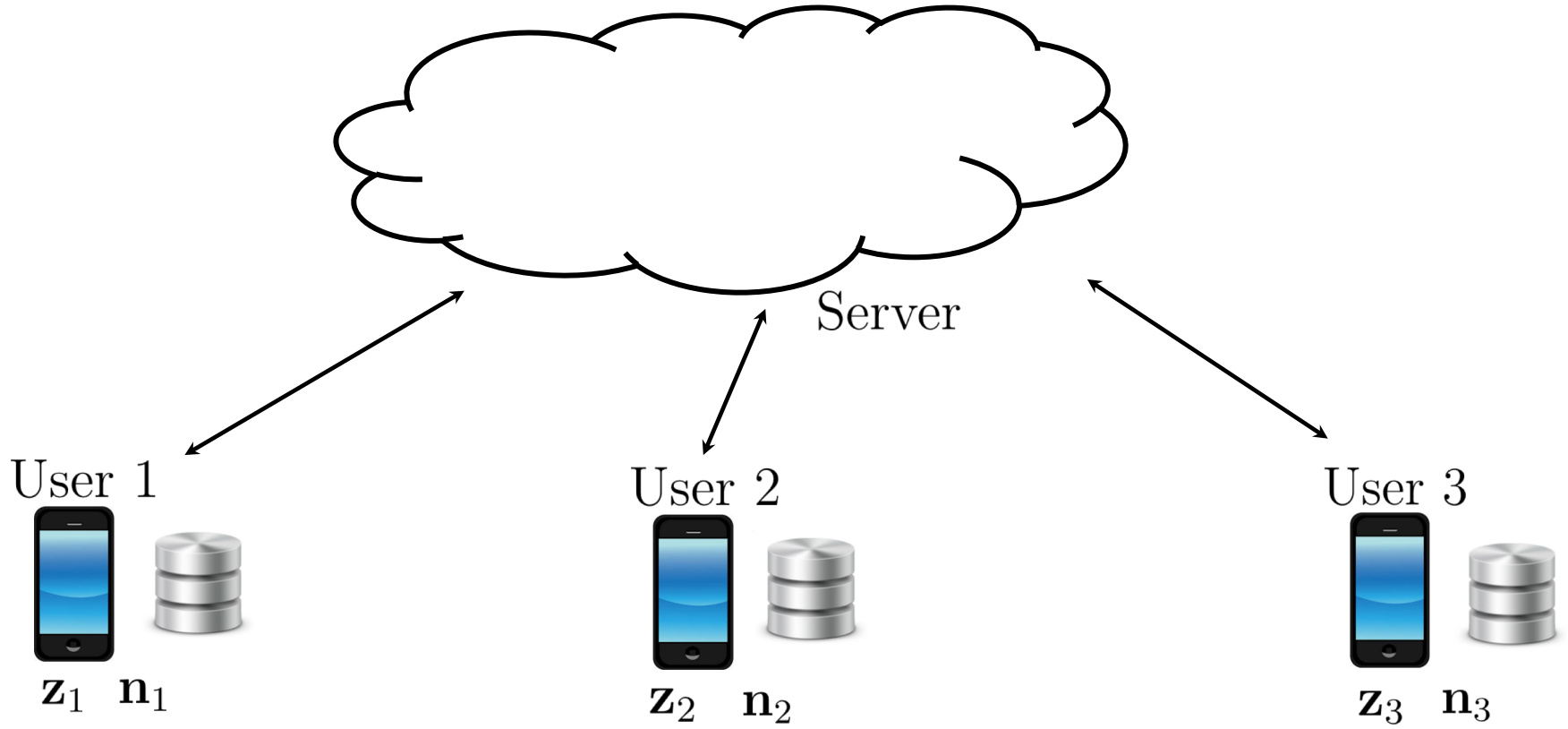
adds randomness that protects the local models

3) One-shot aggregate-model recovery

recovers the aggregate-model by canceling out the aggregate-mask

Example (LightSecAgg)

1) Offline encoding and sharing of local masks



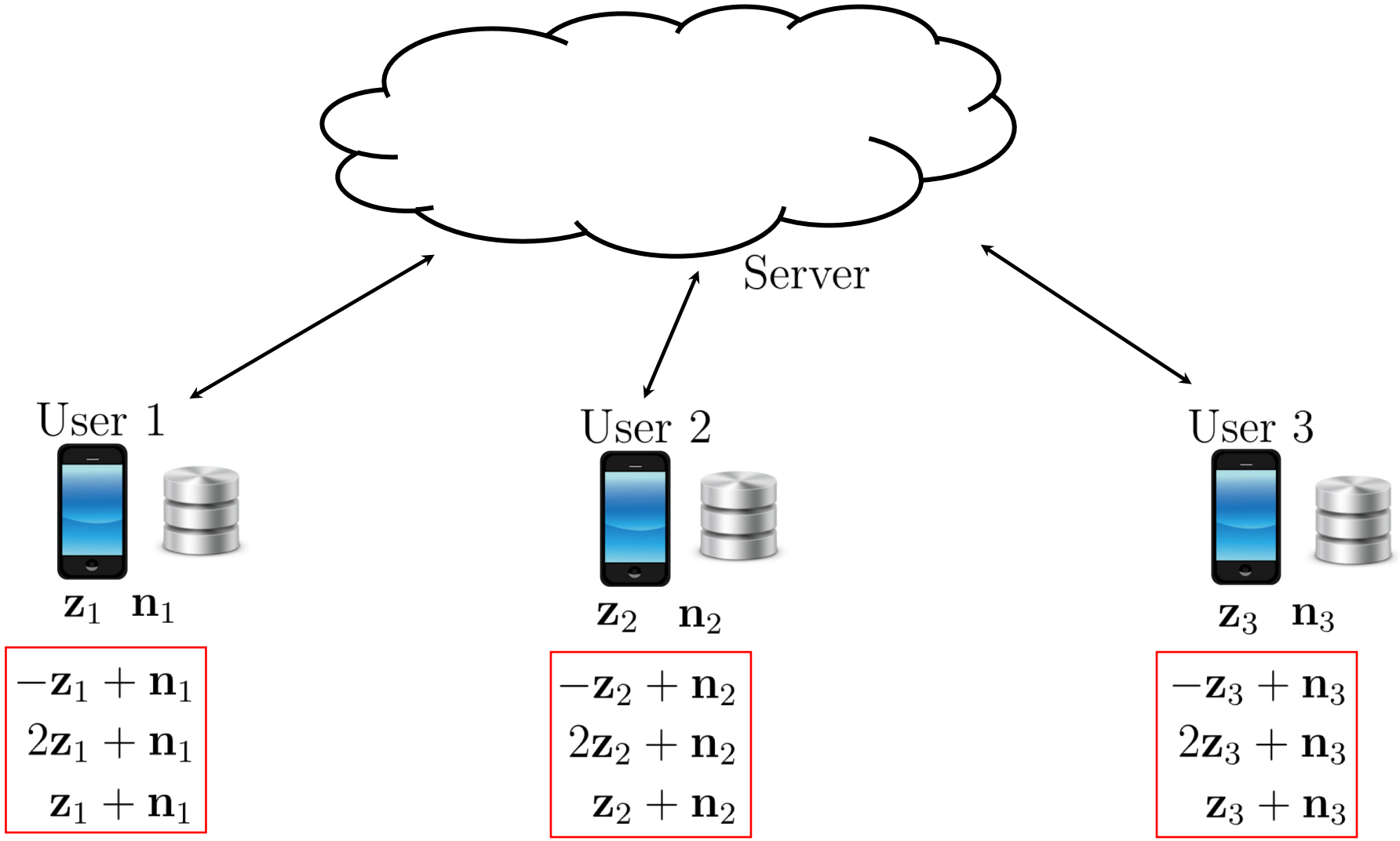
Encoding via MDS

$$\begin{bmatrix} -\mathbf{z}_1 + \mathbf{n}_1 \\ 2\mathbf{z}_1 + \mathbf{n}_1 \\ \mathbf{z}_1 + \mathbf{n}_1 \end{bmatrix}$$

1) $\mathbf{z}_1$ and $\mathbf{n}_1$ can be recovered by any 2 out of 3 encoded masks.
   => robustness against dropped users
2) $\mathbf{n}_1$ protects $\mathbf{z}_1$ => privacy against 1 curious user

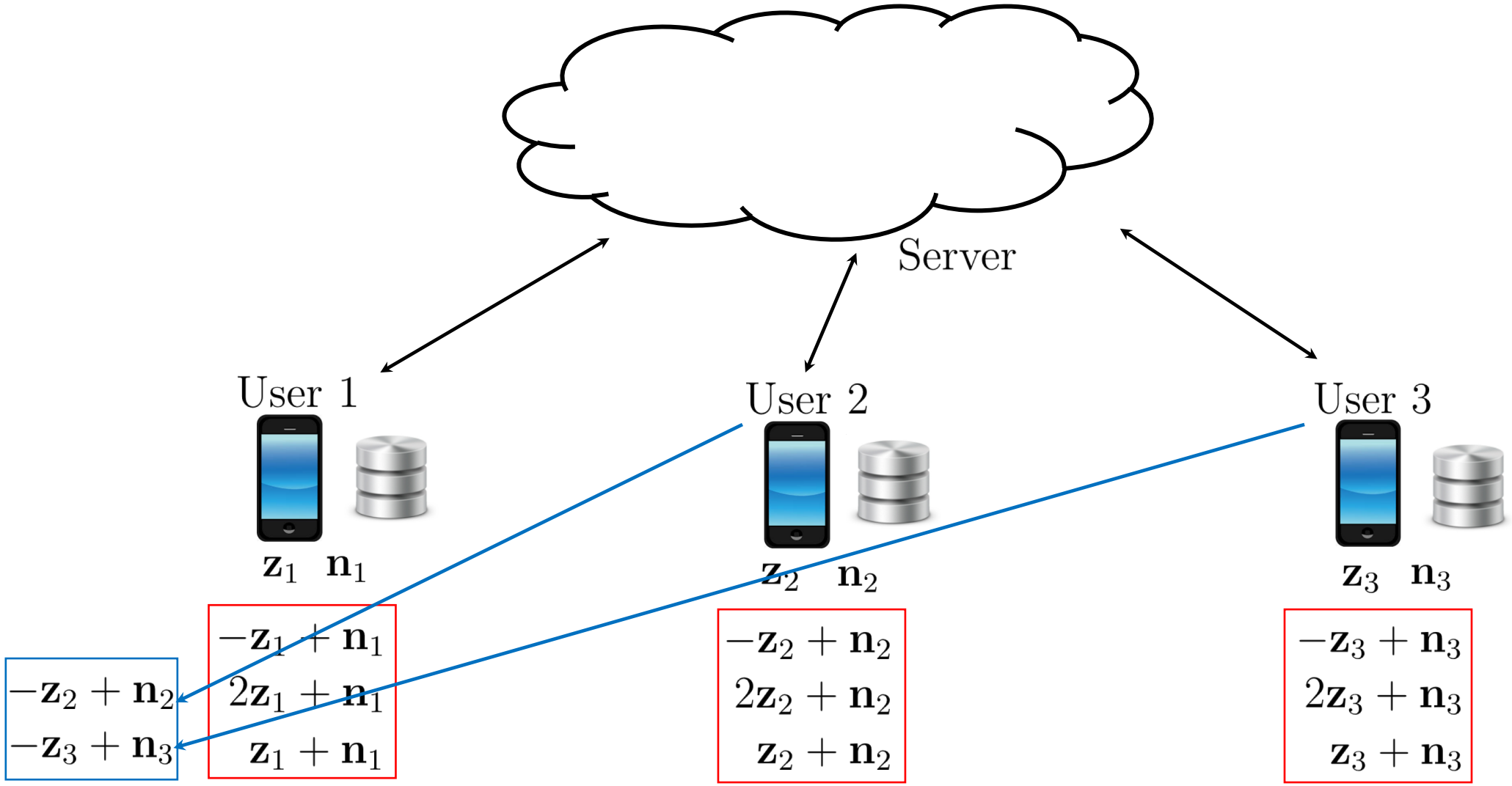
Example (LightSecAgg)

1) Offline encoding and sharing of local masks



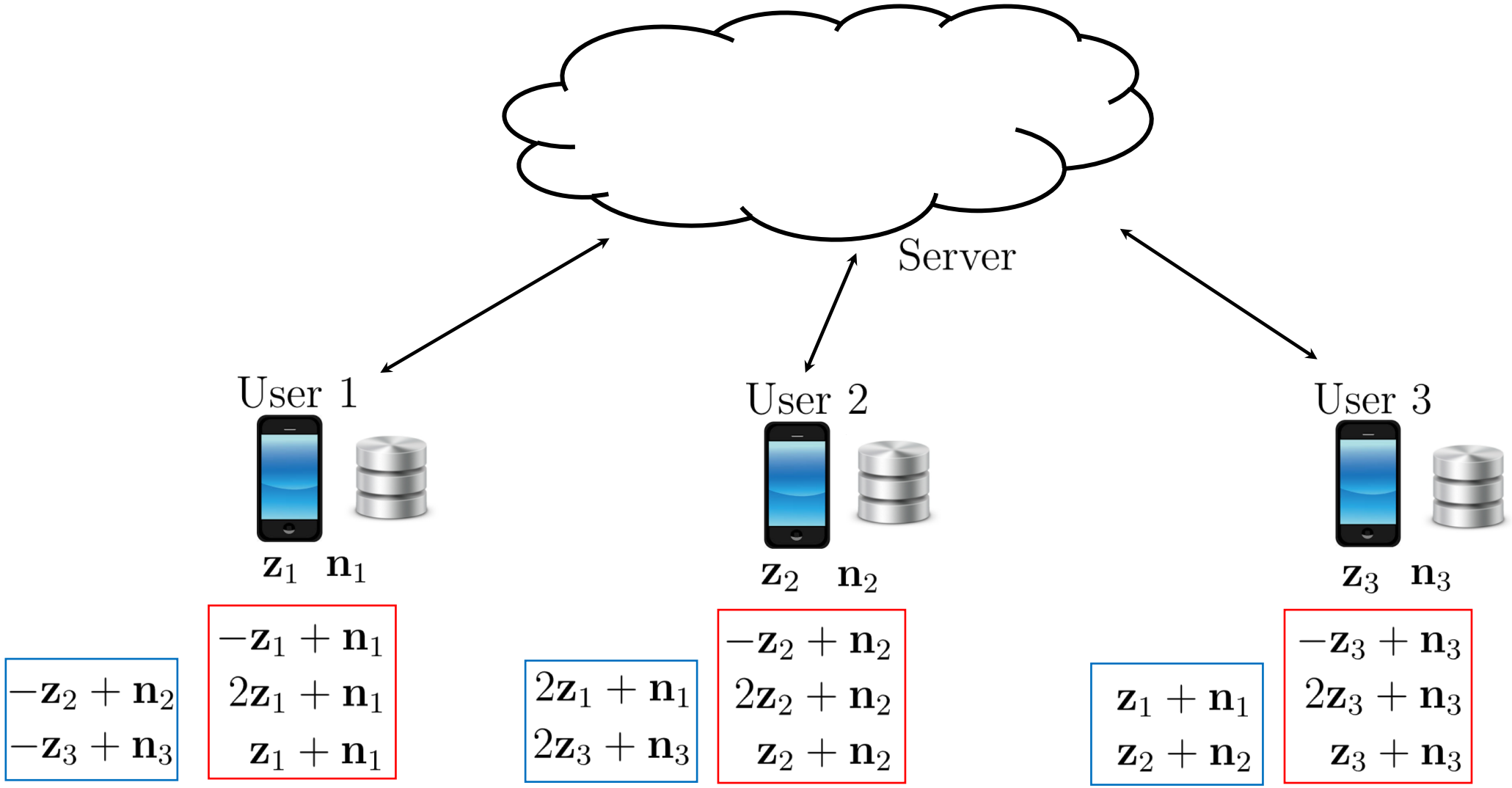
Example (LightSecAgg)

1) Offline encoding and sharing of local masks



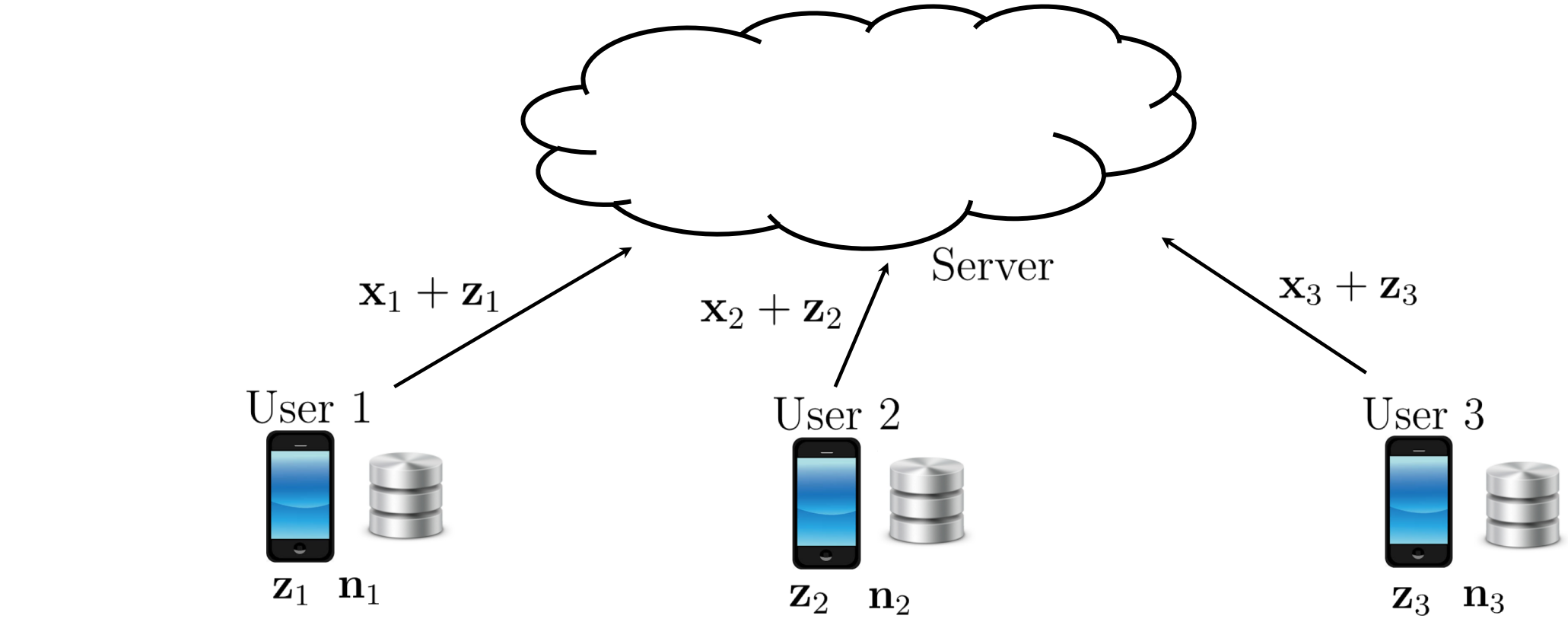
Example (LightSecAgg)

1) Offline encoding and sharing of local masks



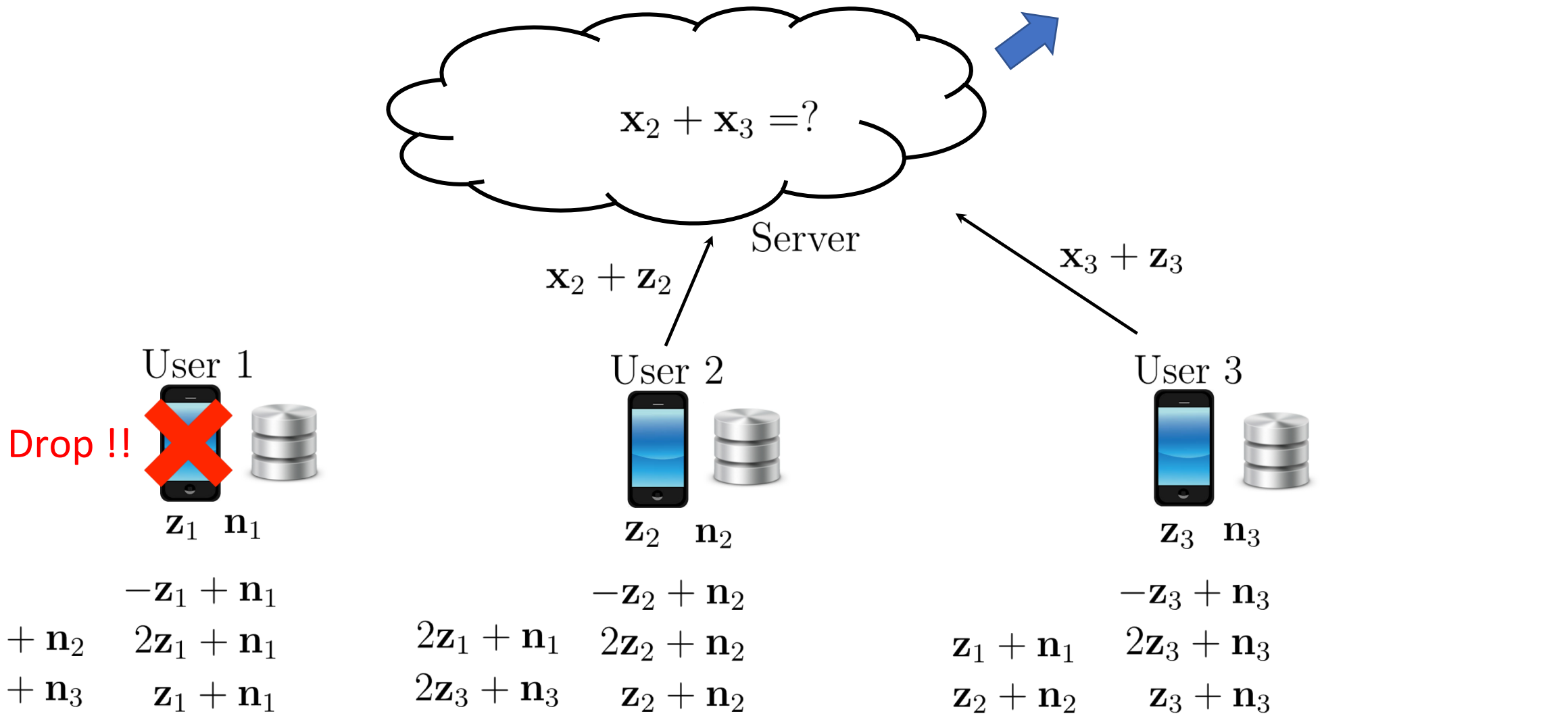
Example (LightSecAgg)

2) Masking and uploading of local models



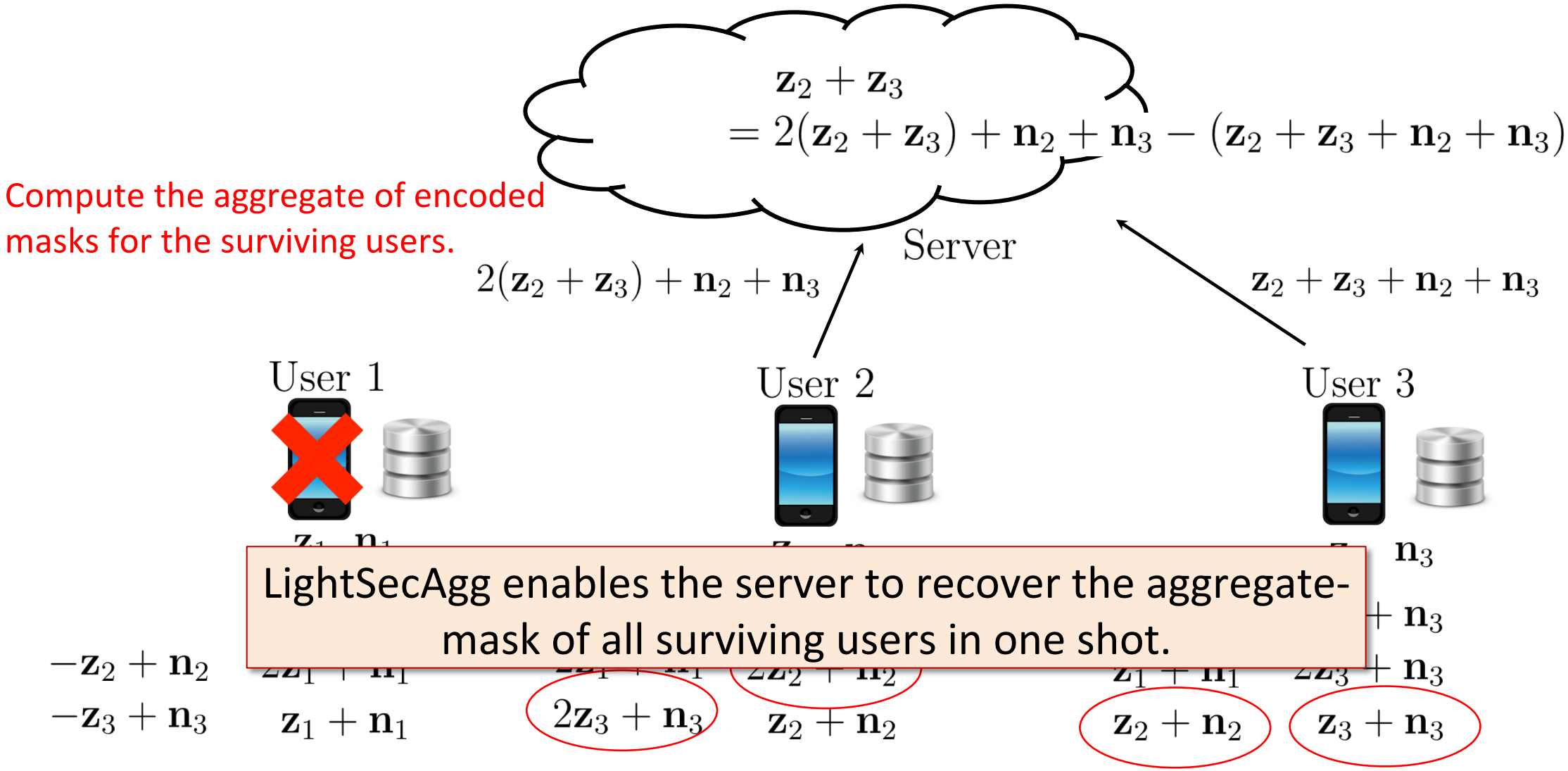
Example (LightSecAgg)

2) Masking and uploading of local models



Example (LightSecAgg)

3) One-shot aggregate-model recovery

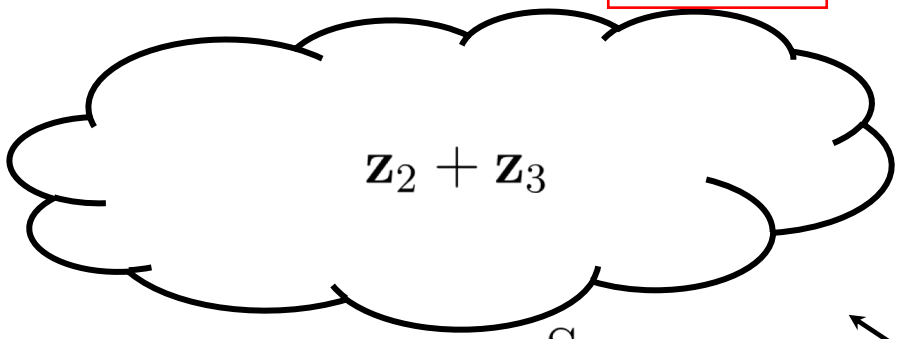


Example (LightSecAgg)

3) One-shot aggregate-model recovery

Aggregate-model

$$\mathbf{x}_2 + \mathbf{x}_3 = (\mathbf{x}_2 + \mathbf{z}_2) + (\mathbf{x}_3 + \mathbf{z}_3) - (\mathbf{z}_2 + \mathbf{z}_3)$$



Server

$2(\mathbf{z}_2 + \mathbf{z}_3) + \mathbf{n}_2 + \mathbf{n}_3$

$\mathbf{z}_2 + \mathbf{z}_3 + \mathbf{n}_2 + \mathbf{n}_3$

User 1

$\mathbf{z}_1 \quad \mathbf{n}_1$

$-\mathbf{z}_1 + \mathbf{n}_1$

$-\mathbf{z}_2 + \mathbf{n}_2 \quad 2\mathbf{z}_1 + \mathbf{n}_1$

$-\mathbf{z}_3 + \mathbf{n}_3 \quad \mathbf{z}_1 + \mathbf{n}_1$

User 2

$\mathbf{z}_2 \quad \mathbf{n}_2$

$-\mathbf{z}_2 + \mathbf{n}_2$

$2\mathbf{z}_1 + \mathbf{n}_1 \quad 2\mathbf{z}_2 + \mathbf{n}_2$

$2\mathbf{z}_3 + \mathbf{n}_3 \quad \mathbf{z}_2 + \mathbf{n}_2$

User 3

$\mathbf{z}_3 \quad \mathbf{n}_3$

$-\mathbf{z}_3 + \mathbf{n}_3$

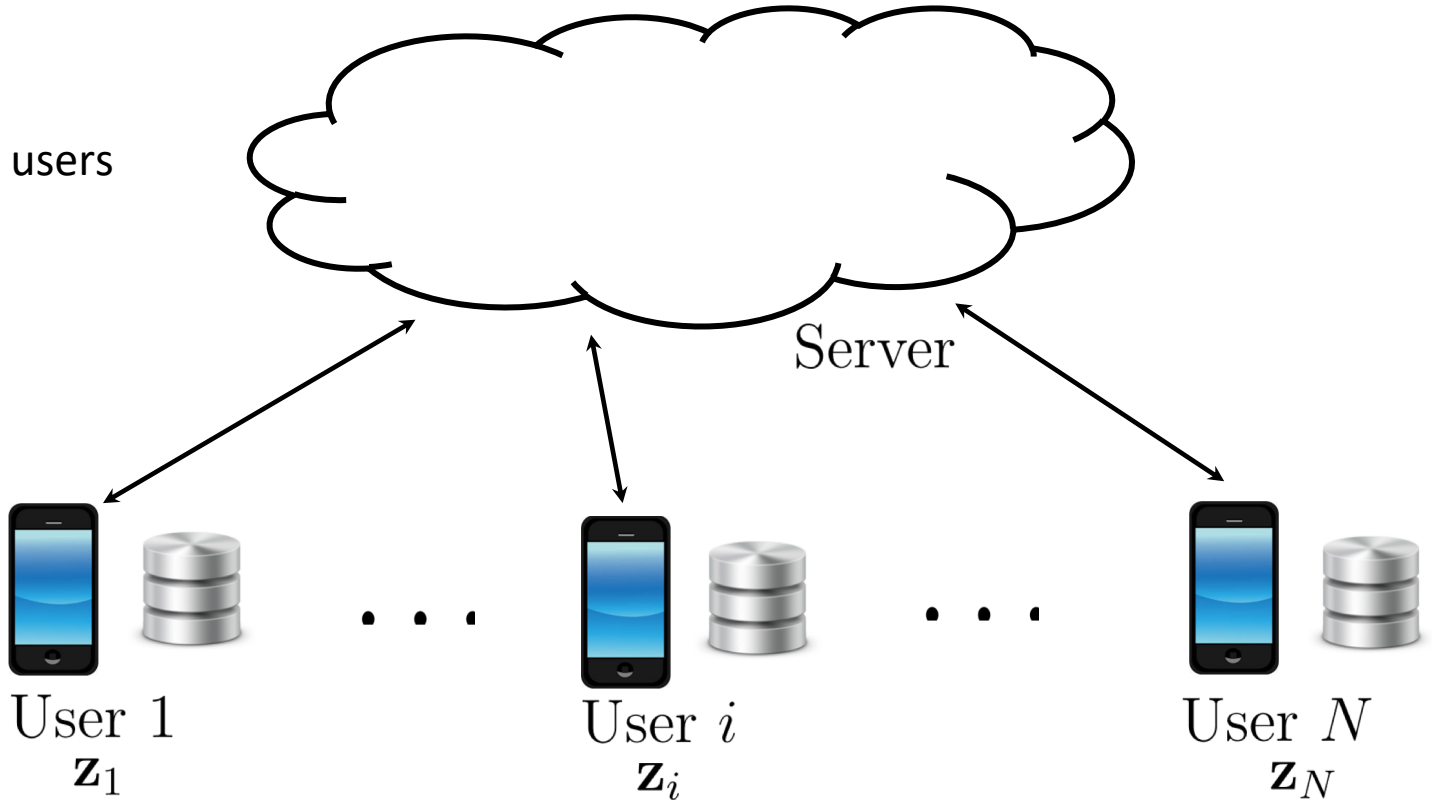
$\mathbf{z}_1 + \mathbf{n}_1 \quad 2\mathbf{z}_3 + \mathbf{n}_3$

$\mathbf{z}_2 + \mathbf{n}_2 \quad \mathbf{z}_3 + \mathbf{n}_3$

Phase 1: Offline Encoding and Sharing of Local Masks

Design Parameters:
U: targeted number of surviving users
T: privacy guarantee
D: dropout-resiliency guarantee

$N-D \geq U > T \geq 0$



Local mask partitioning $[\mathbf{z}_1]_1, \cdots, [\mathbf{z}_1]_{U-T}$

$$\mathbf{n}_1 = [[\mathbf{n}_1]_1, \cdots, [\mathbf{n}_1]_T]$$

Encoding via T-private MDS $[\tilde{\mathbf{z}}_1]_1, \cdots, [\tilde{\mathbf{z}}_1]_N$

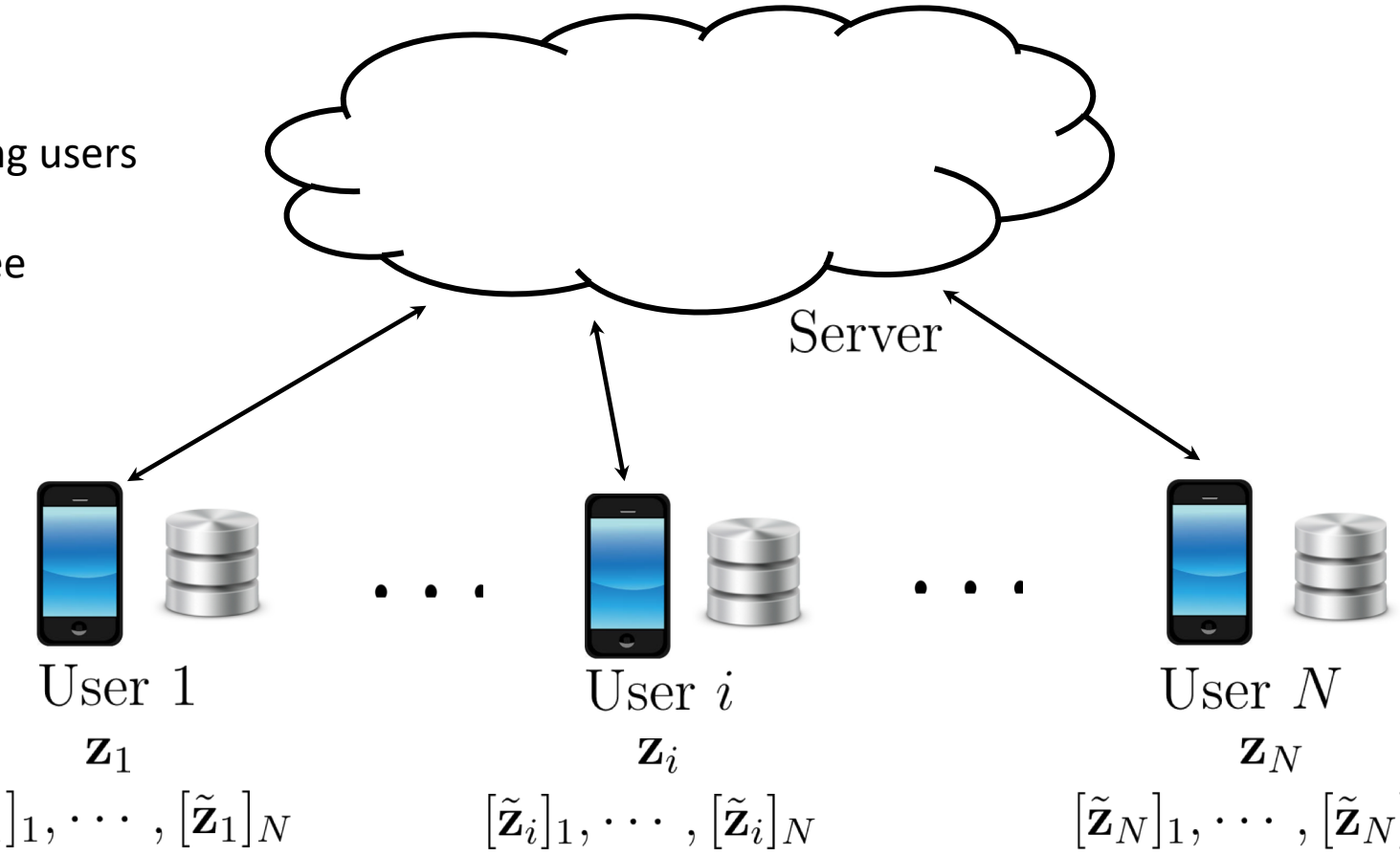
1) $\mathbf{z}_1$ and $\mathbf{n}_1$ can be recovered by any U encoded masks.
=> robustness against D dropped users

2) A T-private MDS guarantees $I(\mathbf{z}_1; \{[\tilde{\mathbf{z}}_1]_j\}_{j\in\mathcal{T}}) = 0$
=> privacy against any T colluding users

Phase 1: Offline Encoding and Sharing of Local Masks

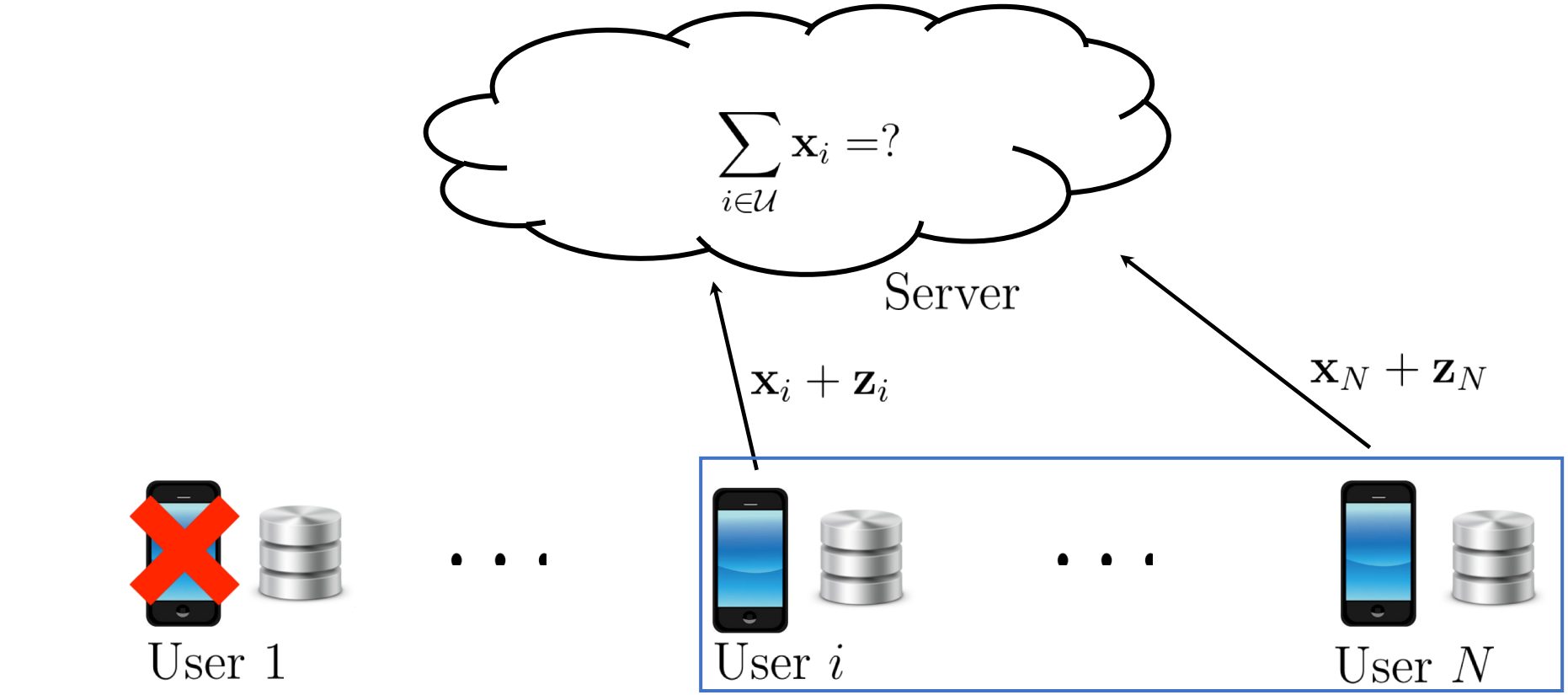
Design Parameters:
U: targeted number of surviving users
T: privacy guarantee
D: dropout-resiliency guarantee

$N-D \geq U > T \geq 0$



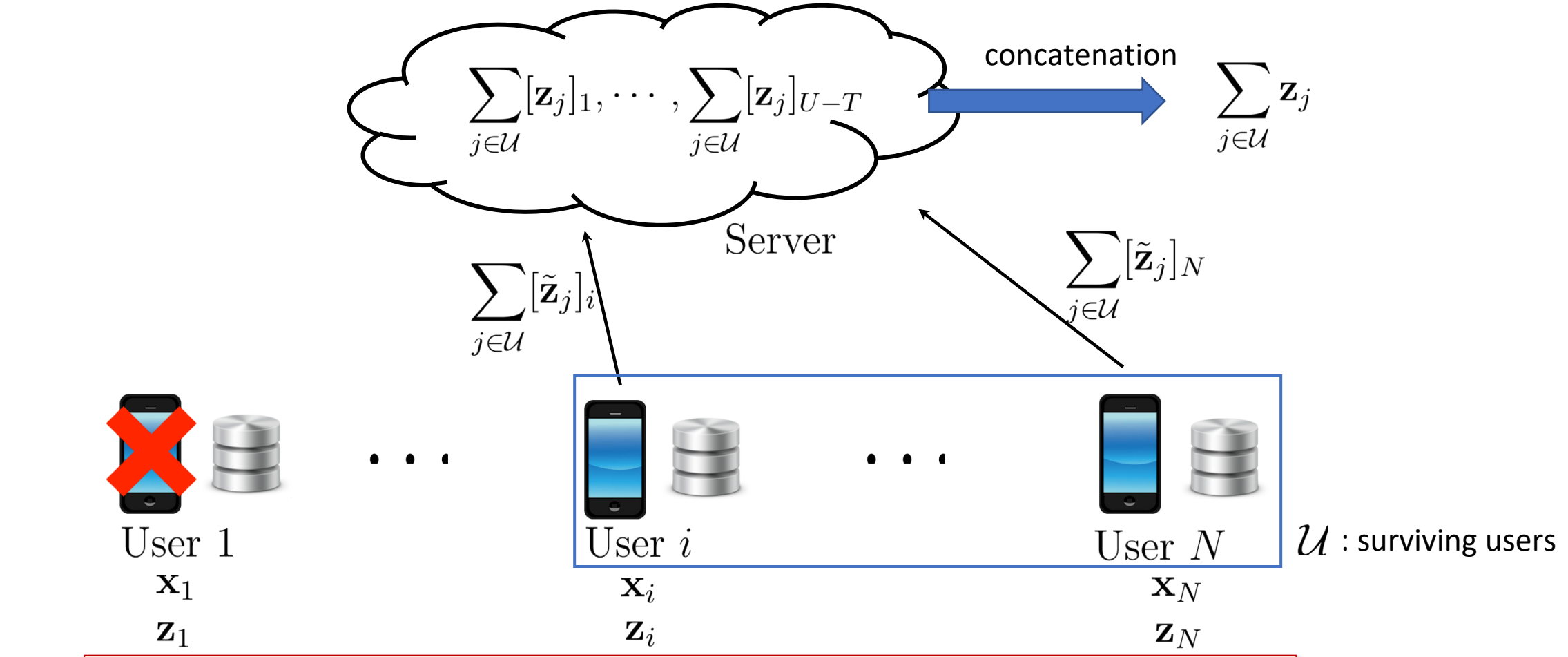
Local mask sharing $[\tilde{\mathbf{z}}_1]_1, [\tilde{\mathbf{z}}_2]_1, \cdots, [\tilde{\mathbf{z}}_N]_1$ $[\tilde{\mathbf{z}}_1]_i, [\tilde{\mathbf{z}}_2]_i, \cdots, [\tilde{\mathbf{z}}_N]_i$ $[\tilde{\mathbf{z}}_1]_N, [\tilde{\mathbf{z}}_2]_N, \cdots, [\tilde{\mathbf{z}}_N]_N$

Phase 2: Masking and Uploading of Local Models



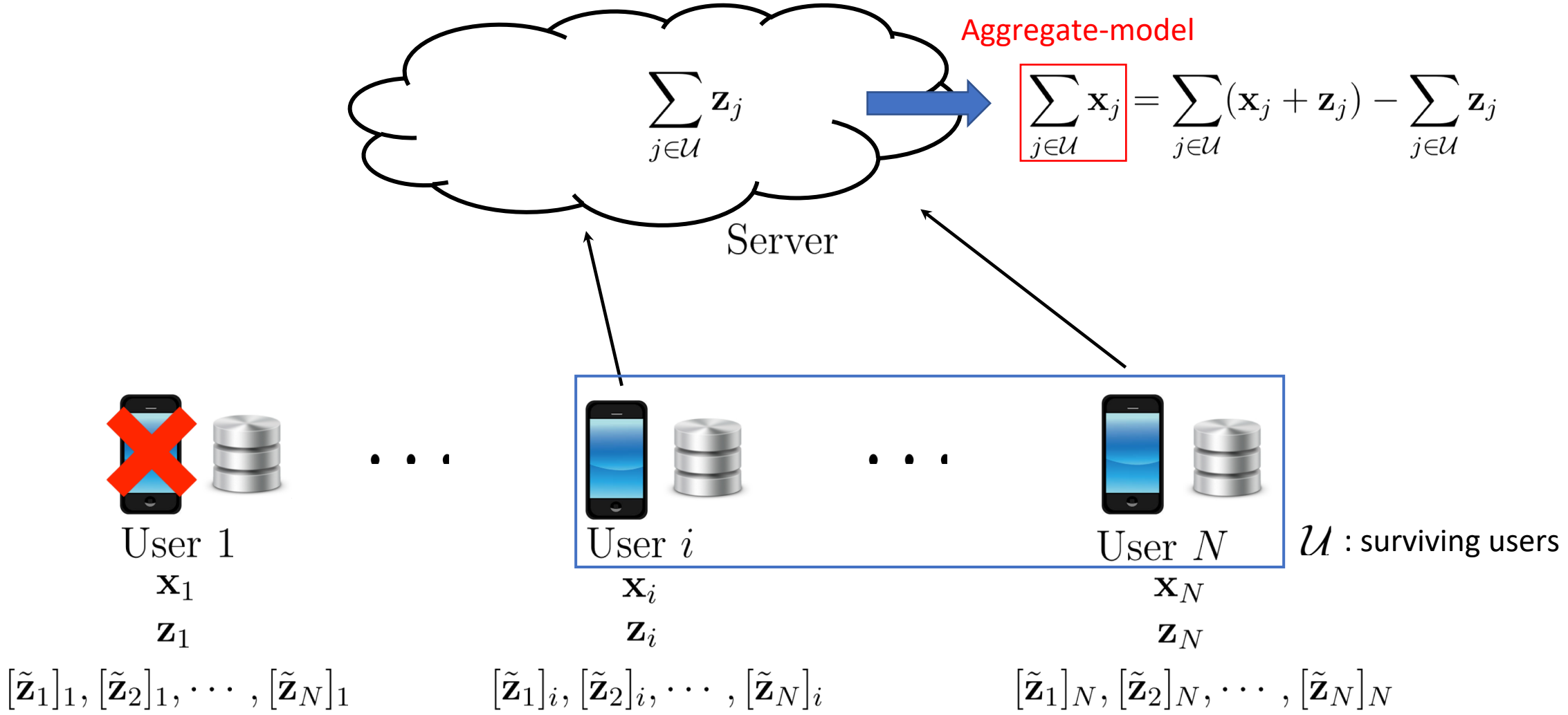
Phase 3: One-Shot Aggregate-Model Recovery

The server recovers the aggregate-mask after receiving any U messages via MDS decoding.



The server only needs to reconstruct one mask in the recovery phase, independent of the number of dropped users.

Phase 3: One-Shot Aggregate-Model Recovery



Theoretical Guarantees

Theorem: *LightSecAgg can simultaneously achieve*

(1) privacy guarantee against up to any T colluding users, and

(2) dropout-resiliency guarantee against up to any D dropped users,

for any pair of privacy guarantee T and dropout-resiliency guarantee D such that $T + D < N$.

Theoretical Guarantees

- Complexity comparison between SecAgg, SecAgg+ and LightSecAgg:
  - d: model size.
  - s: length of the secret keys.

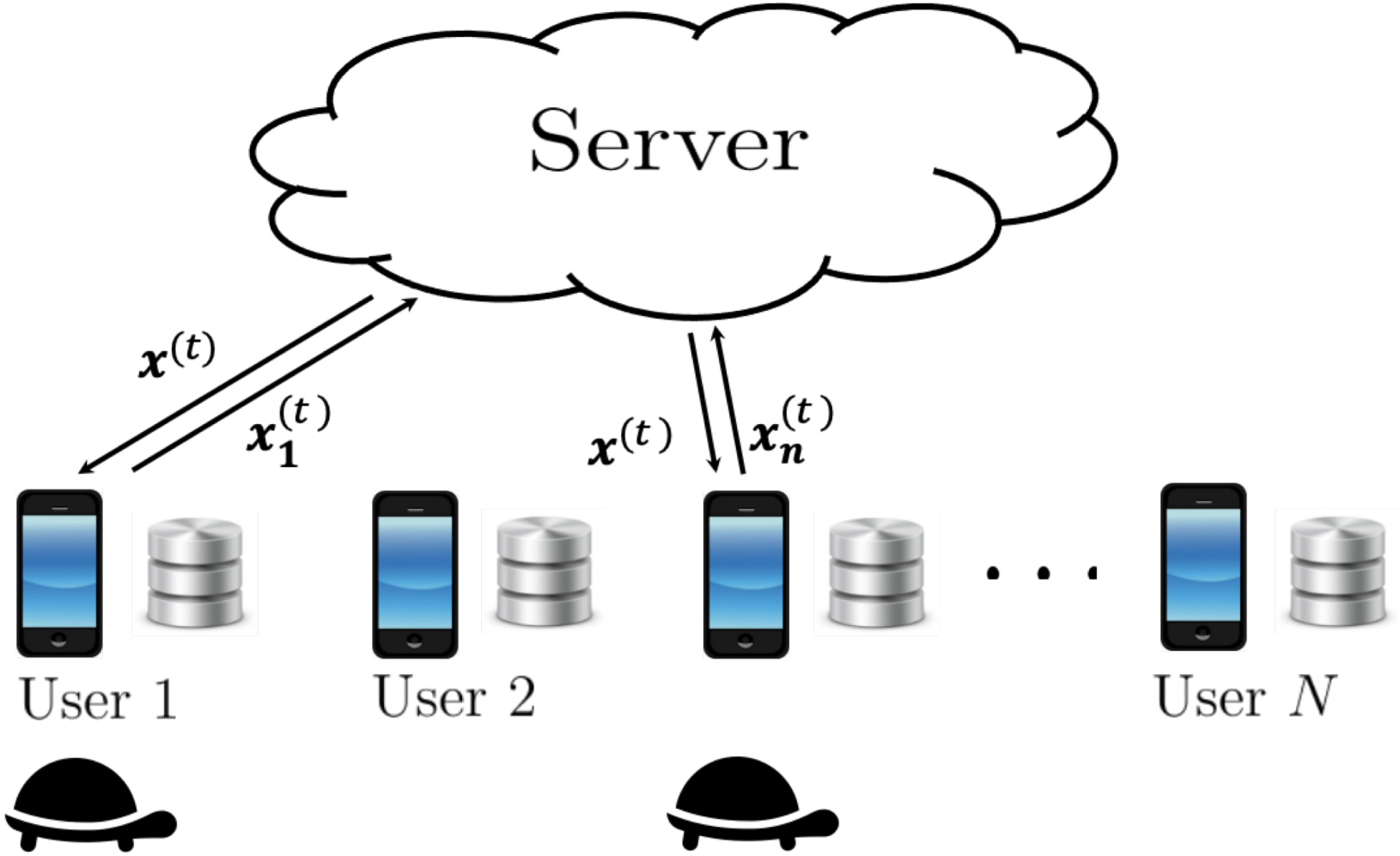
	SecAgg	SecAgg+	LightSecAgg
Offline communication per user	$O(sN)$	$O(s \log N)$	$O(d)$
Offline computation per user	$O(dN + sN^2)$	$O(d \log N + s \log^2 N)$	$O(d \log N)$
Online communication per user	$O(d + sN)$	$O(d + s \log N)$	$O(d)$
Online communication at server	$O(dN + sN^2)$	$O(dN + sN \log N)$	$O(dN)$
Online computation per user	$O(d)$	$O(d)$	$O(d)$
Reconstruction complexity at server	$O(dN^2)$	$O(dN \log N)$	$O(d \log N)$

LightSecAgg significantly improves the computation efficiency at the server during aggregation.

Part 2. LightSecAgg for Asynchronous Federated Learning

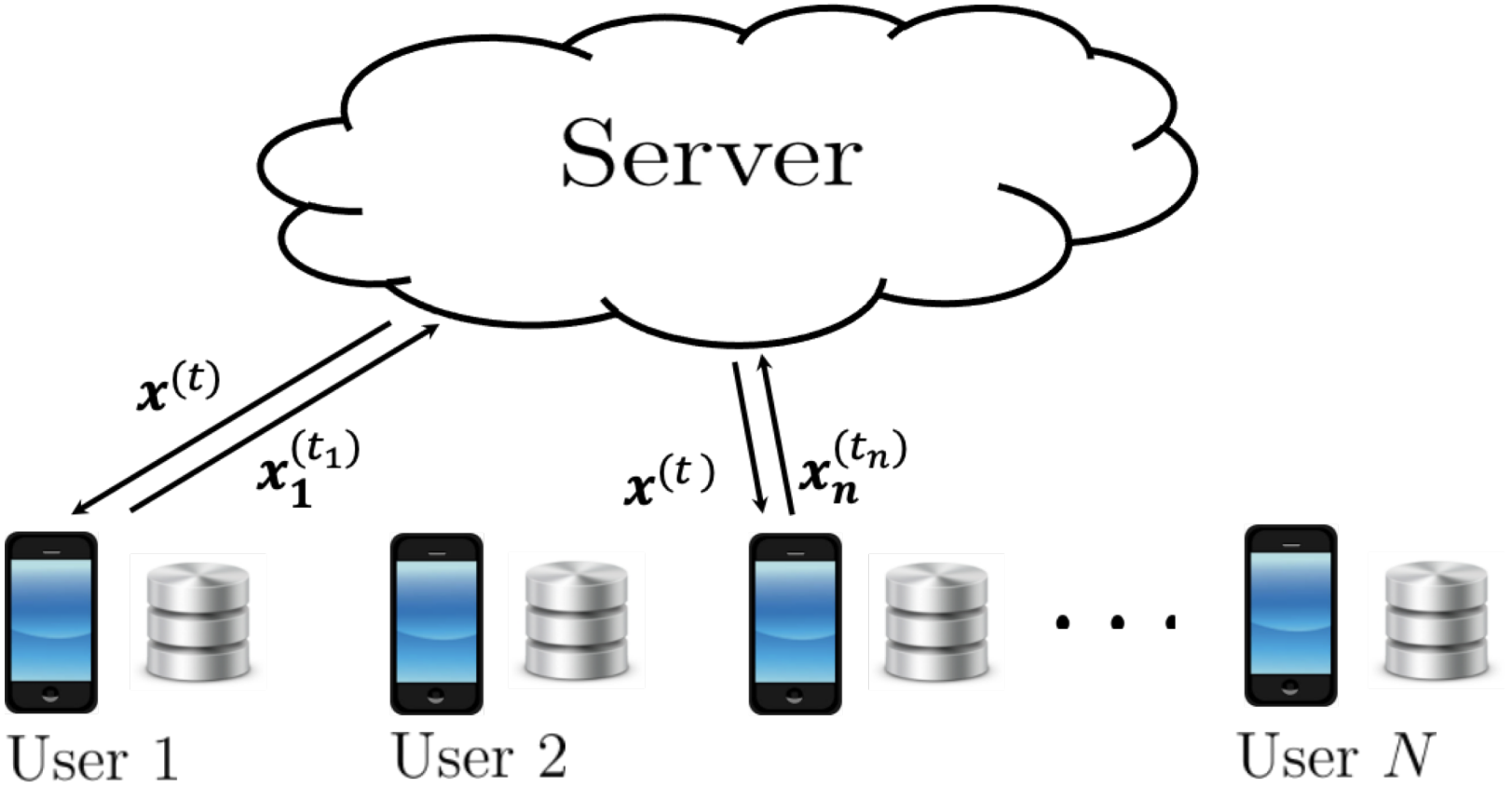
Asynchronous FL

- Synchronous FL suffers from stragglers!



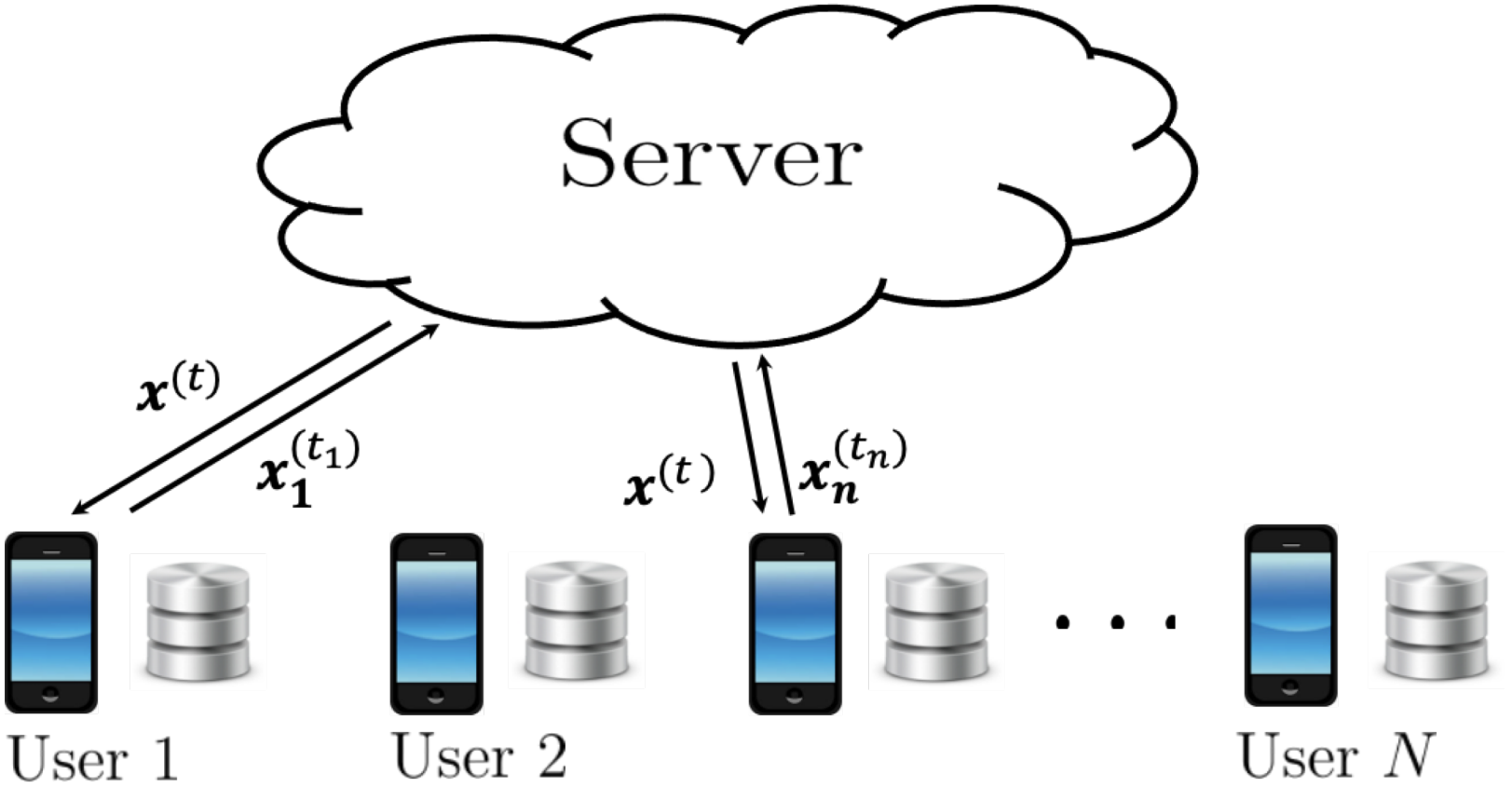
Asynchronous FL

- Updates are not synchronized.
- Each local model received updates the global model.



Asynchronous FL

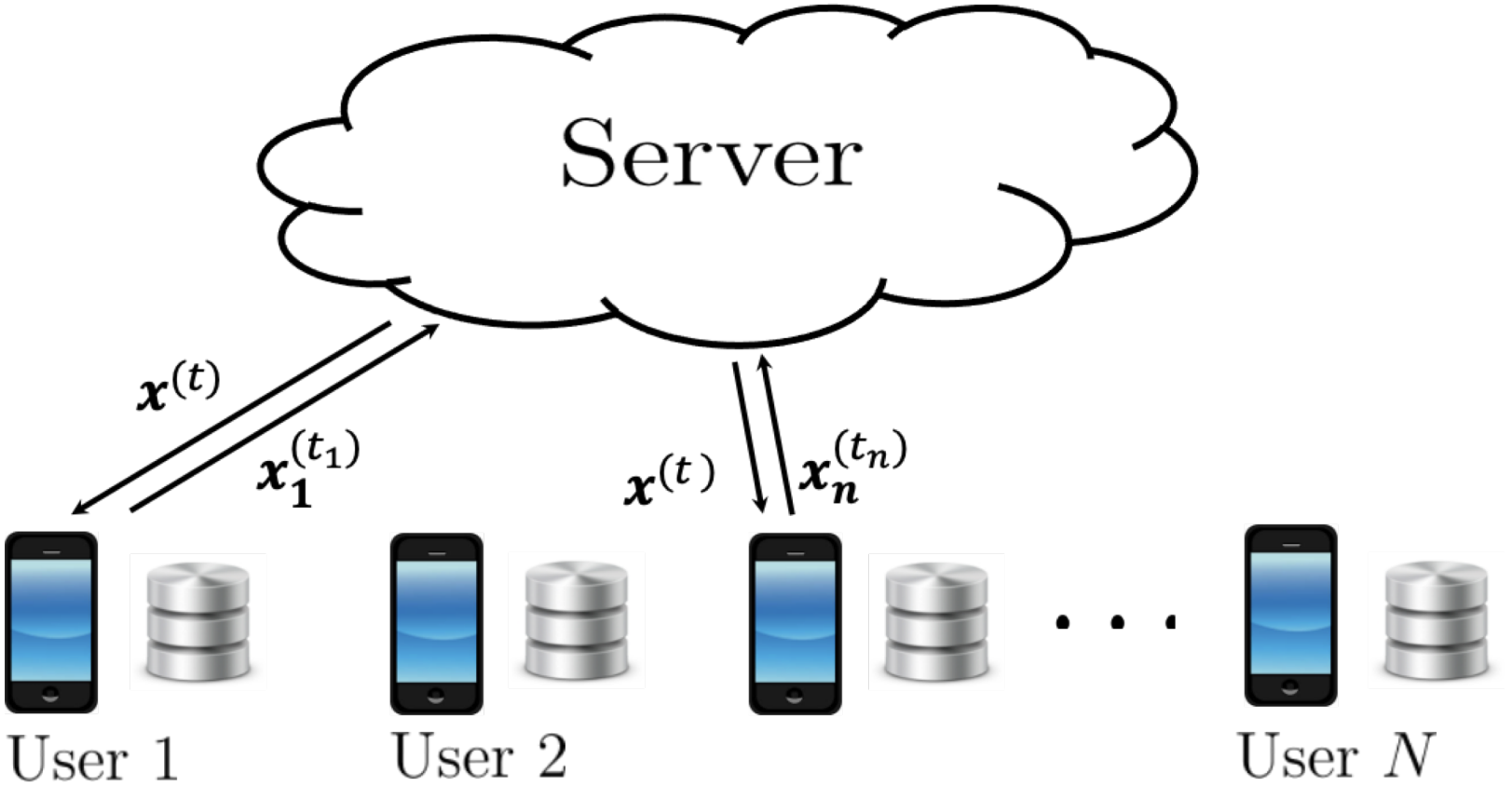
- Updates are not synchronized.
- Each local model received updates the global model.



Not compatible with secure aggregation!

Asynchronous FL

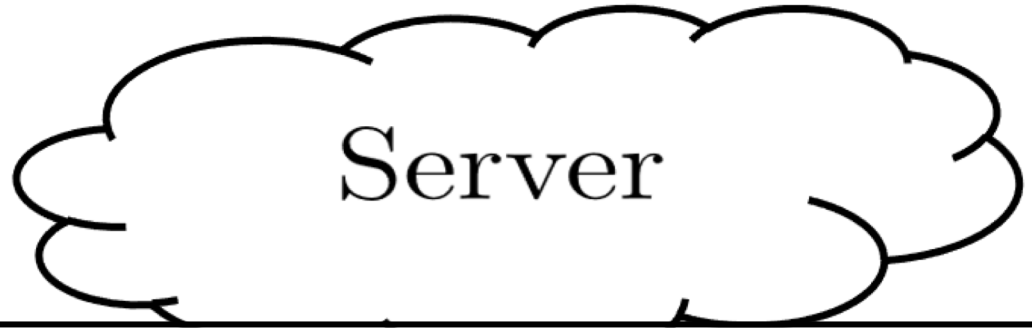
- Updates are not synchronized.
- Each local model received updates the global model.



How we enable secure aggregation in asynchronous FL?

Asynchronous FL

- Updates are not synchronized.
- Each local model received updates the global model.



Use

Federated Learning with Buffered Asynchronous Aggregation

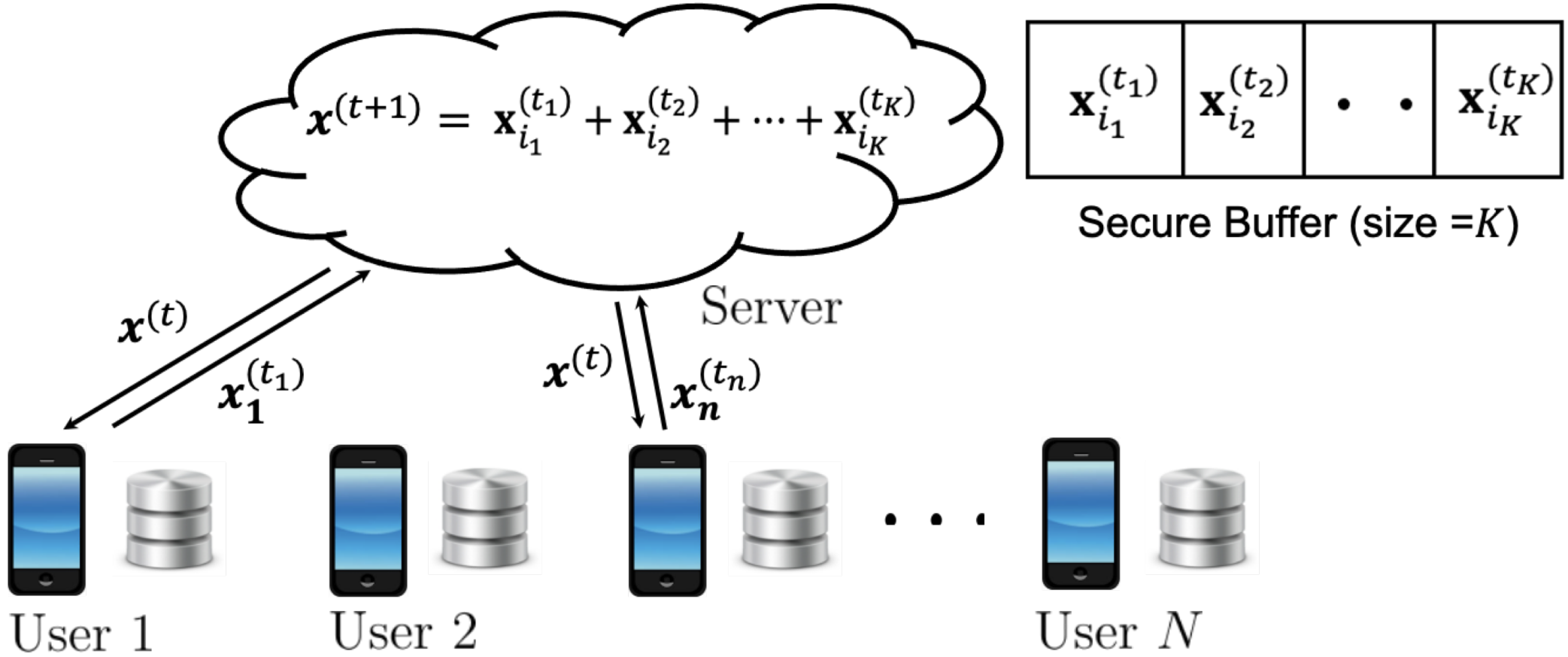
John Nguyen Kshitiz Malik Hongyuan Zhan Ashkan Yousefpour
Michael Rabbat Mani Malek Dzmitry Huba

Facebook
{ngjhn,kmalik2,hyzhan,yousefpour,mikerabbat,manimalek,huba}@fb.com

How we enable secure aggregation in asynchronous FL?

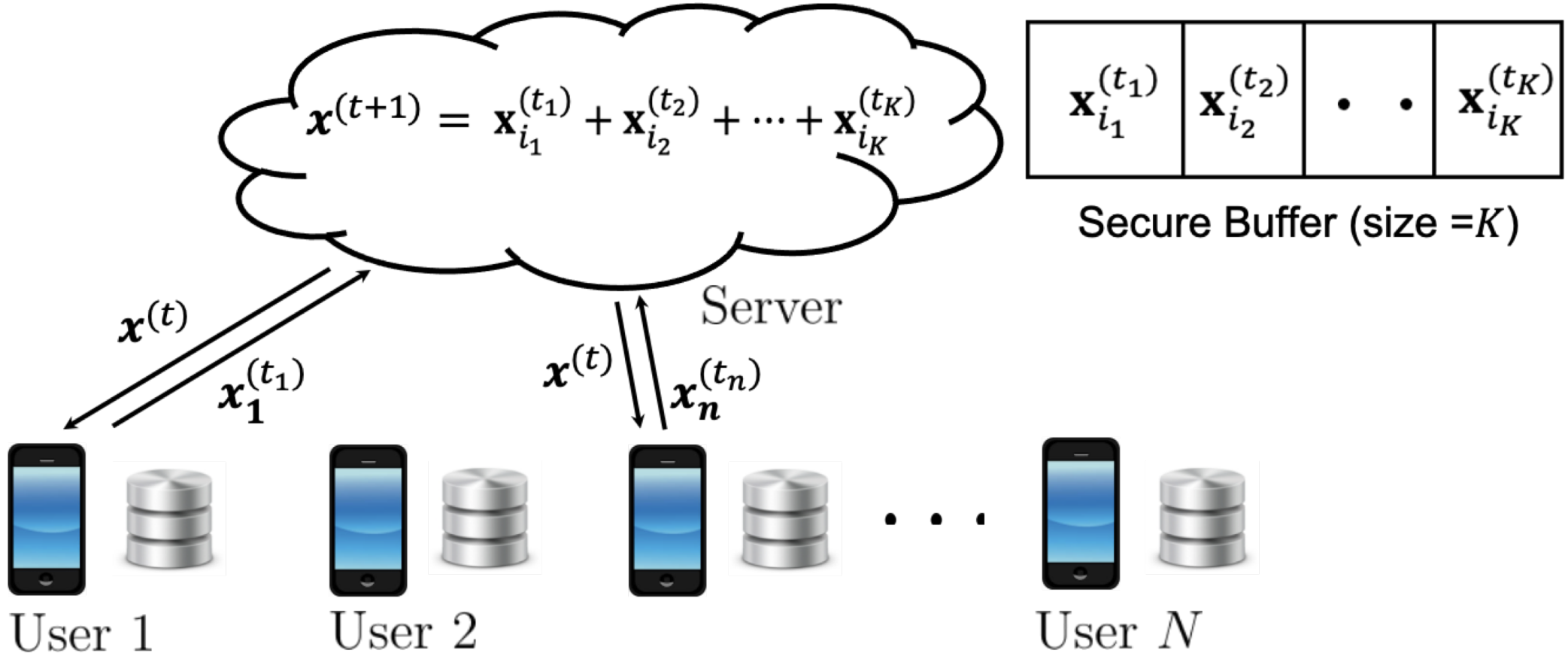
Asynchronous FL

- Typical Asynchronous FL: K=1 (not compatible with secure aggregation)
- Buffered Asynchronous FL (FedBuff): K>1



Asynchronous FL

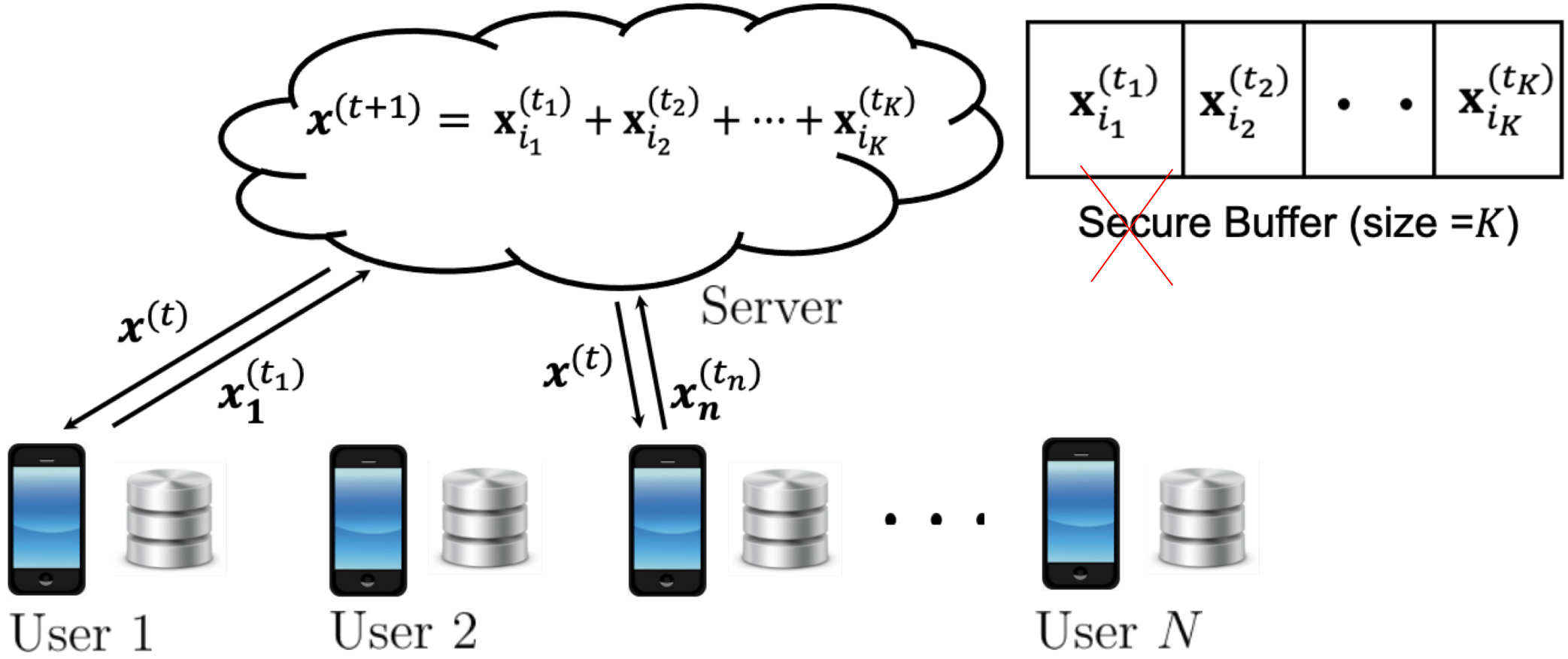
- Typical Asynchronous FL: K=1 (not compatible with secure aggregation)
- Buffered Asynchronous FL (FedBuff): K>1



The TEE-enabled secure buffer, however, has limited memory.
(K must be small!)

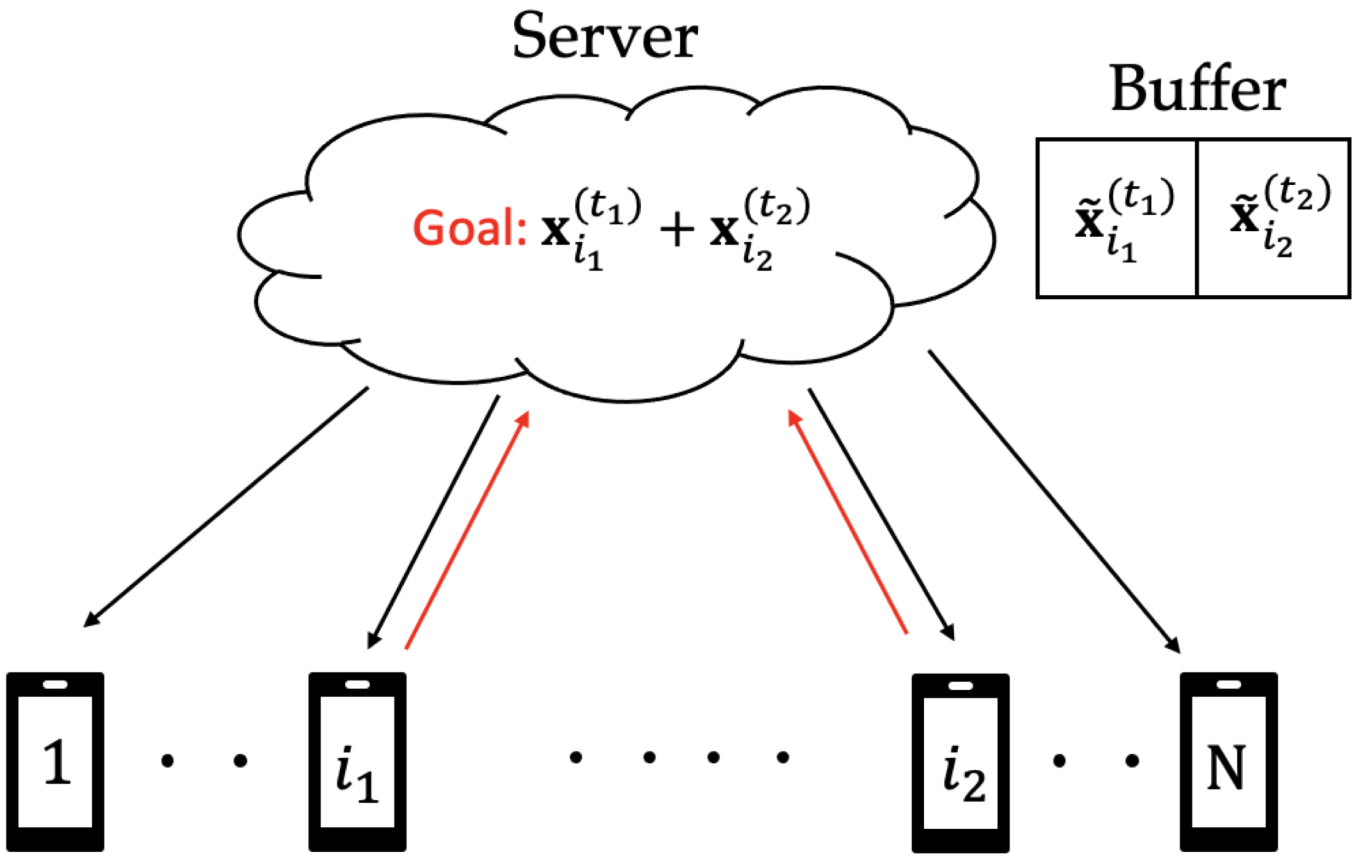
Asynchronous FL

- Typical Asynchronous FL: K=1 (not compatible with secure aggregation)
- Buffered Asynchronous FL (FedBuff): K>1



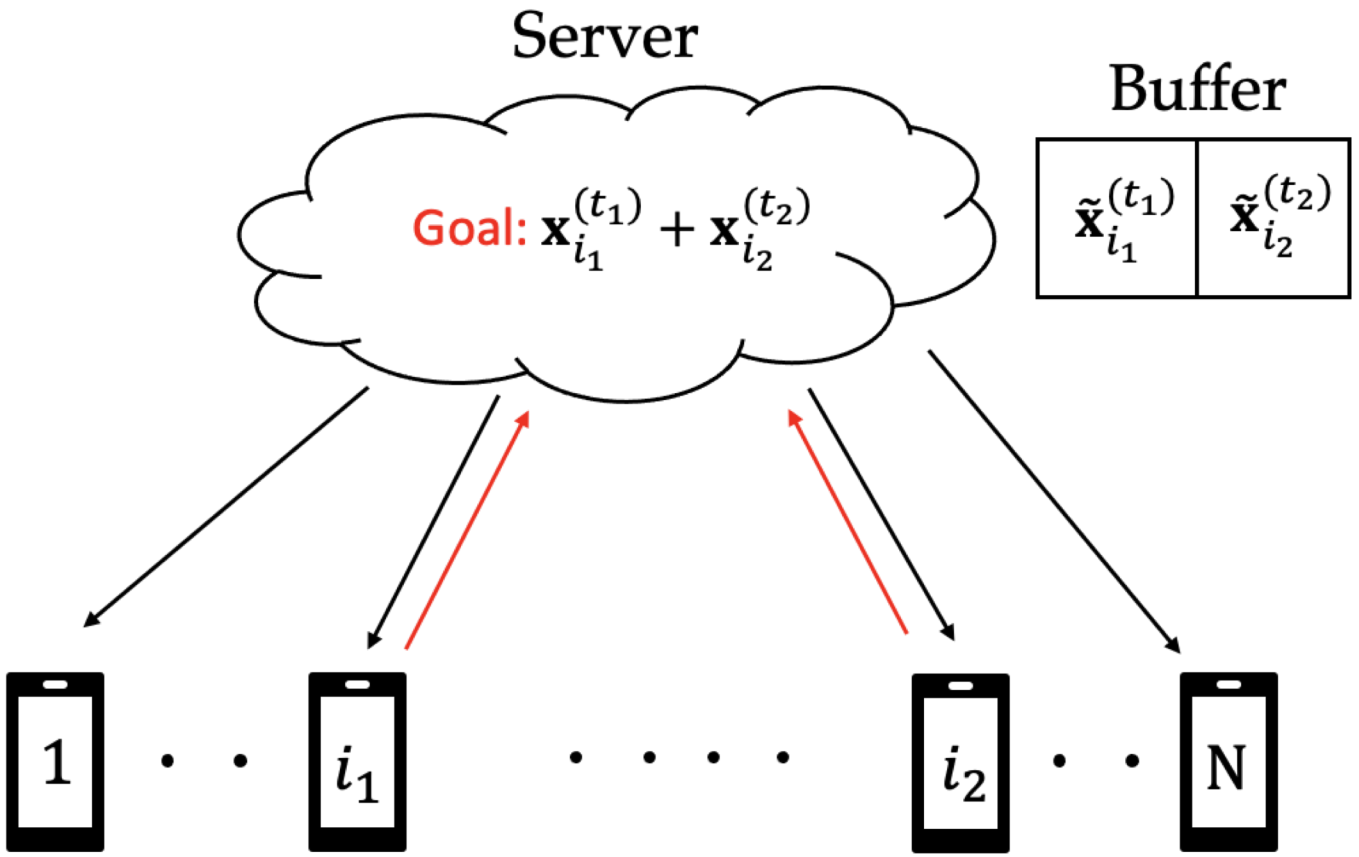
LightSecAgg does not require a secure buffer!

Incompatibility of SecAgg with Asynchronous FL



The masks do not cancel out due to the mismatch in staleness!

Incompatibility of SecAgg with Asynchronous FL



$$\tilde{\mathbf{x}}_{i_1}^{(t_1)} = \mathbf{x}_{i_1}^{(t_1)} + \mathbf{z}_{i_1,i_2}^{(t_1)} + \cdots \qquad \tilde{\mathbf{x}}_{i_2}^{(t_2)} = \mathbf{x}_{i_2}^{(t_2)} - \mathbf{z}_{i_1,i_2}^{(t_2)} + \cdots$$

How to design the masks to cancel out even if they belong to different rounds?

Asynchronous LightSecAgg

Key objective: Design the masks such that they cancel out even if they belong to different training rounds.

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta_g \boldsymbol{g}^{(t)} \text{ where } \boldsymbol{g}^{(t)} = \sum_{i \in S^{(t)}} \tilde{\boldsymbol{x}}_i^{(t;t_i)} = \sum_{i \in S^{(t)}} \overline{\boldsymbol{x}}_i^{(t;t_i)} + \boxed{\sum_{i \in S^{(t)}} \boldsymbol{z}_i^{(t_i)}}$$

Our focus

LightSecAgg is compatible as it does not use pair-wise masking!

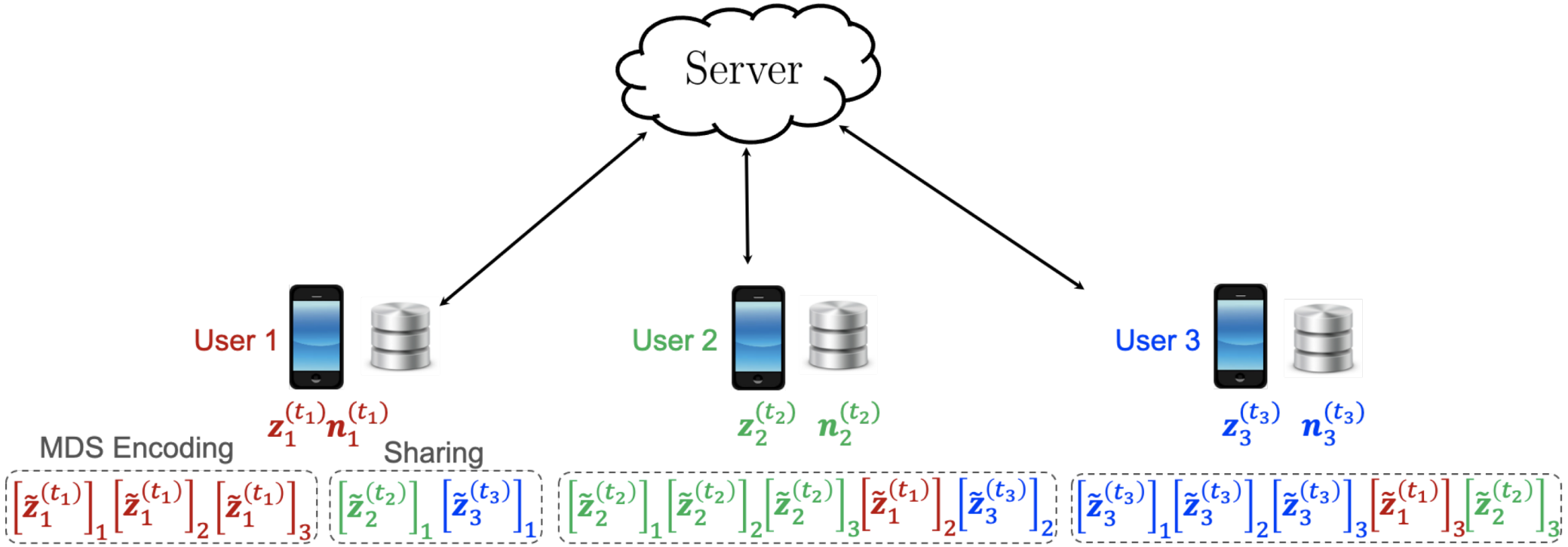
Asynchronous LightSecAgg

Key objective: Design the masks such that they cancel out even if they belong to different training rounds.

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta_g \boldsymbol{g}^{(t)} \text{ where } \boldsymbol{g}^{(t)} = \sum_{i \in S^{(t)}} \tilde{\boldsymbol{x}}_i^{(t;t_i)} = \sum_{i \in S^{(t)}} \overline{\boldsymbol{x}}_i^{(t;t_i)} + \sum_{i \in S^{(t)}} \boldsymbol{z}_i^{(t_i)}$$

Our focus

Step 1. Offline encoding and sharing of local masks



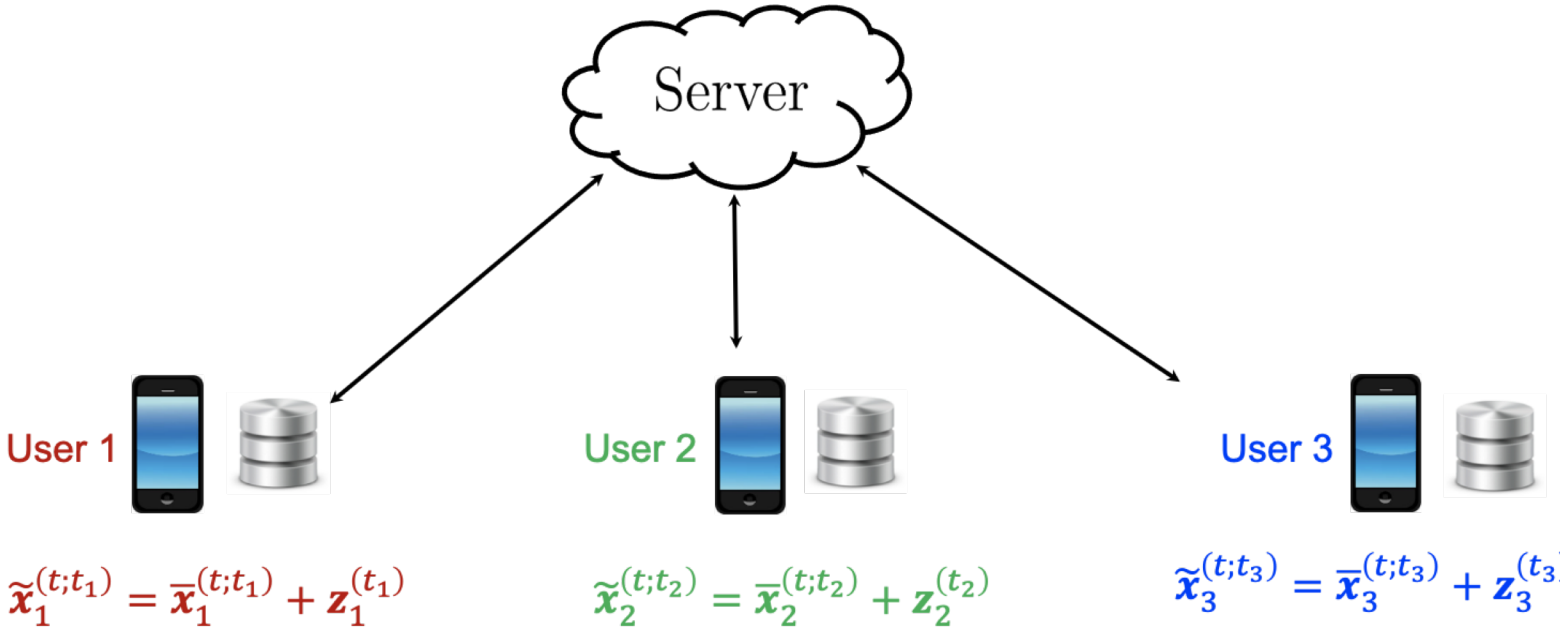
Asynchronous LightSecAgg

Key objective: Design the masks such that they cancel out even if they belong to different training rounds.

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta_g \boldsymbol{g}^{(t)} \text{ where } \boldsymbol{g}^{(t)} = \sum_{i \in S^{(t)}} \tilde{\boldsymbol{x}}_i^{(t;t_i)} = \sum_{i \in S^{(t)}} \overline{\boldsymbol{x}}_i^{(t;t_i)} + \boxed{\sum_{i \in S^{(t)}} \boldsymbol{z}_i^{(t_i)}}$$

Our focus

Step 2. Quantization & Masking



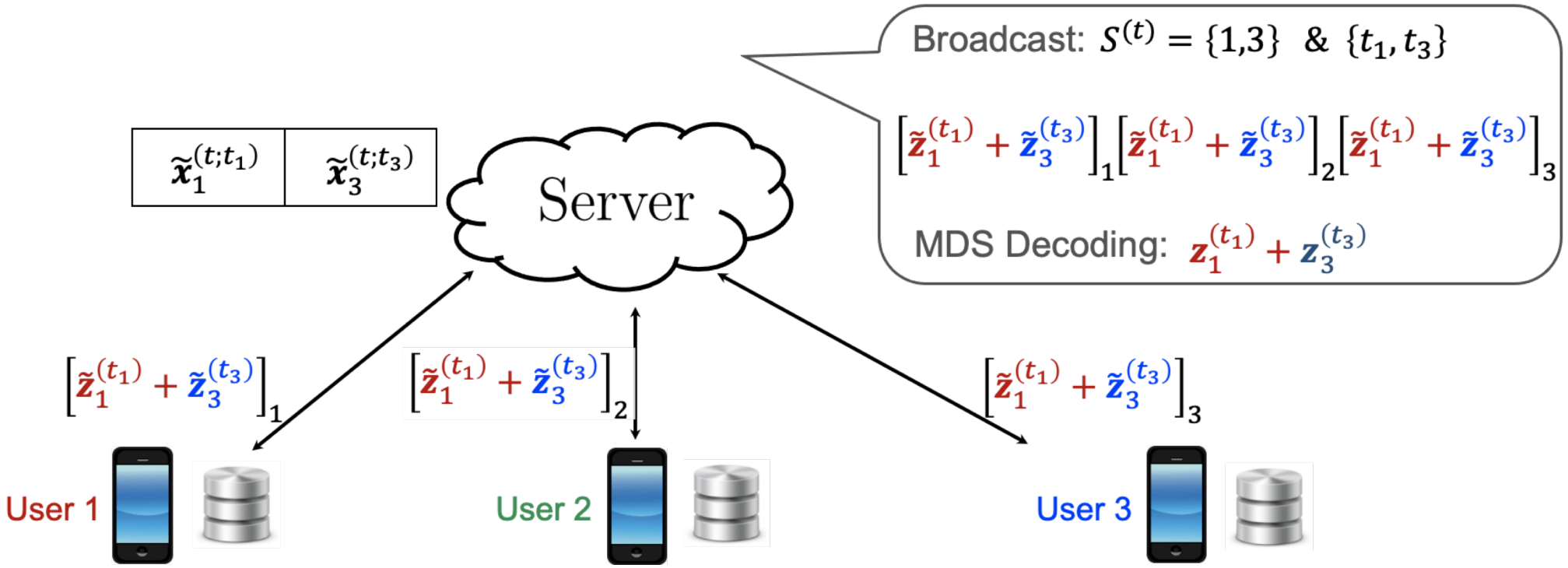
Asynchronous LightSecAgg

Key objective: Design the masks such that they cancel out even if they belong to different training rounds.

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta_g \boldsymbol{g}^{(t)} \text{ where } \boldsymbol{g}^{(t)} = \sum_{i \in S^{(t)}} \tilde{\boldsymbol{x}}_i^{(t;t_i)} = \sum_{i \in S^{(t)}} \overline{\boldsymbol{x}}_i^{(t;t_i)} + \boxed{\sum_{i \in S^{(t)}} \boldsymbol{z}_i^{(t_i)}}$$

Our focus

Step 3. One-Shot Recovery of Aggregate Masks

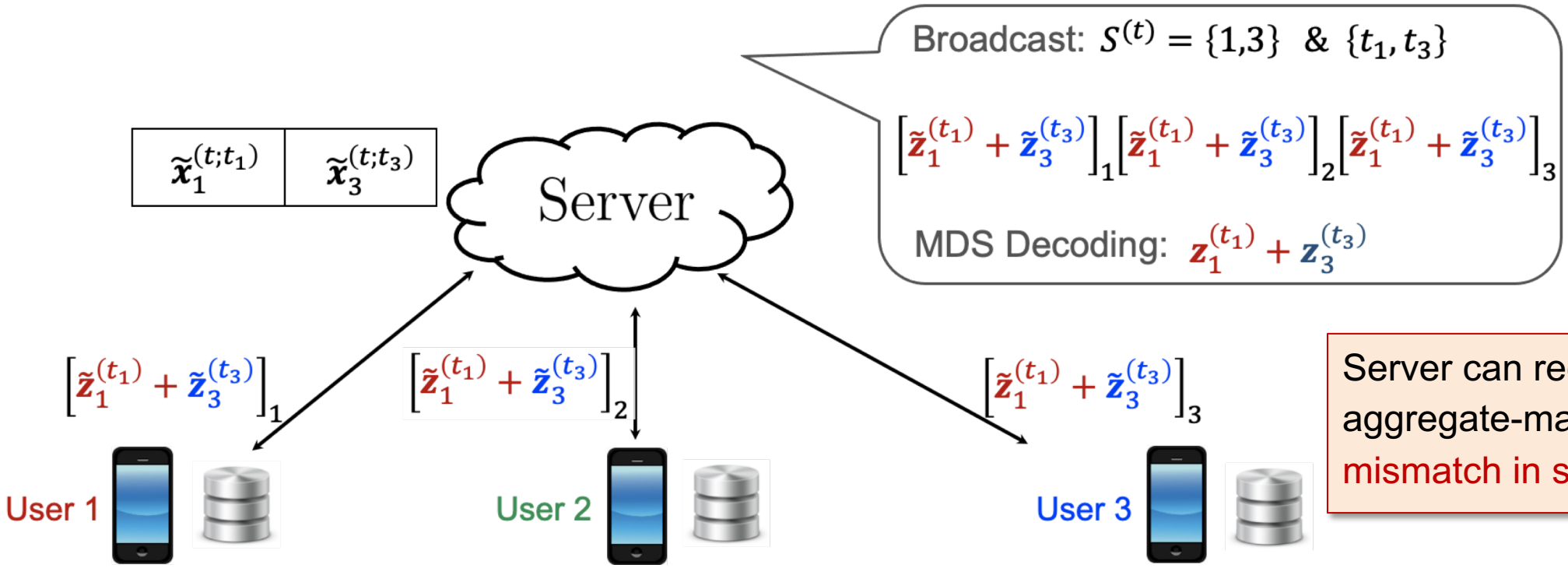


Asynchronous LightSecAgg

Key objective: Design the masks such that they cancel out even if they belong to different training rounds.

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta_g \boldsymbol{g}^{(t)} \text{ where } \boldsymbol{g}^{(t)} = \sum_{i \in S^{(t)}} \tilde{\boldsymbol{x}}_i^{(t;t_i)} = \sum_{i \in S^{(t)}} \overline{\boldsymbol{x}}_i^{(t;t_i)} + \underbrace{\sum_{i \in S^{(t)}} \boldsymbol{z}_i^{(t_i)}}_{\text{Our focus}}$$

Step 3. One-Shot Recovery of Aggregate Masks



Server can reconstruct the aggregate-masks even if there is mismatch in staleness !

Convergence Analysis

- Convergence rate of LightSecAgg and FedBuff are the same except for the increased variance of the local updates due to the quantization noise.

Theorem: *The global model of LightSecAgg achieves the following ergodic convergence rate*

$$\frac{1}{J}\sum_{t=0}^{J-1}\mathbb{E}\left[|\nabla F(\boldsymbol{x}^{(t)})|^2\right] \leq \frac{2F^*}{\eta_g\eta_l EKJ} + \frac{L\eta_g\eta_l\sigma_{c_l}^2}{2} + 3L^2E^2\eta_l^2\left(\eta_g^2K^2\tau_{\max}^2\right)\sigma^2,$$

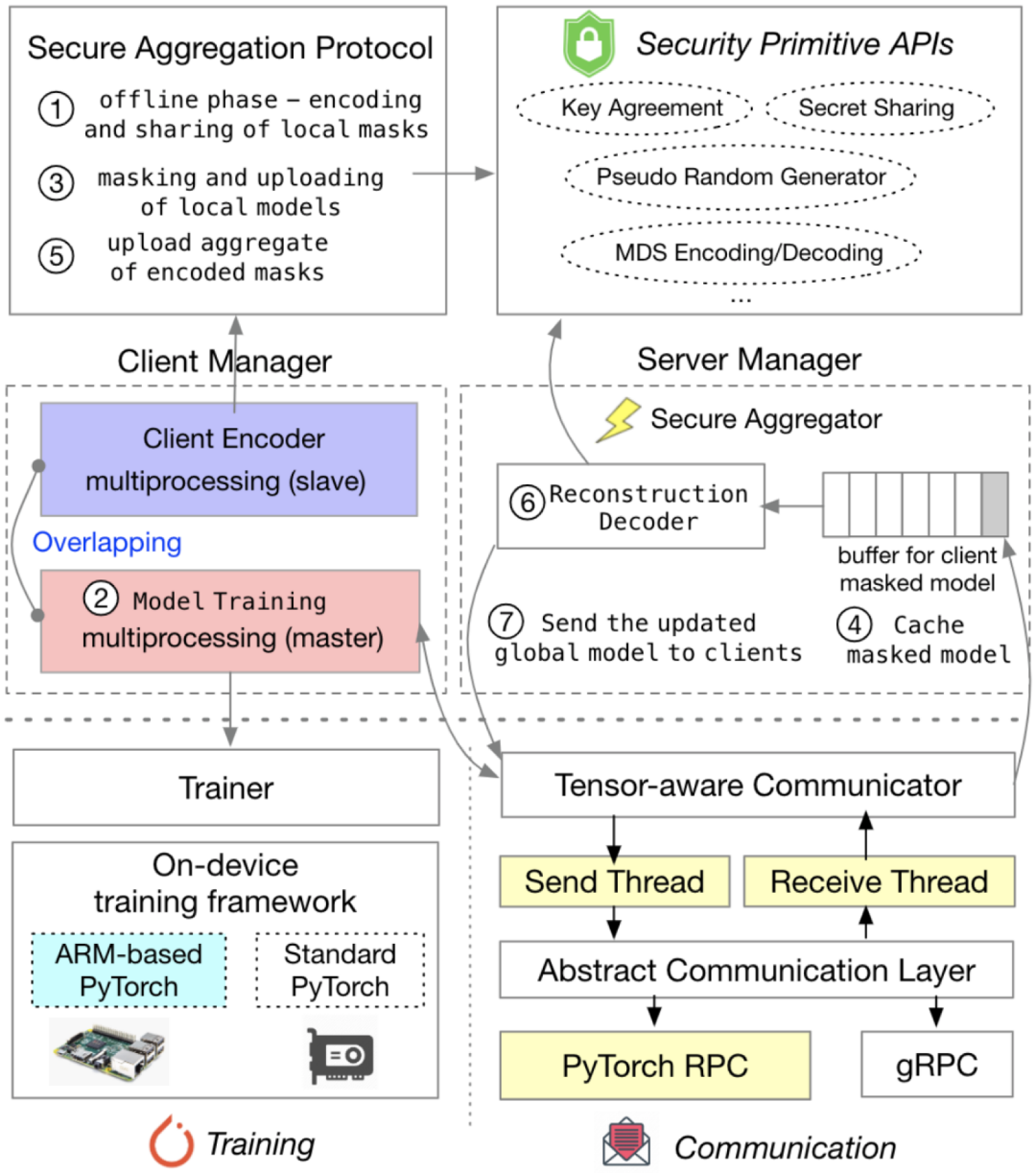
where $F^* = F(\boldsymbol{x}^{(0)}) - F(\boldsymbol{x}^*)$, $\sigma^2 = G + \sigma_g^2 + \sigma_{c_l}^2$, *and* $\sigma_{c_l}^2 = \boxed{\frac{d}{4{c_l}^2}} + \sigma_l^2$.

$\tau_{\max}$: bound on delay
σ_ℓ^2 : bound on local variance
σ_g^2 : bound on global variance
L-smooth gradients

Negligible for large c_l

Part 3. System Design and Experiments

Overview of the System Design



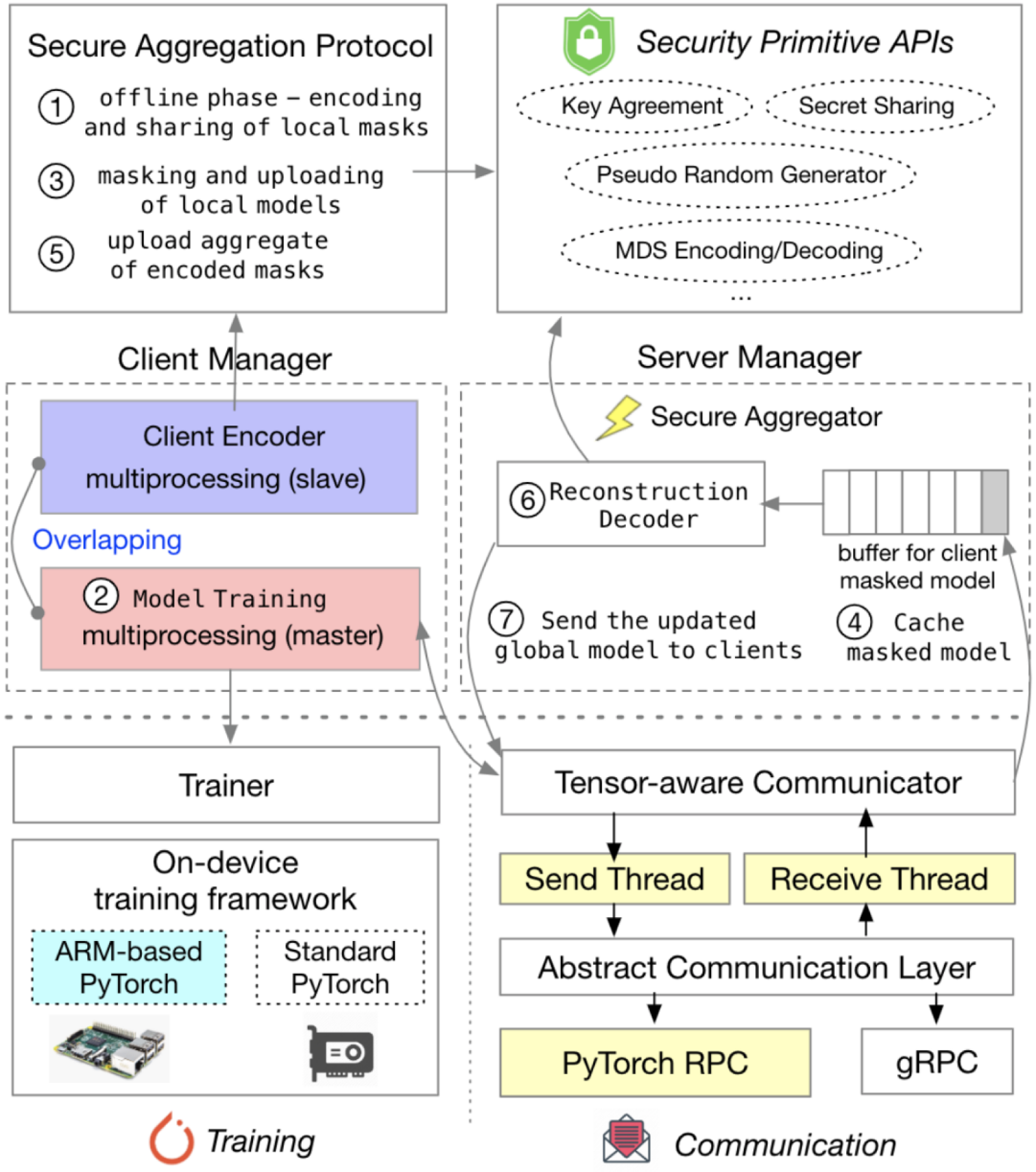
Design Goals:

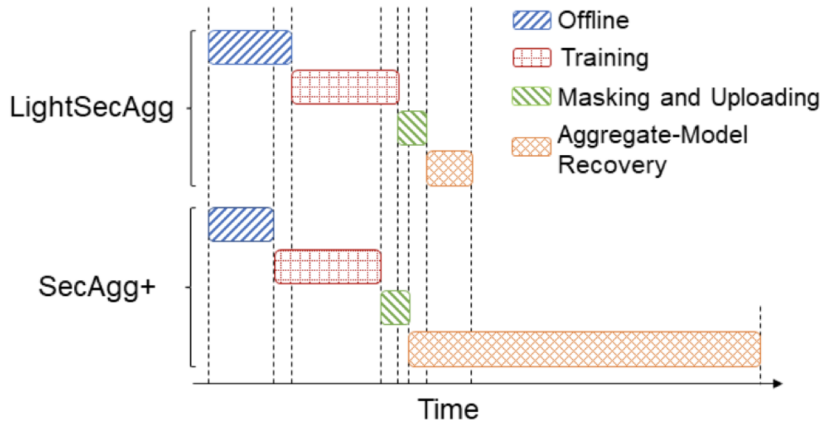
1. Reduce the cost of encoding and decoding at the edge

2. Optimize the communication backend, making it Torch Tensor-aware

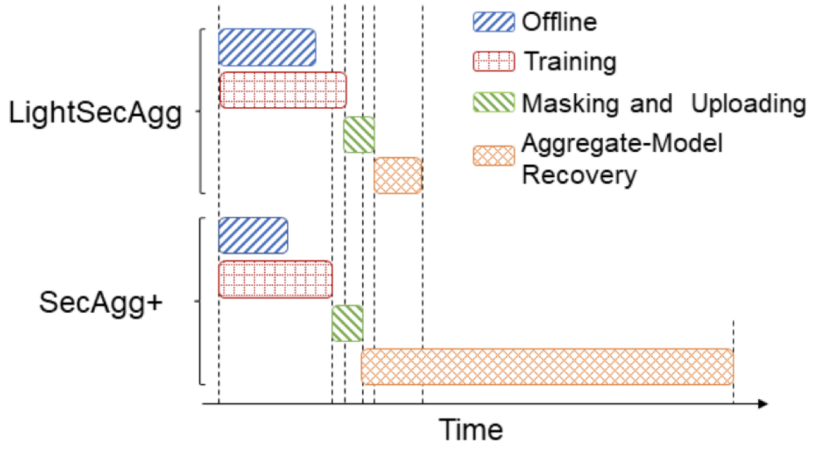
3. Make the system API friendly to pure ML researchers who may not have expertise in SA/Security.

1. Parallelization of offline phase and model training



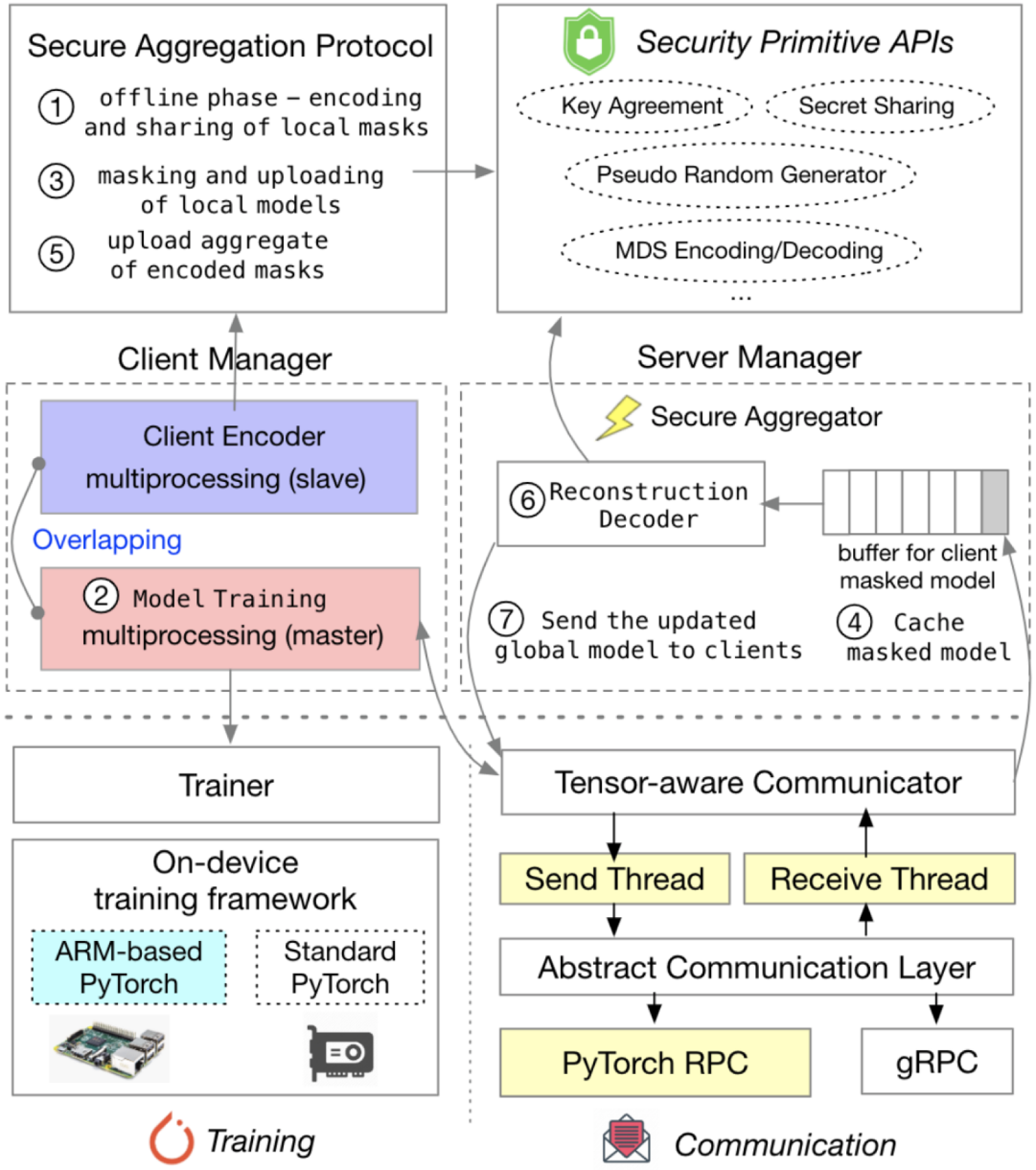


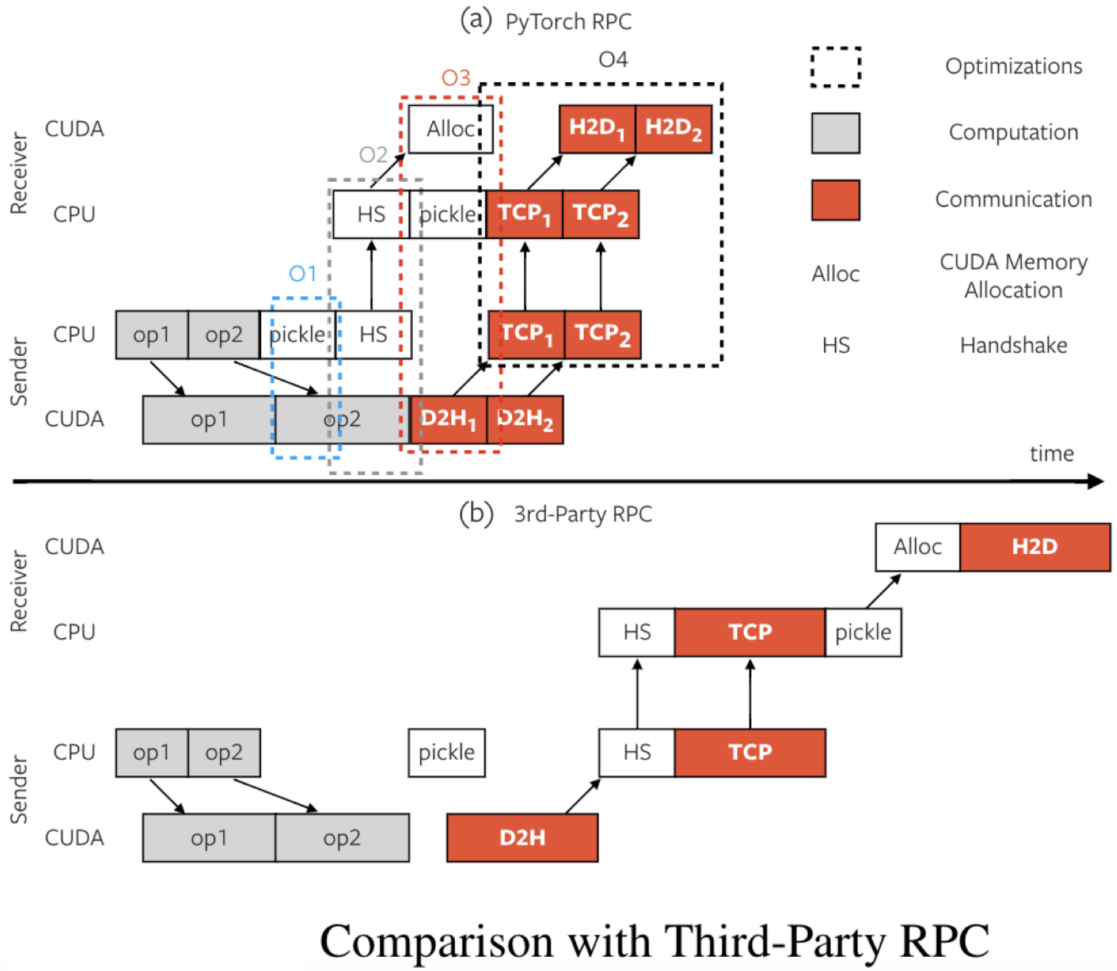
(a) Non-overlapped



(b) Overlapped

2. Tensor-aware RPC (Remote Procedure Call)



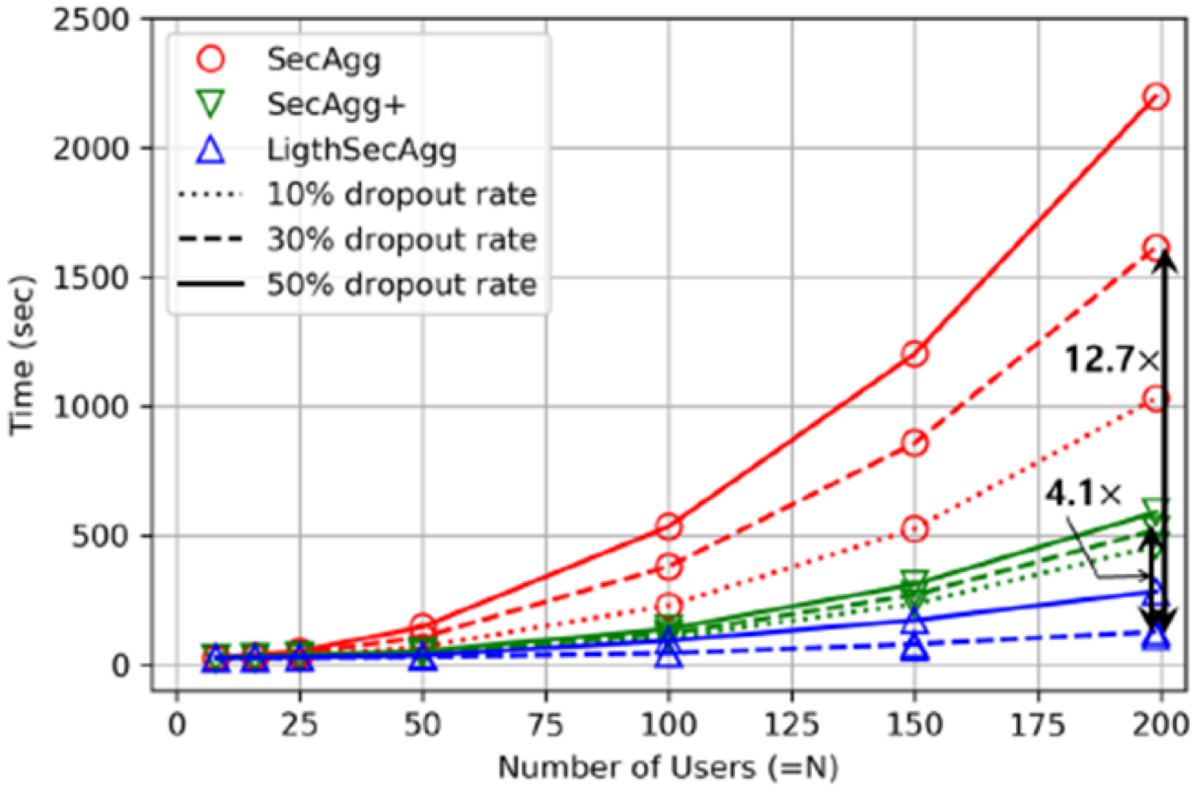


Comparison with Third-Party RPC

Experiments

- Experiment setup:
  - Amazon EC2 cloud using m3.medium machine instances
  - Four different machine learning tasks
  - Communication using the MPI4Py message passing interface on Python
  - Each user drops with a fixed dropout rate p = 0.1, p = 0.3, and p = 0.5

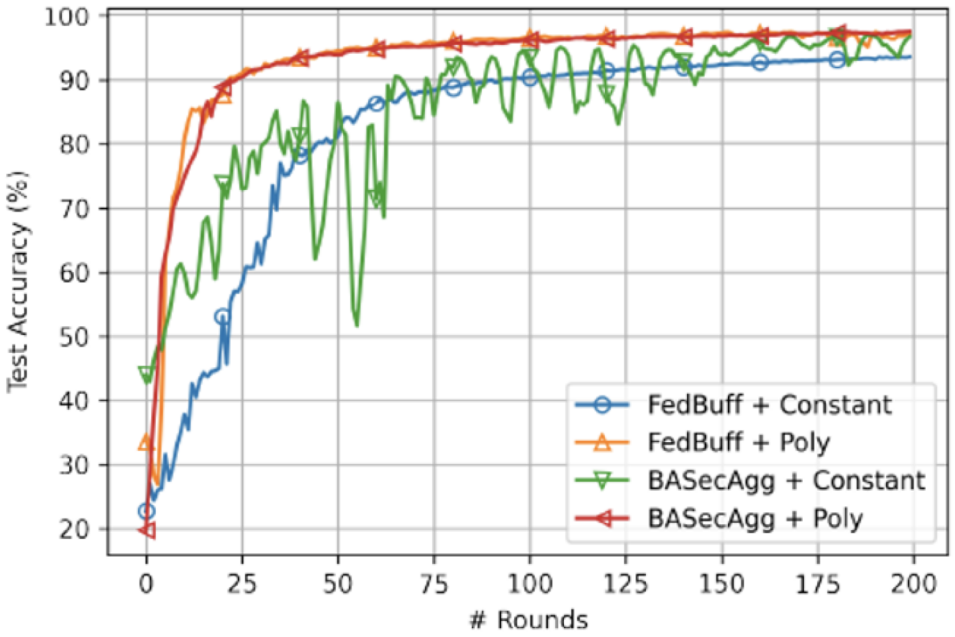
Experiments



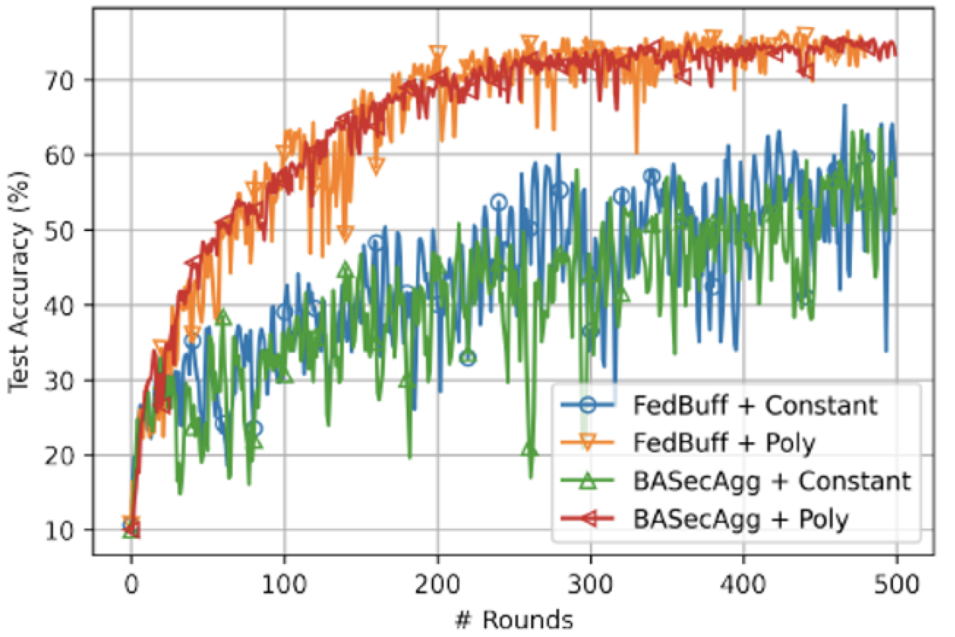
LightSecAgg achieves a performance gain of up to 12.7x !!

Experiments

- LightSecAgg has almost the same performance as FedBuff while LightSecAgg includes quantization noise to for privacy.



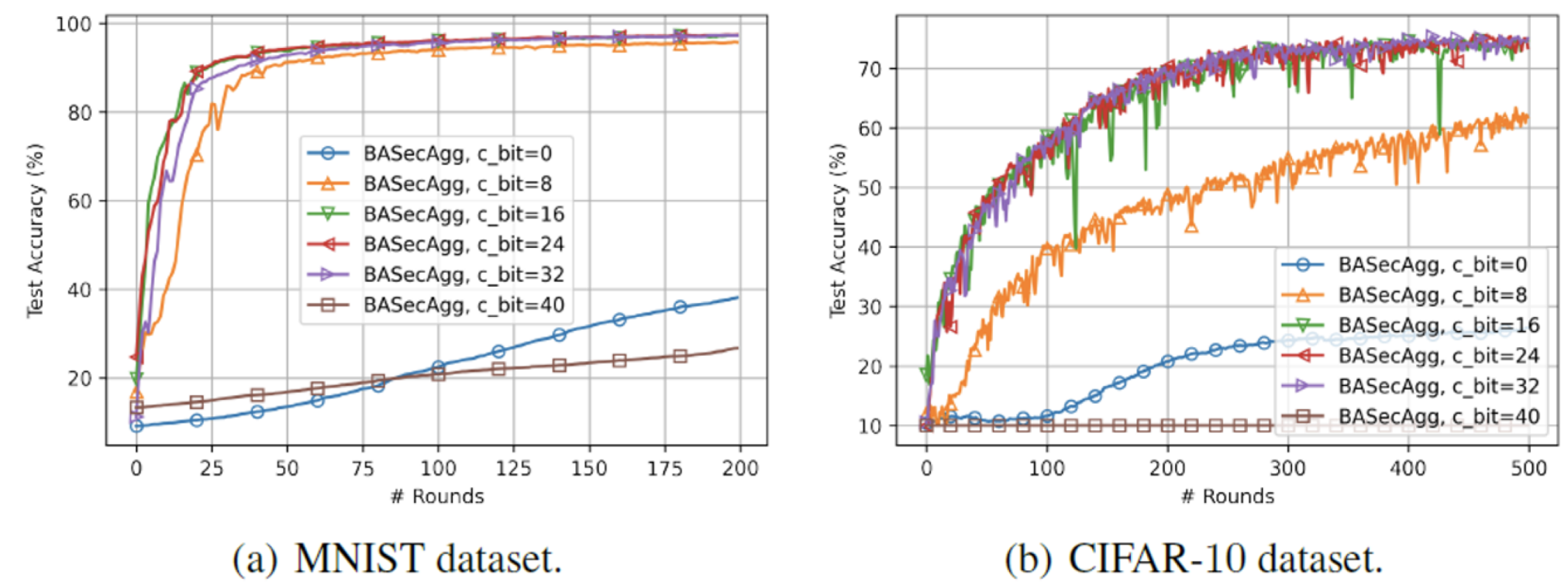
(a) MNIST dataset.



(b) CIFAR-10 dataset.

Experiments

- Effect of the quantization parameter: Trade-off between rounding and wrap-around errors



$$c_l = 2^{c_{bit}}$$

Large c_l results in large wrap-around error, while small c_l results in large rounding error.

Experiments

Table 2: Summary of four implemented machine learning tasks and performance gain of **LightSecAgg** with respect to **SecAgg** [4] and **SecAgg+** [2]. All learning tasks are for image classification. MNIST, FEMNIST and CIFAR-100 are low-resolution datasets, while images in GLD-23K are high resolution, which cost much longer training time for one mini-batch; LR and CNN are shallow models, but MobileNetV3 and EfficientNet-B0 are much larger models, but they are tailored for efficient edge training and inference.

No.	Dataset	Model	Model Size (d)	Gain	
				Non-overlapped	Overlapped
1	MNIST [14]	Logistic Regression	7,850	6.7×, 2.5×	8.0×, 2.9×
2	FEMNIST [5]	CNN [17]	1,206,590	11.3×, 3.7×	12.7×, 4.1×
3	CIFAR-100 [13]	MobileNetV3 [11]	3,111,462	7.6×, 2.8×	9.5×, 3.3×
4	GLD-23K [27]	EfficientNet-B0 [24]	5,288,548	3.3×, 1.6×	3.4×, 1.7×

LightSecAgg can survive and speedup the training of large deep neural network models on high resolution image datasets.

Concluding Remarks

- We propose a new perspective for secure model aggregation in FL, by turning the focus from “pairwise random-seed reconstruction of the dropped users” to “one-shot aggregate-mask reconstruction of the surviving users”.

- We propose LightSecAgg that provides the same level of privacy and dropout-resiliency guarantees as the state-of-the-art while substantially reducing the aggregation complexity.

- In a realistic FL framework, extensive empirical results show that LightSecAgg can provide substantial speedup over baseline protocols for training diverse machine learning models with a performance gain up to 12.7x.