2026 NEW SLIDES FOR COLLABORATIVE ATTACKS

Zizheng Liu, Bharat K. Bhargava

March 11, 2026

PURDUE UNIVERSITY | Department of Computer Science

1

Reasoning and Detecting Collaborative Attacks in Autonomous UAV Networks

Collaborative Attack Definition

Multiple adversarial nodes execute atomic attacks in a specific order.

- To accelerate the attack procedure.
- To hide one adversarial node from detection.
- To amplify the attack impact.

A Motivating Example of Collaborative Attack

Case A: A node launches multiple atomic attacks: identity abuse + DoS attack

Detection Mechanism

Adversarial Node

Identity abuse attack (e.g., impersonation attack, Sybil attack) → Abnormal identity presence

DoS attack (e.g., blackhole attack) → Packet lost on route including the node

A Motivating Example of Collaborative Attack

Case B: The adversary decompose adversarial capability into two nodes

Adversarial Nodes

Create fake identities

Fake identities

DoS attack on behalf of fake identities

*

Packet lost on route including the node

Abnormal identity presence

Detection Mechanism

A Motivating Example of Collaborative Attack

Case B: The adversary decompose adversarial capability into two nodes

Detection Mechanism

Create fake identities

*

Fake identities

DoS attack on behalf of fake identities

Packet lost on route including the node

Abnormal identity presence

Adversarial Nodes

"The adversary increases the stealthiness and persistence of the attack by decoupling different attack capabilities across multiple nodes."

Our Proposed Solution

We propose a framework to detect collaborative attacks in wireless distributed systems.

We want to achieve the following properties:

Data free

- We cannot see all collaborative patterns

Composable

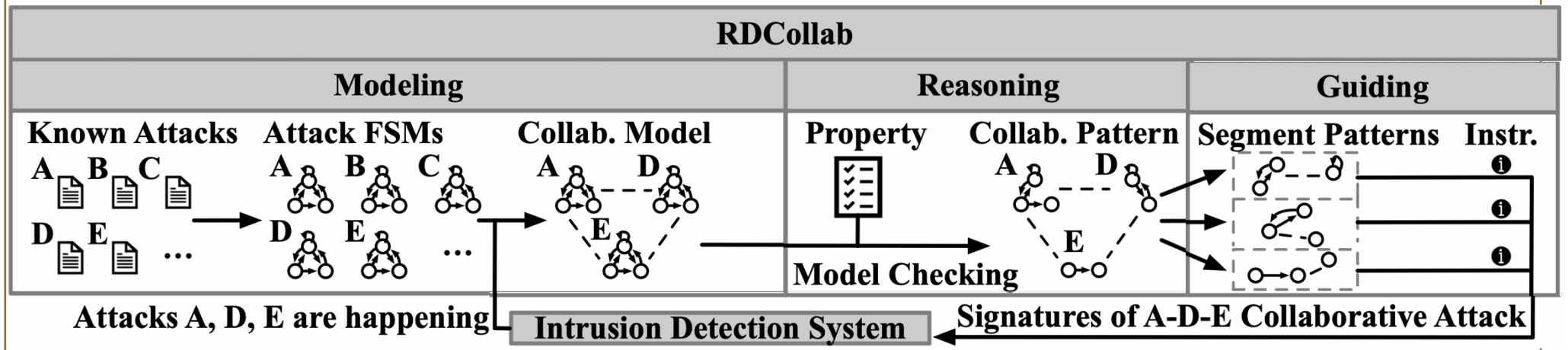
- Collaborative can happen among any atomic attacks, so do the detection

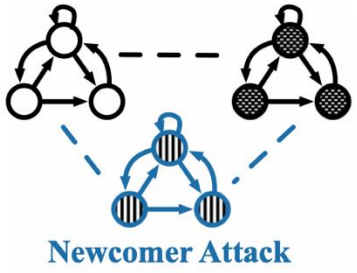
Explainability

- We want to know how collaborative happens to detect it

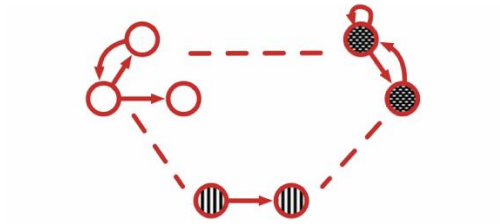
Our Proposed Solution

RDCollab: Reasoning and Detecting Collaborative Attacks in Autonomous UAV Networks





(a) The model is dynamically updated.



(b) The model execution discovered as a counterexample pattern.

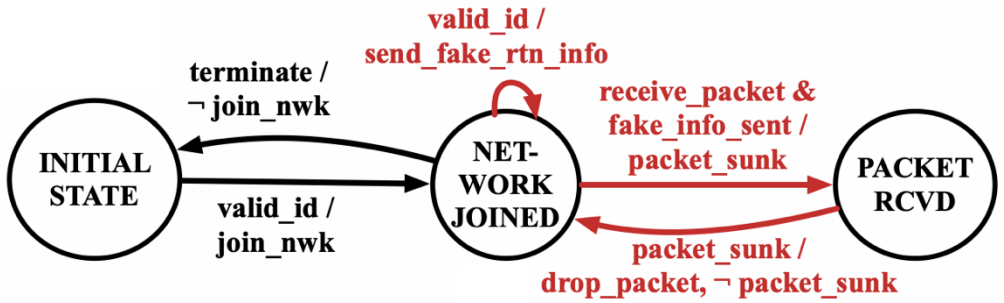
(c) Segment patterns used as collaborative attack signatures to guide the IDS.

RDCollab: The Modeling Phase

The modeling should capture the *causal relationships* among a sequence of actions in an attack process.

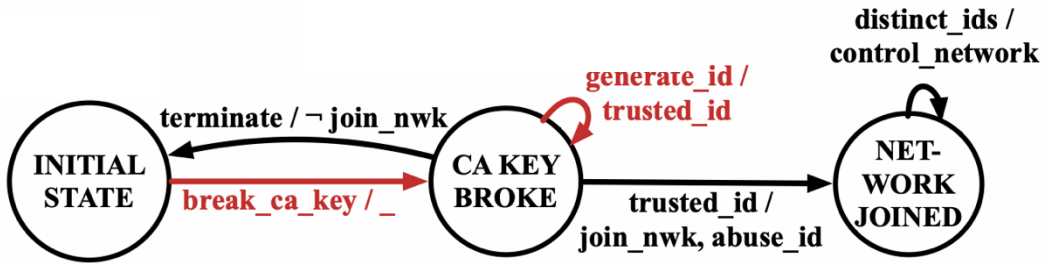
DoS

1. Valid identity
2. Advertise routing info
3. Receive packet
4. Drop packet



Identity Abuse

1. Break certification authority's key
2. Create a valid identity
3. Abuse the fake identity



RDCollab: The Modeling Phase

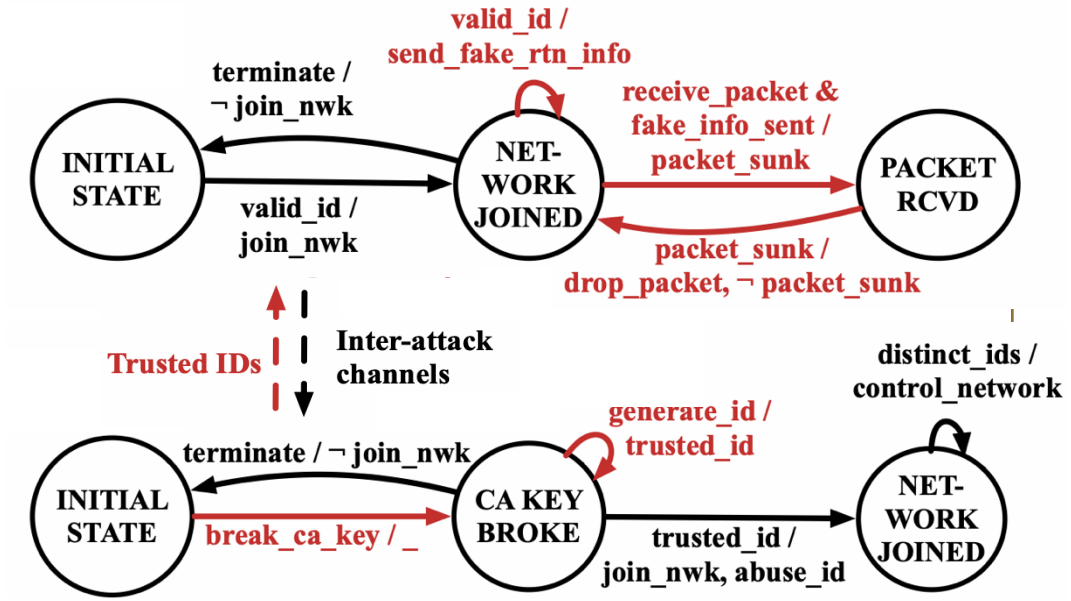
The modeling should capture the *causal relationships* among a sequence of actions in an attack process.

DoS

1. Valid identity
2. Advertise routing info
3. Receive packet
4. Drop packet

Identity Abuse

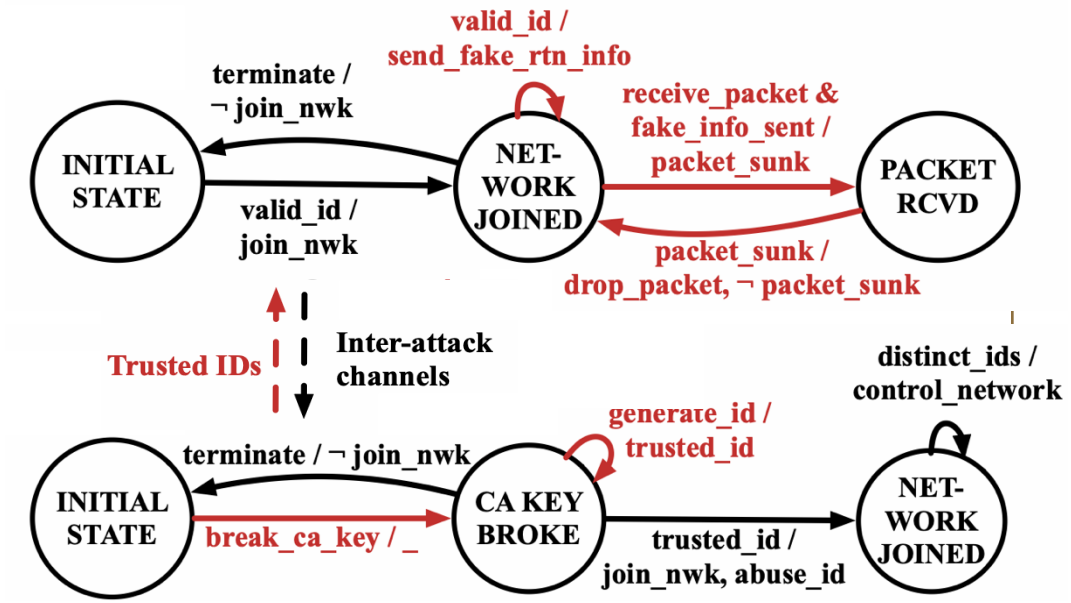
1. Break certification authority's key
2. Create a valid identity
3. Abuse the fake identity



RDCollab: The Modeling Phase

The modeling should use a general language to describe states and transitions in attack FSMs.

1. Privilege (e.g., identity)
2. Impact (e.g., drop packet, abuse identity)
3. Visibility (e.g., advertise routing info, abuse identity)



RDCollab: The Reasoning Phase

Model Checking

Describing the executions we want or to avoid.

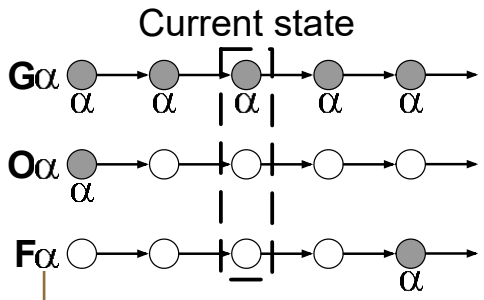
"The collaboration patterns that cause synergy effects do not exist in the model execution"

"Node A may transfer `privilege` to node B, allowing B to cause negative `impacts`."

Linear Temporal Logic (LTL) Property

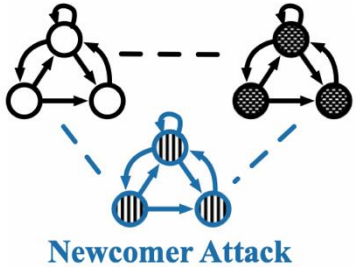
$$G(imp \rightarrow O(priv_1 \ \& \ priv_2 \ ...))$$

- G(α): α holds at all times (globally)
- O(α): α held in at least one of the previous time steps (once)
- F(α): α will hold in some time in the future (eventually)



RDCollab: The Reasoning Phase

Model Checking



"G(imp → O($priv_1$ & $priv_2$ ...))"

Abstracted System Model

Property

Model Checker

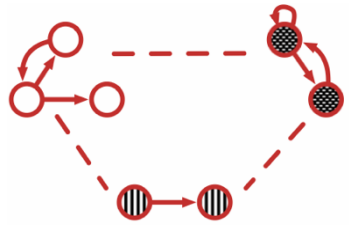
Hold?

Yes

True

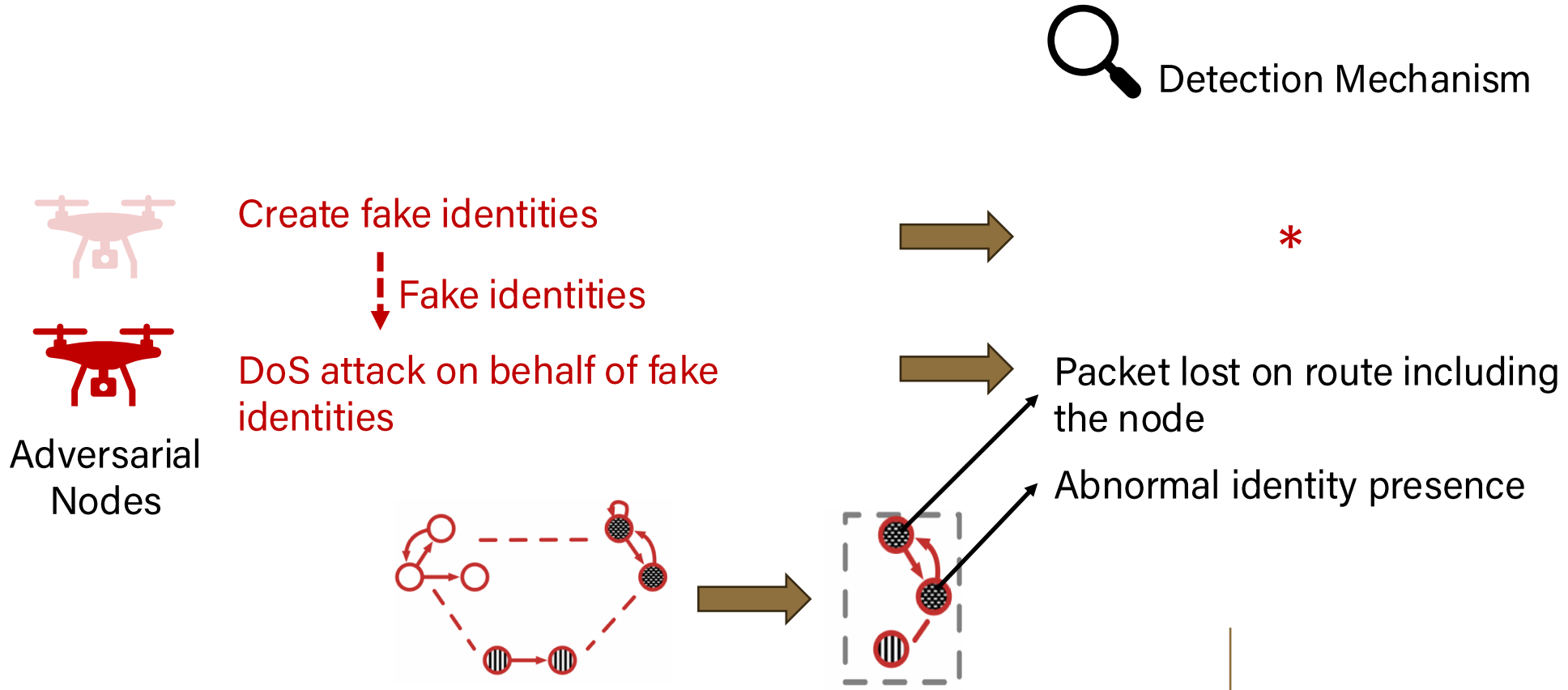
No

Counterexample



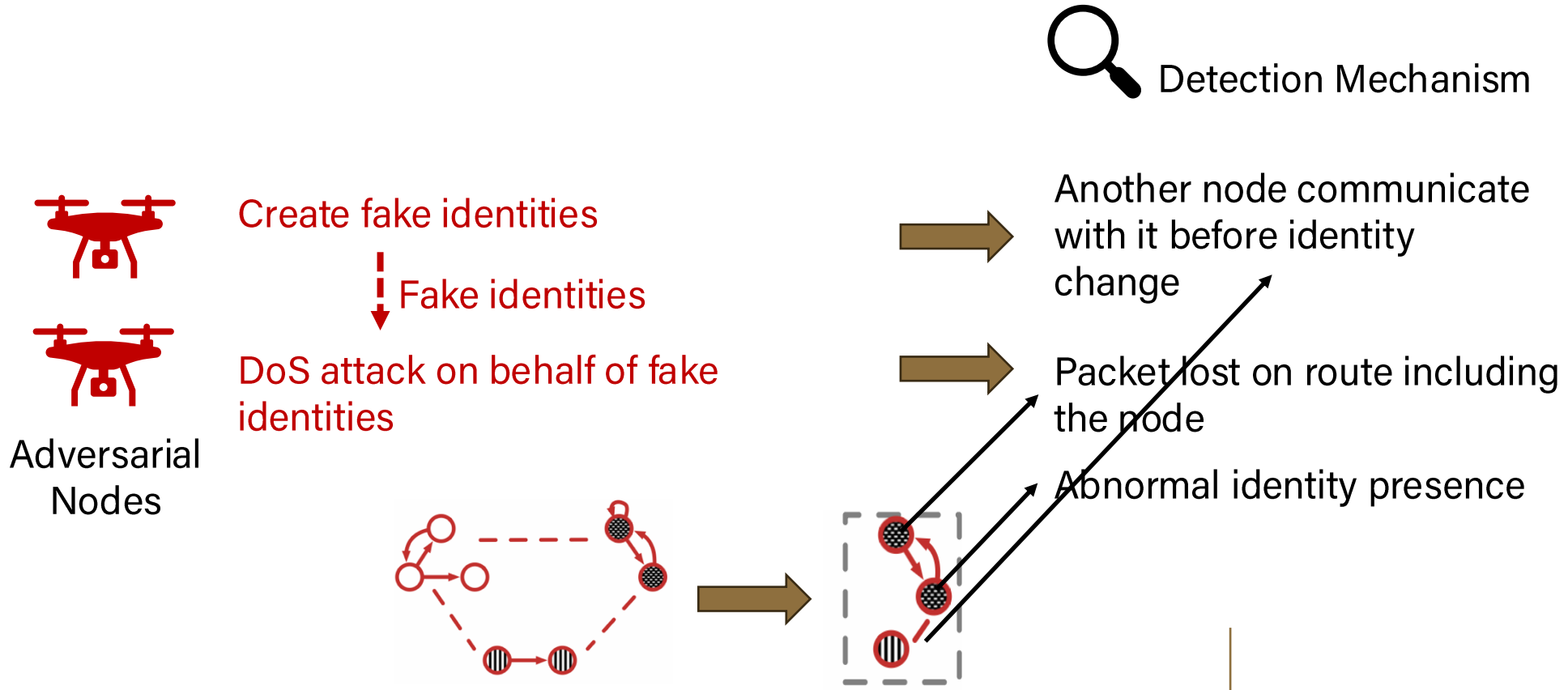
RDCollab: The Guiding Phase

The counterexamples give explainable sequence of coordinated events.



RDCollab: The Guiding Phase

The counterexamples give explainable sequence of coordinated events.



RDCollab Implementation

Atomic attacks and properties covered

- DoS, identity abuse, and tunneling attacks

GP-I: “Node A may transfer `privilege` to node B, allowing B to cause negative `impacts`.”

GP-II: “A node does not perform any `visible` action in order to cause negative `impacts` (given the atomic attack is detectable).”

GP-III: “Multiple nodes can cause negative `impacts` with only one node satisfying the condition need in the atomic attack causing that impact.”

RDCollab Reasoning Phase Evaluation

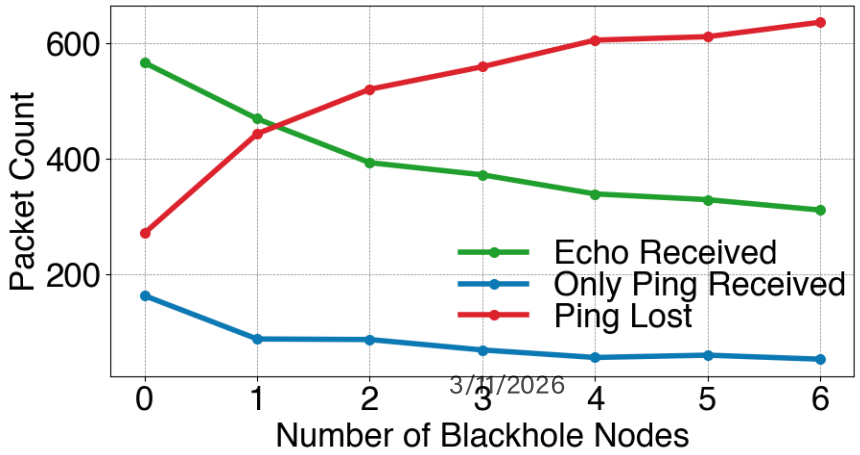
Attack Scenario	Properties Violated / Generated		
	GP-I	GP-II	GP-III
DoS-tunnel	0/6	5/5	11/30
DoS-ID abuse	3/3	2/4	0/6
tunnel-ID abuse	3/3	2/4	0/5
DoS-tunnel-ID abuse	3/6	7/7	14/34
Total	**50/113**		

Collaborative Attack	Main Synergy Effects	DoS	Tunneling	ID Abuse	Notable Collaborative Pattern
Hidden Blackhole Attack	Hiding Effect	Yes	No	Yes	Secret (i.e., IDs) exchange
Hidden Wormhole Attack	Hiding Effect	No	Yes	Yes	Secret exchange
Duplicated Routing Disturbance Attack	Amplifying Effect	Yes	Yes	No	Attack damage duplication
Distributed Data Exfiltration Attack	Shortcut Effect	Yes	Yes	No	Traffic transfer
Distributed Sinkhole Attack	Shortcut Effect	Yes	Yes	No	Traffic transfer
Hidden Distributed Data Exfiltration Attack	Hiding Effect, Shortcut Effect	Yes	Yes	Yes	Secret exchange, Traffic transfer
Hidden Distributed Sinkhole Attack	Hiding Effect, Shortcut Effect	Yes	Yes	Yes	Secret exchange, Traffic transfer

RDCollab Experiment Setting

FANET simulation settings

- 20 UAVs: one ID abuse attacker, one to six DoS attackers, others are legitimate nodes;
- Fly randomly in 100 m * 100 m * 100 m 3D space, following the Gauss Markov Mobility Model;
- WiFi 802.11n for PHY and MAC layer, AODV for network layer routing;
- Ping + echo between two randomly selected legitimate nodes every second;
- 1,000 simulation seconds for each run.



RDCollab Detection Phase Evaluation

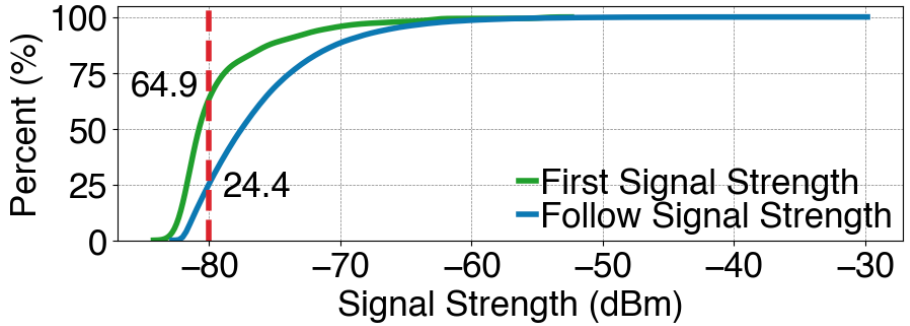
Synergy effects caused by the interaction between the DoS and the ID abuse attack

1. Invalidating the blacklisting-based blackhole attacker defense mechanisms;
2. Worsening the routing integrity due to faulty routing entries caused by frequently changing IDs;
3. Worsening the routing availability due to ‘ghost routing entries’;
4. Hiding the *real* ID abuse attacker.

RDCollab Detection Phase Evaluation

We highlight how RDCollab guides *existing* detection mechanisms to collaborate to detect attack interactions.

- DoS detection[1]: detects RREPs w/ high seq. no. and low hop cnt., and packet transmission failures.
- ID abuse detection[2]: detects the packet from first-seen neighbors with high signal strength.

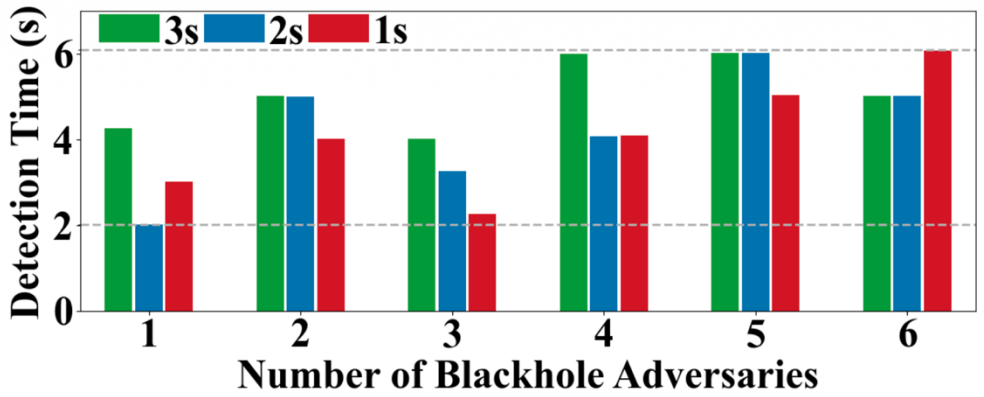


1. Faraji-Biregani et al. *Secure Communication between UAVs Using a Method based on Smart Agents in Unmanned Aerial Vehicles*
2. Abbas et al. *Lightweight Sybil Attack Detection in MANETS*

RDCollab Detection Phase Evaluation

Detecting the *real* ID abuse node

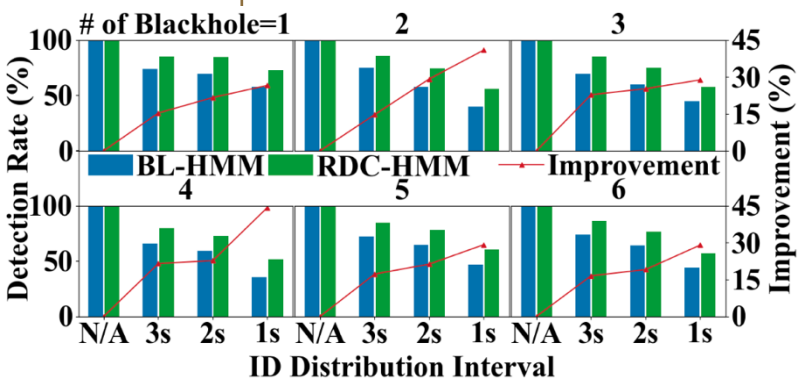
- The ID abuse node transfers IDs to DoS nodes every, two or three seconds;
- Extracts correlation between inter-node communication and abnormal events.

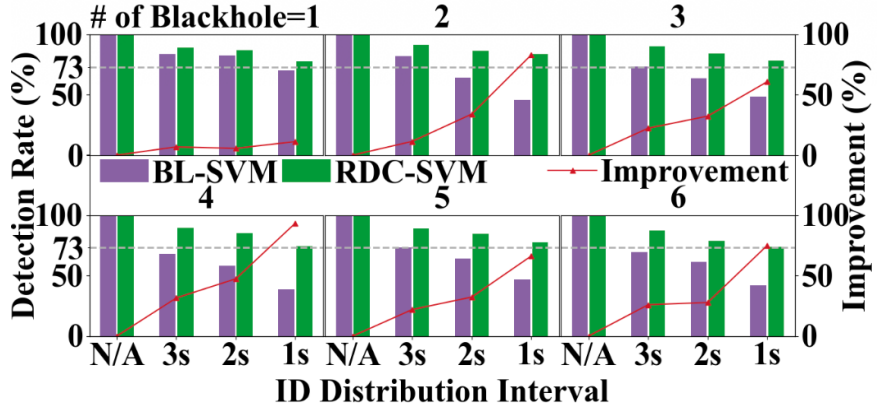


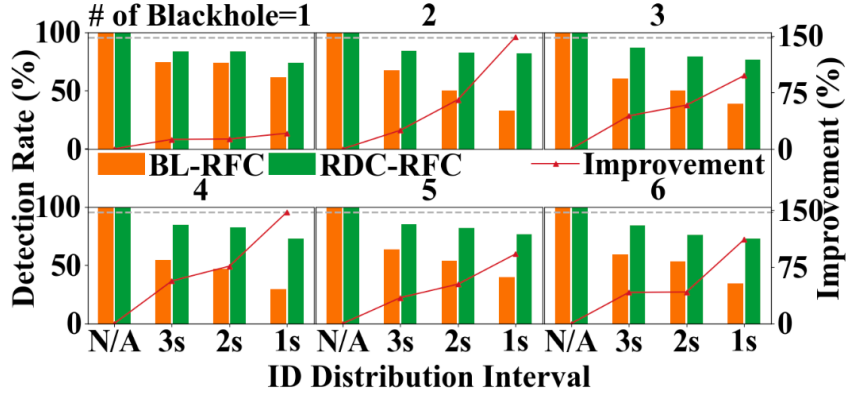
RDCollab Detection Phase Evaluation

Improving the affected detection mechanisms

- We assume the real ID abuse node is not removed from FANET after being detected and keeps transferring IDs;
- We implement lightweight ML-based DoS attack detectors using RREPs w/ high seq. no. and low hop cnt., and packet transmission failures as observations, trained in blackhole attack only scenarios; detection rate = $\frac{ID\ detected}{ID\ abused} * 100\%$





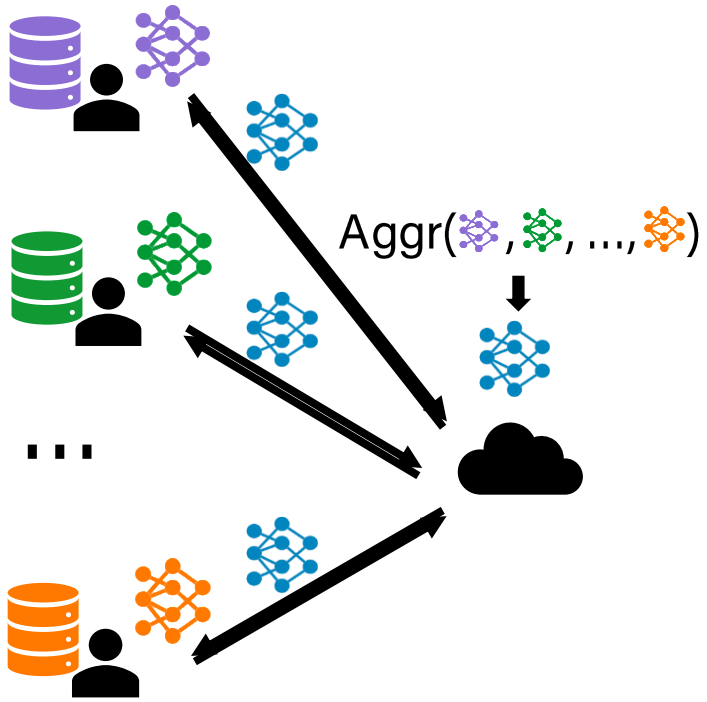


2

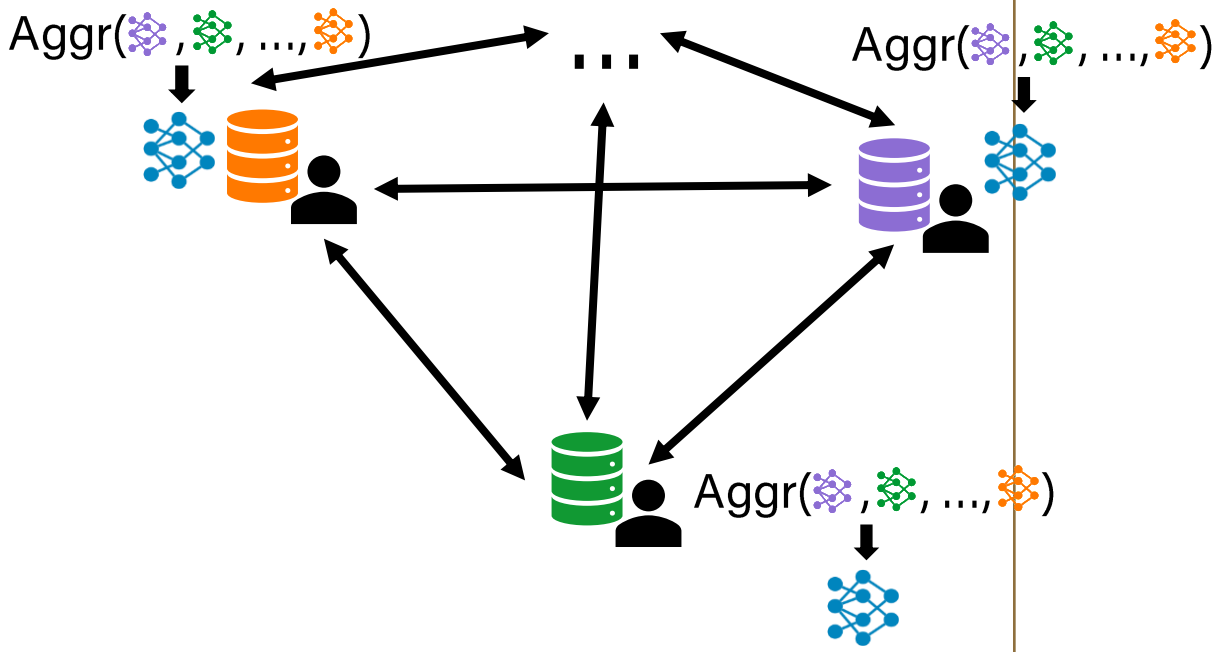
Federated Learning Against Collaborative Model Inference Attacks

Attacks Against Federated Learning

Centralized FL (e.g., Google and clients)



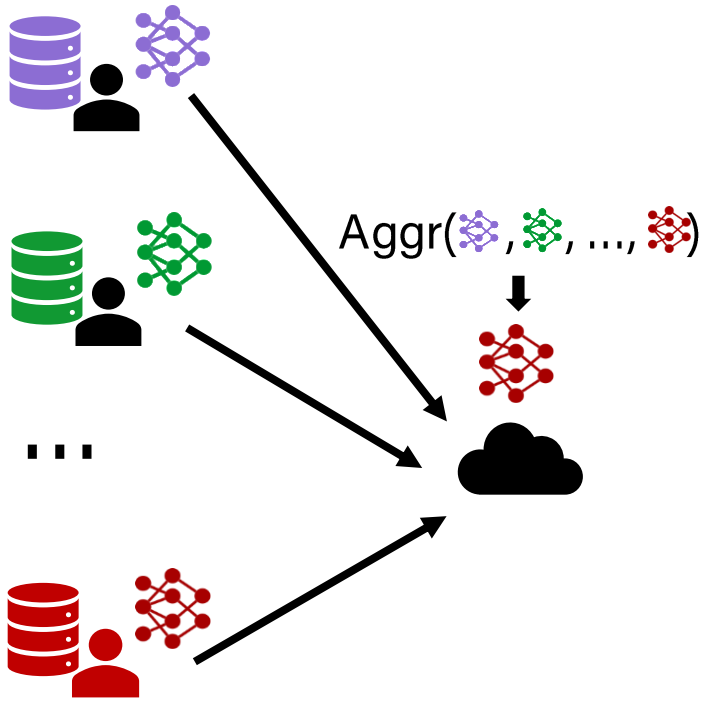
Decentralized FL (e.g., an autonomous UAV network)



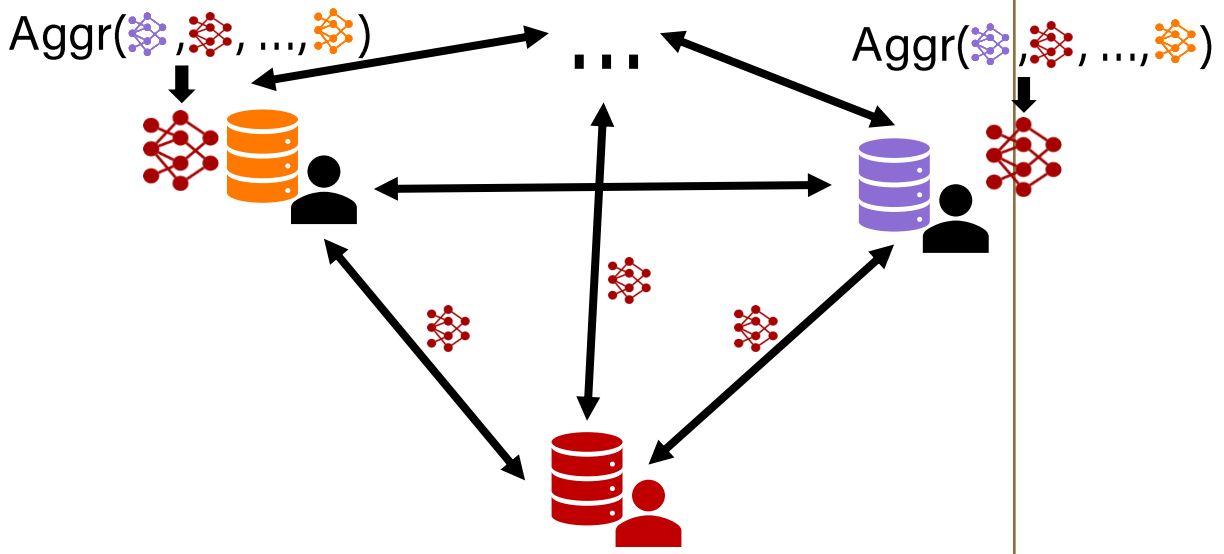
Local data: D_1, D_2,...,D_n
Local models: W_1^r, W_2^r,...,W_n^r
Aggr(W_1^r, W_2^r,...,W_n^r) = G^r
Global model: G^r

Attacks Against Federated Learning

Centralized FL (e.g., Google and clients)



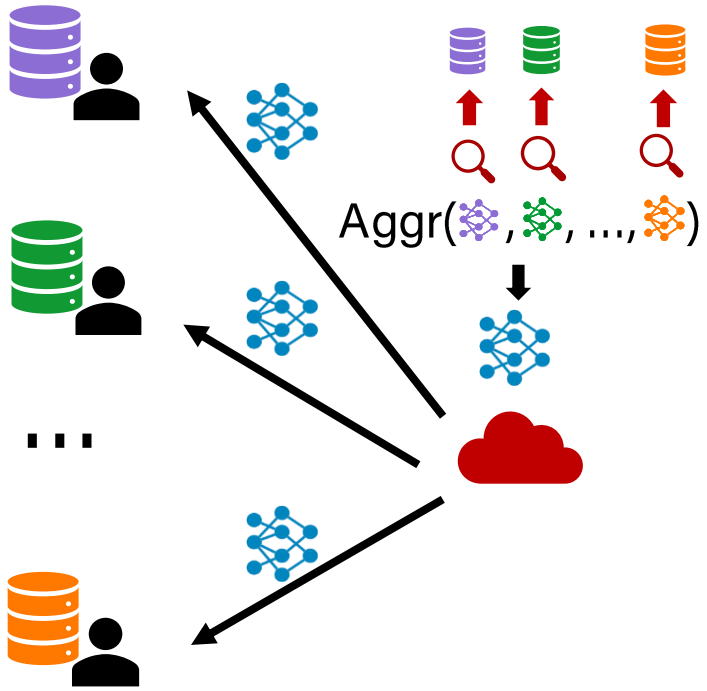
Decentralized FL (e.g., an autonomous UAV network)



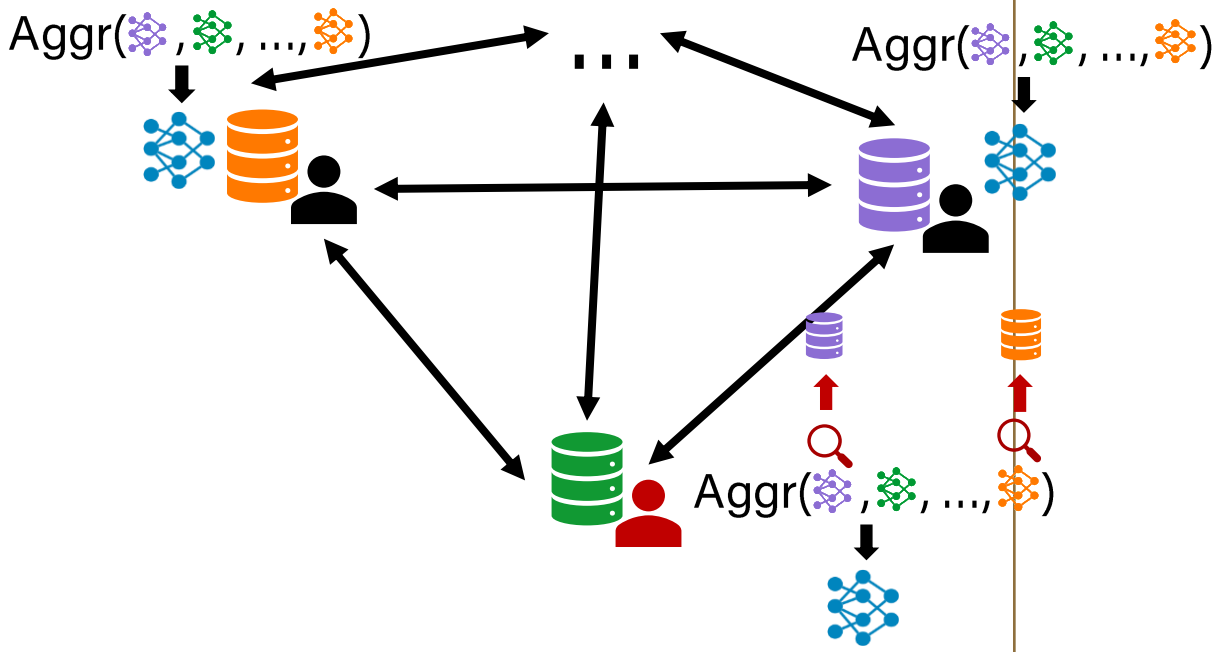
/ : poisoned data / model

Attacks Against Federated Learning

Centralized FL (e.g., Google and clients)



Decentralized FL (e.g., an autonomous UAV network)



: privacy leakage attack

State-of-the-Art Defense

Robust FL

Verify(W_1, W_2, ..., W_n) = W_1, W_2,...
Aggr(W_1, W_2, ...) = G

An example of Verify():
Verify($W_1, W_2, ..., W_n$) = {Valid(W_i) · W_i} for i ∈ [n] where Valid can be

- Norm[3] Valid(W) = $I[\|W\|_2 < \rho]$
- Norm Ball[4] Valid(W) = $I[\|W - U\|_2 \leq \rho]$
- Zeno++[5] Valid(W) = $I[\gamma \langle W, U\rangle - \rho\|W\|_2 \geq -\gamma\epsilon]$
- Cosine Similarity[6] Valid(W) = $I[\langle W,U\rangle / (\|W\|_2 \|U\|_2) < \rho]$
- ...

State-of-the-Art Defense

Robust FL

Verify(W_1, W_2, ..., W_n) = W_1, W_2,...
Aggr(W_1, W_2, ...) = G

f := Aggr(Verify());
x_i := W_i, y_i := G;
Compute $f(x_1, ..., x_n)=y_1, ..., y_n$ with MPC:
Aggr(Verify($W_1, W_{2, ..., } W_n$)) = G

Parties may cheat during the MPC procedure to stop it from outputting the global model.
Our work bridge this gap.

Privacy Preserving FL

- Differential Privacy (DP)[7]:

$$\forall S, \forall D \sim D' : \Pr[M(D) \in S] \leq e^{\varepsilon} \cdot \Pr[M(D') \in S] + \delta$$

  - DP provides (ε, δ)-privacy.
  - The leakage bound is usually very loose ($\varepsilon \geq 8$) to preserve utility in practice[8].
- Secure Multiparty Computation (MPC)[9]:
  - n parties with private input $x_1, ..., x_n$ compute $f(x_1, ..., x_n)=y_1, ..., y_n$ where x_i and y_i are only revealed to party i, f is a public function.
  - MPC provides zero-knowledge privacy for local models.

We Bridge the Gap Between MPC w/ GOD and Federated Learning

FL Layer

- FL robustness: Aggr(Verify())[1-6]
- FL privacy and robustness: run Aggr(Verify()) with MPC[9,11-13] (secure aggregation)

MPC Layer

Adversary can misbehave to stop the protocols from delivering the output.

- Semi-honest MPC: only provides privacy and correctness if all parties follow the protocol (passive / honest but curious adversaries).
- Malicious MPC (active / malicious adversaries)
  - Secure with abort: provides privacy even if parties deviate from the protocol, abort the protocol upon deviation detected. But does not know which party misbehaved.
  - Guaranteed output delivery (GOD): identifies misbehaved parties, eliminate them, remaining parties continue the computation, repeat until outputs are delivered. (SOTA GOD MPC only support addition and multiplication between integers)

We designed the first secure aggregation protocols with guaranteed output delivery.

- We designed protocols to support secure computation primitives beyond addition and multiplication.
- We designed protocols to support fixed point arithmetic.

We implemented and evaluated the framework in local/wide-area distributed nodes.

We Bridge the Gap Between MPC w/ GOD and Federated Learning

FL Layer

It is easy to detect who is misbehaving since everything are shared in plaintext.

MPC Layer

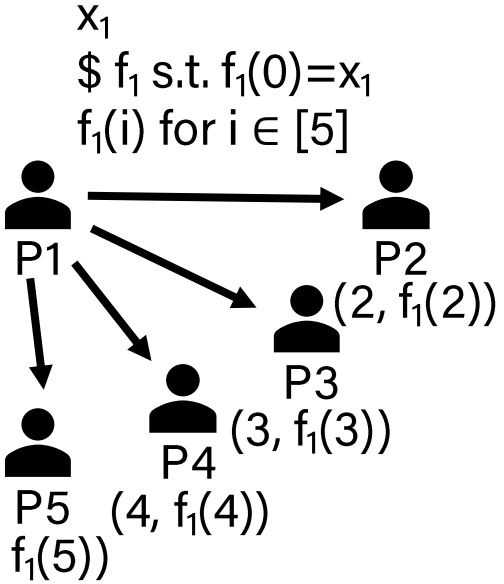
Adversary can misbehave to stop the protocols from delivering the output.

Since MPC provides zero-knowledge privacy, all the values exchanged between parties look *uniformly random*, making it hard to detect who is sending malformed values.

MPC Building Block: Shamir's Secret Sharing

$f_1(t) = x_1 + a_1 t + b_1 t^2$
$f_2(t) = x_2 + a_2 t + b_2 t^2$
...
$F(t) = \sum x_{i\in[5]} + \sum a_{i\in[5]}\, t + \sum b_{i\in[5]}\, t^2$

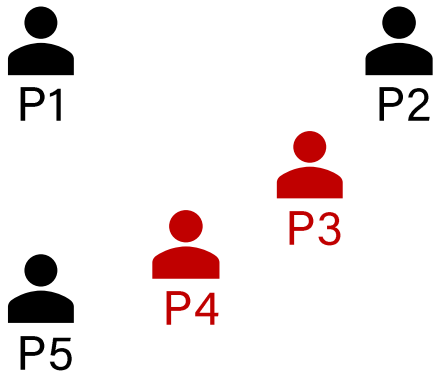
Shamir's (5,2)-secret sharing example.



1. Party 1 has secret input x_1.
2. It generates a degree-2 polynomial f_1 such that $f_1(0)=x_1$.
3. It evaluates f_1 on $i \in \{1,2,3,4,5\}$ and sends $f_1(i)$ to party i.
4. Since f_1 has degree 2, any 2+1=3 shares can reconstruct f_1, thus reveal the secret $f_1(0) = x_1$.
   - For example, P2,4,5 collaborate and know that f_1 is a degree-2 polynomial with points $\{(2, f_1(2)), (4, f_1(4)), (5, f_1(5))\}$. They can reconstruct f_1, and compute $f_1(0)=x_1$.
5. However, any 2 parties know nothing about the secret.
6. Other parties (P2~P5) shares their secrets ($x_2 \sim x_5$) in the same way.
7. Then party Pj has $\{(j, f_i(j))\}_{i\in[5]}$. For example, P3 has $\{(3, f_1(3)), (3, f_2(3)), (3, f_3(3)), (3, f_4(3)), (3, f_5(3))\}$. It can compute $F(3) := \sum_{i\in[5]} f_i(3)$, where F is a degree-2 polynomial such that $F(0) = X := \sum_{i\in[5]} x_i$.
8. Now, any 3 parties can collaborate to reconstruct F (thus, X) while any 2 parties know nothing about X.

MPC Building Block: Shamir's Secret Sharing

Shamir's (5,2)-secret sharing example.



During this procedure, x_i are hidden to parties as long as no more than 2 parties collaborate to reconstruct the secret.
Now define a protocol $\Pi_{5,2}$ using Shamir's (5,2)-SS that

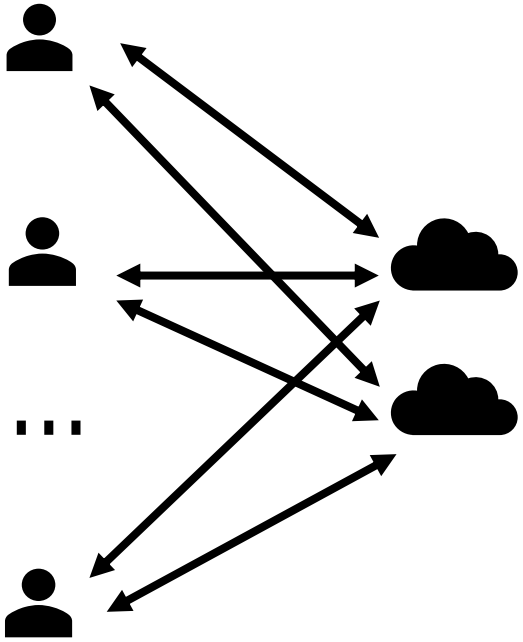
- Requires honest parties not to collaborate to reconstruct x_i, but they are required to collaborate to reconstruct $X = \sum_{i\in[5]} x_i$ after receiving all shares.
- Malicious parties may collaborate to reconstruct x_i.

This protocol protects the privacy of x_i if number of malicious parties is no more than 2.

Now define the Aggr() function for FL as FedAvg[10], namely $G = (\sum_{i\in[n]} W_i)/n$. And parties use Shamir's (n,t)-secret sharing where $n \geq 2t+1$. The protocol $\Pi_{n,t}$ allows parties to aggregate all local models W_i into the global model G without revealing any local model's privacy as long as the number of malicious parties is no more than t.
The technique applying MPC to aggregate FL models is known as secure aggregation.

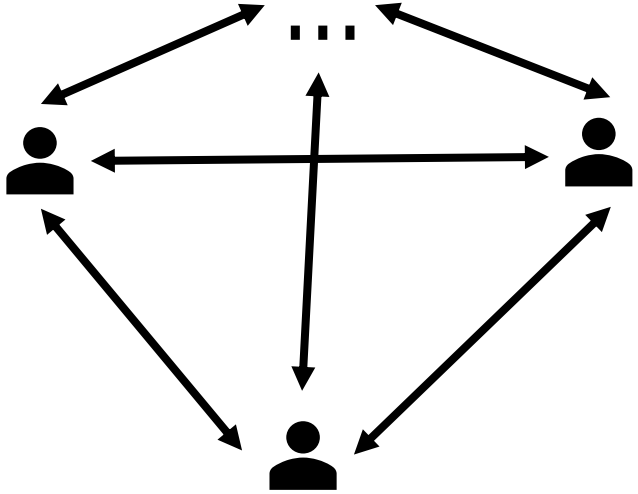
Some Scenarios of Secure Aggregation

Conventional FL



Split the server into two untrusted, non-colluding servers. Clients use Shamir's (2,1)-secret sharing to share local models. Two servers run the secure aggregation protocol to obtain the global model and send to clients.

Decentralized FL



In a distributed system with n total parties, t ($n \geq 2t+1$, i.e., honest majority) of them are malicious and colluding, parties use Shamir's (n,t)-secret sharing to share local models, any t+1 parties can run the secure aggregation, obtain the global model and broadcast to all parties.

The DN Multiplication Protocol

The DN multiplication Protocol[14]

All parties have $([r]_t, [r]_{2t})^*$, parties select a special party P_{king} as a coordinator.

1. All parties first locally compute $[e]_{2t} := [x]_t \cdot [y]_t + [r]_{2t}$.
2. P_{king} collects all shares of $[e]_{2t}$ and reconstructs the secret e (recall $n \geq 2t+1$). Then P_{king} generates a degree-t sharing $[e]_t$ and distributes the shares to all other parties.
3. All parties locally compute $[x \cdot y]_t = [e]_t - [r]_t$.

* r is a random value, $[r]_t$ is a degree-t sharing of it and $[r]_{2t}$ is a degree-2t sharing. All parties run a protocol DoubleRand, each party obtains a share of $[r]_t$ and $[r]_{2t}$, but no one learns r. Randomness generation protocols similar to DoubleRand are used in protocols introduced later.

Compared with addition, which can be computed locally by each party, multiplication requires extra random value shares, more communication rounds and more computations.

Verifying the Protocol Correctness

The above protocols are semi-honest secure, not even secure with abort, we need to verify their correctness.

1. For m executions of a protocol, we denote their *transcripts* as
$$\{([x_i], [y_i], [r_i], [r'_i], [a_i], [b_i])\}^m_{i=1}.$$

2. Parties use randomness generation protocol to generate sharings of random values $[x_0]$ and $[y_0]$, run the protocol on random input and get transcript
$$([x_0], [y_0], [r_0], [r'_0], [a_0], [b_0]).$$

3. Parties run a protocol to generate a public random value λ, combine the transcripts as
$$([x], [y], [r], [r'], [a], [b]) = \Sigma_{i\in\{0,\ldots,m\}} \lambda^i([x_i], [y_i], [r_i], [r'_i], [a_i], [b_i]).$$

4. Parties broadcasts their shares of $([x], [y], [r], [r'], [a], [b])$ and check consistency.

Guaranteed Output Delivery

High-level idea: parties maintain two public sets *Disp* and *Devi* to record parties in dispute and parties found deviated from the protocol.

Disp records indices of pairs of dispute parties, for example, Disp = {(1,2)} means parties P_1 and P_2 are in dispute on shares used in the computation.
Devi records indices of parties found deviated from the protocol, for example Devi = {3} means party P_3 is found deviated from the protocol.

- Parties in Devi are corrupted, but parties in Disp may not be corrupted.
- At least one party recorded in each pair in Disp is corrupted.
- If a party is in dispute with t+1 parties, it is considered corrupted and its index is put into Devi.
- A party found deviated is considered in dispute with all parties.

Support Guaranteed Output Delivery

Add authentication tag for each share used for computation.

Assume $[z]_t = [x]_t + [y]_t$ caused the inconsistency, where shares of $[x]_t$ are distributed by P_{dx} and $[y]_t$ are distributed by P_{dy}.

Rx\Dist	P_1	…	P_{dx}	P_{dy}	…	P_n	
P_1	0	…	$[x]_t^{(1)}$	$[y]_t^{(1)}$	…	0	$[z]_t^{(1)}$
…	…	…	…	…	…	…	…
P_i	0	…	$[x]_t^{(i)}$	$[y]_t^{(i)}$	…	0	$[z]_t^{(i)}$
…	…	…	…	…	…	…	…
P_n	0	…	$[x]_t^{(n)}$	$[y]_t^{(i)}$	…	0	$[z]_t^{(n)}$

Support Guaranteed Output Delivery

Add authentication tag for each share used for computation.

Assume $[z]_t = [x]_t + [y]_t$ caused the inconsistency, where shares of $[x]_t$ are distributed by P_{dx} and $[y]_t$ are distributed by P_{dy}.

Rx\Dist	P_1	…	P_{dx}	P_{dy}	…	P_n	
P_1	0	…	$[x]_t^{(1)}$	$[y]_t^{(1)}$	…	0	$[z]_t^{(1)}$
…	…	…	…	…	…	…	…
P_i	0	…	$[x]_t^{(i)}$	$[y]_t^{(i)}$	…	0	$[z]_t^{(i)}$
…	…	…	…	…	…	…	…
P_n	0	…	$[x]_t^{(n)}$	$[y]_t^{(i)}$	…	0	$[z]_t^{(n)}$

- Case 1: P_i cheat during computing $[z]_t^{(i)} := [x]_t^{(i)} + [y]_t^{(i)}$

Support Guaranteed Output Delivery

Add authentication tag for each share used for computation.

Assume $[z]_t = [x]_t + [y]_t$ caused the inconsistency, where shares of $[x]_t$ are distributed by P_{dx} and $[y]_t$ are distributed by P_{dy}.

Rx\Dist	P_1	…	P_{dx}	P_{dy}	…	P_n	
P_1	0	…	$[x]_t^{(1)}$	$[y]_t^{(1)}$	…	0	$[z]_t^{(1)}$
…	…	…	…	…	…	…	…
P_i	0	…	$[x]_t^{(i)}$	$[y]_t^{(i)}$	…	0	$[z]_t^{(i)}$
…	…	…	…	…	…	…	…
P_n	0	…	$[x]_t^{(n)}$	$[y]_t^{(i)}$	…	0	$[z]_t^{(n)}$

- Case 1: P_i cheat during computing $[z]_t^{(i)} := [x]_t^{(i)} + [y]_t^{(i)}$
- Case 2: P_i did not cheat, but $[x]_t$ (or $[y]_t$) is not consistent.

Support Guaranteed Output Delivery

Add authentication tag for each share used for computation.

In general

Rx\Dist	P_1	…	P_j	P_{j+1}	…	P_n	
P_1	$[z(j)]_t^{(1)}$	…	$[z(j)]_t^{(1)}$	$[z(j+1)]_t^{(1)}$	…	$[z(n)]_t^{(1)}$	$[z]_t^{(1)}$
…	…	…	…	…	…	…	…
P_i	$[z(j)]_t^{(i)}$	…	$[z(j)]_t^{(i)}$	$[z(j+1)]_t^{(i)}$	…	$[z(n)]_t^{(i)}$	$[z]_t^{(i)}$
…	…	…	…	…	…	…	…
P_n	$[z(j)]_t^{(n)}$	…	$[z(j)]_t^{(n)}$	$[z(j+1)]_t^{(n)}$	…	$[z(n)]_t^{(n)}$	$[z]_t^{(n)}$

1. Upon parties finding [z] inconsistent, parties cannot directly broadcast [z] they hold.
2. Parties run randomness generation protocol to obtain [r] and broadcast [r]. If [r] is consistent, mask [z] with [r] as [a] = [z] + [r]. Otherwise set [a] = [r].
3. Now parties can safely broadcast their shares of [a].

Support Guaranteed Output Delivery

Add authentication tag for each share used for computation.

Rx\Dist	P_1	…	P_j	P_{j+1}	…	P_n	
P_1	$[a(j)]_t^{(1)}$	…	$[a(j)]_t^{(1)}$	$[a(j+1)]_t^{(1)}$	…	$[a(n)]_t^{(1)}$	$[a]_t^{(1)}$
…	…	…	…	…	…	…	…
P_i	$[a(j)]_t^{(i)}$	…	$[a(j)]_t^{(i)}$	$[a(j+1)]_t^{(i)}$	…	$[a(n)]_t^{(i)}$	$[a]_t^{(i)}$
…	…	…	…	…	…	…	…
P_n	$[a(j)]_t^{(n)}$	…	$[a(j)]_t^{(n)}$	$[a(j+1)]_t^{(n)}$	…	$[a(n)]_t^{(n)}$	$[a]_t^{(n)}$

Any $[a(j)]_t^{(i)}$, a share dealt by P_j and received by P_i, can be revealed.

: shares been revealed and checked

Support Guaranteed Output Delivery

Add authentication tag for each share used for computation.

Rx\Dist	P_1	…	P_j	P_{j+1}	…	P_n	
P_1	$[a(j)]_t^{(1)}$	…	$[a(j)]_t^{(1)}$	$[a(j+1)]_t^{(1)}$	…	$[a(n)]_t^{(1)}$	$[a]_t^{(1)}$
…	…	…	…	…	…	…	…
P_i	$[a(j)]_t^{(i)}$	…	$[a(j)]_t^{(i)}$	$[a(j+1)]_t^{(i)}$	…	$[a(n)]_t^{(i)}$	$[a]_t^{(i)}$
…	…	…	…	…	…	…	…
P_n	$[a(j)]_t^{(n)}$	…	$[a(j)]_t^{(n)}$	$[a(j+1)]_t^{(n)}$	…	$[a(n)]_t^{(n)}$	$[a]_t^{(n)}$

Any $[a(j)]_t^{(i)}$, a share dealt by P_j and received by P_i, can be revealed.

1. Each party P_i has to provide a decomposition of $[a]_t^{(i)}$. If P_i fails to do so, its index is recorded in Devi.

: shares been revealed and checked

Support Guaranteed Output Delivery

Add authentication tag for each share used for computation.

Rx\Dist	P_1	…	P_j	P_{j+1}	…	P_n	
P_1	$[a(j)]_t^{(1)}$	…	$[a(j)]_t^{(1)}$	$[a(j+1)]_t^{(1)}$	…	$[a(n)]_t^{(1)}$	$[a]_t^{(1)}$
…	…	…	…	…	…	…	…
P_i	$[a(j)]_t^{(i)}$	…	$[a(j)]_t^{(i)}$	$[a(j+1)]_t^{(i)}$	…	$[a(n)]_t^{(i)}$	$[a]_t^{(i)}$
…	…	…	…	…	…	…	…
P_n	$[a(j)]_t^{(n)}$	…	$[a(j)]_t^{(n)}$	$[a(j+1)]_t^{(n)}$	…	$[a(n)]_t^{(n)}$	$[a]_t^{(n)}$

Any $[a(j)]_t^{(i)}$, a share dealt by P_j and received by P_i, can be revealed.

1. Each party P_i has to provide a decomposition of $[a]_t^{(i)}$. If P_i fails to do so, its index is recorded in Devi.
2. Otherwise, there exists an index j such that [a(j)] is inconsistent.
   - P_j accuses that $[a(j)]_t^{(i)}$ revealed by P_i is incorrect. If they are not in dispute, (i,j) is recorded in *Disp*.

: shares been revealed and checked

Support Guaranteed Output Delivery

Add authentication tag for each share used for computation.

Rx\Dist	P_1	…	P_j	P_{j+1}	…	P_n	
P_1	$[a(j)]_t^{(1)}$	…	$[a(j)]_t^{(1)}$	$[a(j+1)]_t^{(1)}$	…	$[a(n)]_t^{(1)}$	$[a]_t^{(1)}$
…	…	…	…	…	…	…	…
P_i	$[a(j)]_t^{(i)}$	…	$[a(j)]_t^{(i)}$	$[a(j+1)]_t^{(i)}$	…	$[a(n)]_t^{(i)}$	$[a]_t^{(i)}$
…	…	…	…	…	…	…	…
P_n	$[a(j)]_t^{(n)}$	…	$[a(j)]_t^{(n)}$	$[a(j+1)]_t^{(n)}$	…	$[a(n)]_t^{(n)}$	$[a]_t^{(n)}$

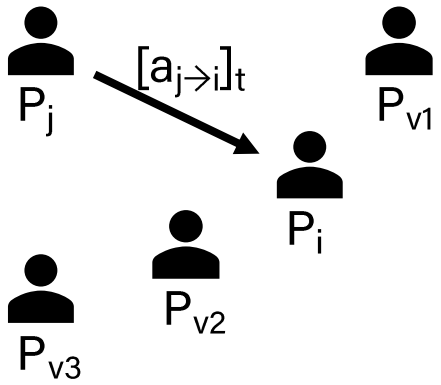
Any $[a(j)]_t^{(i)}$, a share dealt by P_j and received by P_i, can be revealed.

1. Each party P_i has to provide a decomposition of $[a]_t^{(i)}$. If P_i fails to do so, its index is recorded in Devi.
2. Otherwise, there exists an index j such that [a(j)] is inconsistent.
   - P_j accuses that $[a(j)]_t^{(i)}$ revealed by P_i is incorrect. If they are not in dispute, (i,j) is recorded in *Disp*.
   - Otherwise, P_i and P_j have to authenticate $[a(j)]_t^{(i)}$ to other parties.

: shares been revealed and checked

Guaranteed Output Delivery

Parties compute tags for share $[a_{j\to i}]_t$ dealt by P_j received by P_i.



Recall the definition of MPC: $f(x_1, \ldots, x_n)=y_1, \ldots, y_n$
Define a tag computation function tag-comp() whose input and output are define in the table below.
P_i and P_j are in dispute; all other parties become their *verifier*.

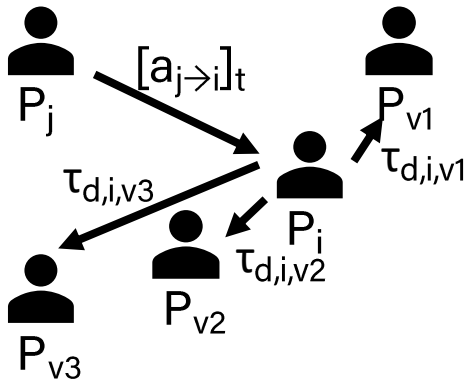
- Each verifier prepares authentication key pair $(\mu_{v\to i}, u_{v\to i})$ for P_i.
- Whenever, P_j distributes $[a_{j\to i}]_t$ to P_i, parties run tag-comp and P_i receives $\tau_{d,i,v1}, \tau_{d,i,v2}, \tau_{d,i,v3}$.

\Party	P_j	P_i	P_{v1}	P_{v2}	P_{v3}
Input	$[a_{j\to i}]_t$	$\perp$	$(\mu_{v1\to i}, u_{v1\to i})$	$(\mu_{v2\to i}, u_{v2\to i})$	$(\mu_{v3\to i}, u_{v3\to i})$
Output	$\perp$	$\tau_{d,i,v1}, \tau_{d,i,v2}, \tau_{d,i,v3}$	$\perp$	$\perp$	$\perp$

Where: $\tau_{d,i,v} = \mu_{v\to i} \cdot [a_{j\to i}]_t + u_{v\to i}$.

Guaranteed Output Delivery

Parties compute tags for share $[a_{j\to i}]_t$ dealt by P_j received by P_i.



The complexity is $O(n^3)$

Recall the definition of MPC: $f(x_1, ..., x_n)=y_1, ..., y_n$
Define a tag computation function tag-comp() whose input and output are define in the table below.
P_i and P_j are in dispute; all other parties become their *verifier*.

- Each verifier prepares authentication key pair ($\mu_{v\to i}$, $u_{v\to i}$) for P_i.
- Whenever, P_j distributes $[a_{j\to i}]_t$ to P_i, parties run tag-comp and P_i receives $\tau_{d,i,v1}$, $\tau_{d,i,v2}$, $\tau_{d,i,v3}$.
- Later upon P_j and P_i are in dispute on $[a_{j\to i}]_t$, P_i sends τ to corresponding verifiers.

\Party	P_j	P_i	P_{v1}	P_{v2}	P_{v3}
Input	$[a_{j\to i}]_t$	$\perp$	$(\mu_{v1\to i}, u_{v1\to i})$	$(\mu_{v2\to i}, u_{v2\to i})$	$(\mu_{v3\to i}, u_{v3\to i})$
Output	$\perp$	$\tau_{d,i,v1}$, $\tau_{d,i,v2}$, $\tau_{d,i,v3}$	$\perp$	$\perp$	$\perp$

Where: $\tau_{d,i,v} = \mu_{v\to i} \cdot [a_{j\to i}]_t + u_{v\to i}$.

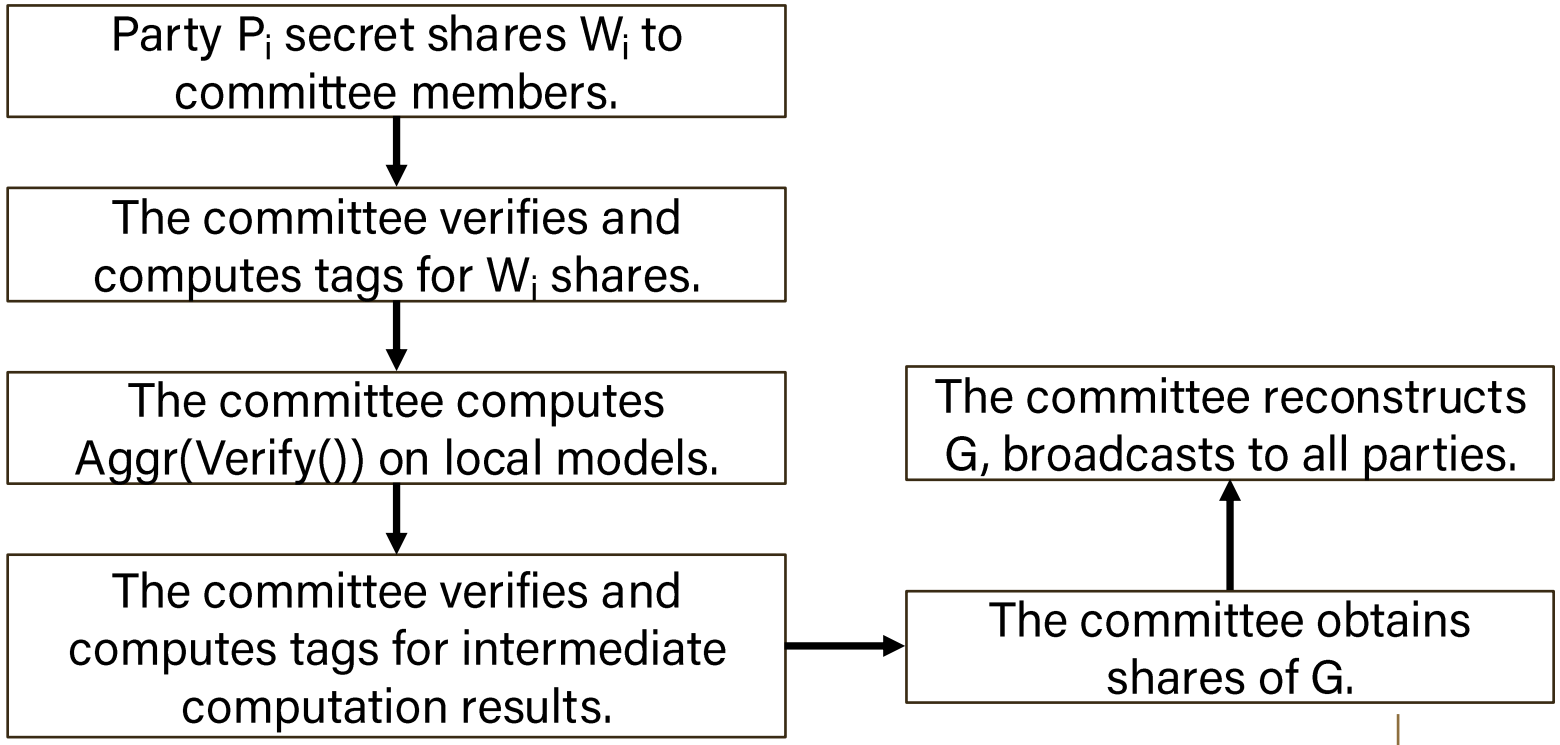
The Overall Procedure

Robust and Privacy-persevering DFL

1. Each party P_i trains a local model W_i on local dataset D_i.
2. Parties convert parameters of W_i from float numbers to field elements.
3. n parties elect k committees using the Feige's committee election protocol.
4. Each party P_i use Shamir's (k, k/2)-Secret Sharing to share W_i to committee members.
5. The committee members verify the shares of W_i for $i \in [n]$, upon successful verification, compute tags for each share.
6. The committee runs Aggr(Verify($\{W_i\}$)) using the protocols for required secure operations (add, mult, sqrt, truncate, ...) to obtain shares of G.
7. If dispute or deviating parties identified during steps 4 to 6, remaining parties update *Disp* and *Devi* and repeat the computation.
8. Committee members reconstruct G, broadcast to all parties.

Summary

We bridged the gap between GOD MPC and FL secure aggregation.



THANK YOU

Q&A