Paper-2

Enabling Execution Assurance of Federated Learning on Untrusted Participants.

Keywords:
Proof of work, Trusted Execution Environment (TEE),

Issues and Defenses:

1. remove/weaken outliers from model update of participants., to prevent model poisoning attacks [7] [9]-[11]
2. ~~[9]-[11]~~ gaurentee convergence of global model [8].
3. data poisoning attacks, stealthy backdoor attacks [12], [13].
4. prevent lazy participants.

TEE related issues.

1. only executing TEE requested training rounds.
2. ~~Defenses~~ freshness can't be gaurenteed, as
3. ~~[7]-[9]~~ TEE's scheduling and I/O are controlled by host participant [20].

Issues Contd

5. old results corresponding correct proofs in the previous FL epoch to claim rewards.
6. sybil-based cheating attacks to ask for multiple rewards.

1. "Commit and prove" : Before exposing sampling decisions, the TEE requires to receive a commitment from the participant freshness :

2. "Dynamic yet deterministic" data selection leverage dynamic input data to make training results different.

3. Bind selecting decision with each FL's epoch's identifier.

4. Tampering free identifiers to participants against sybil attacks.

Intel's Software Guard Extension (SGX)

1. generates a measurement with hash digest
2. program in enclave produces a report including the measurement and supplementary data (e.g., output), and signs it as a quote with a private attestation key fused in the processor.
3. can be verified by trustworthy measurement of binary and and Intel's Attestation Service (IAS)

(sK_TEE, pK_TEE) ; Σ → unforgeable signature

sK → attestation
pK → key

enclave generate attestation

$\sigma = \Sigma.Sign(SK_{TEE}, \phi, out)$

output out is produced by program ϕ running in enclave.

$\Sigma.veri(pK_{TEE}, \sigma, \phi, out)$ outputs 1

Threat Model

mitigating sybil attacks by diminishing gains of adversaries with increased computing efforts.

OVERVIEW

1. **Solicitation**
   specifies training algo, rewards etc.
2. **Registration**
   participants P and SGX-enabled platform can register with info like data description and identifier. Then, P reviews and downloads the program, launches the on enclave with the program, and generates an attestation report to demonstrate the correctness program setup. S verifies → $\Sigma.Veri()$.
3. **Selection**
   S selects subset of participants [29], [33], for unselected S would instruct them when they can reconnect for participation.

4. Reporting: P gets the largest model θ_0^t, performs local training generates model update Δ_p^t and invokes inside program TEE to generate proof σ_p.
TEE runs $(SK_{TEE}, DNN(), D, \theta_0^t, \Delta_p^t) \rightarrow \sigma_p$
verfies using randomly selected training rounds.

5. Verification and Aggregation:
S performs Verify $(pk_{TEE}, DNN(), \theta_0^t, \Delta_p^t, \sigma_p) \rightarrow 1/0$.
S distributes rewards.

DESIGN DETAILS

$Prog_0$ - program outside enclave
$Prog_i$ → inside

A. Baseline Design

$Prog_0$ stores the hash digest of all model parameters θ_s at the end of each training round and sends commit message to TEE when all training rounds finish.
$Prog_i$ requests to check one random training round, $Prog_0$ returns the input parameters, the hash value of output para, corresponding training data. Then $prog_i$ checks whether they are consistent.

Prog$_i$ ← model parameters for last round

↓

computes model update

checks commitment message

generates a proof as local attestation of local execution

Design of Commitment Scheme

1) results of all training rounds and order of results
2) arbitrary result verified according to commitment efficiently.

Straightforward: $\zeta = \{h_1, h_2, \ldots, h_R\}$ (hash from R rounds)

Prog$_i$ randomly fetches several pairs of two successive hash values, requests raw input model parameters from Prog$_0$, and recomputes hash values to insure coincidence.

For constant performance: we adopt Merkle hash tree [35] based commitment

MHT based: Prog$_0$ builds Merkel Tree, leaf nodes are hash values. Internal nodes are concatenation of hash values of two children.

Prog$_i$ requests ~~~~ generates nos to identify the training rounds to be verified.

Input parameters, hash value of output para auxiliary info(sibling nodes of nodes in the path from the requested leaf node to the tree root).

8. Guaranteeing the Freshness of Results

Scheme is hard to gaurentee the freshness of single training round and final results.

$r \rightarrow$ rounds performed, $R \rightarrow$ required no. of rounds, $L \rightarrow$ no. of randomly selected rounds

detected misbehaviour: $1 - \binom{R - \lceil R/r \rceil + 1}{L} / \binom{R}{L}$

Solution: $\binom{n}{r} = {}^{n}C_{r}$

leverage dynamics of input training data, which make epochs differentiable.

Prog$_i$ negotiate with Prog$_0$ about data organization in advance.

Prog$_i$ generates an index and HMAC's for each training data restored outside enclave before training.

Prog$_i$ $\xrightarrow[\text{seed}]{}$ Prog$_0$

$I_{ij} = PRF(seed \, X \, i+j) \bmod N$

index of jth training data at round i

PRF $\rightarrow$ pseudo random function.

N $\rightarrow$ total training data.

Checks checks integrity of outside data with HMAC, trains the DNN, verifies the output parameters Bind training results with used global model seed = Prog$_i$ receive ~~highest~~ digest of current global model (denoted as h^t) from Prog$_0$.

Using signature and model digest, the server can ensure ~~that~~ whether the participant used latest model and faithfully performed local training.

Sybil-based, unique identifier T, tamper free identifier, even if an adversary registers multiple aliases, he must launch some no. of enclave instances to generate proofs.