AI and Cryptography

Lecture 7 – Secure Multiparty Computation for Private ML

Luca Mariot

Semantics, Cybersecurity and Services Group, University of Twente

`l.mariot@utwente.nl`

Trieste, June 30, 2023

Topics and Reading Material

Main topics:

- Basics of Secure Multiparty Computation (SMPC)
- SMPC for privacy-preserving ML

References:

- D. Evans et al.: A Pragmatic Introduction to Secure Multi-Party Computation. NOW Publishers, 2018
- R. Xu et al.: Privacy-Preserving Machine Learning: Methods, Challenges and Directions. arXiv:2108.04417, 2021

This Lecture

Intro to SMPC

Oblivious Transfer

Garbled Circuits

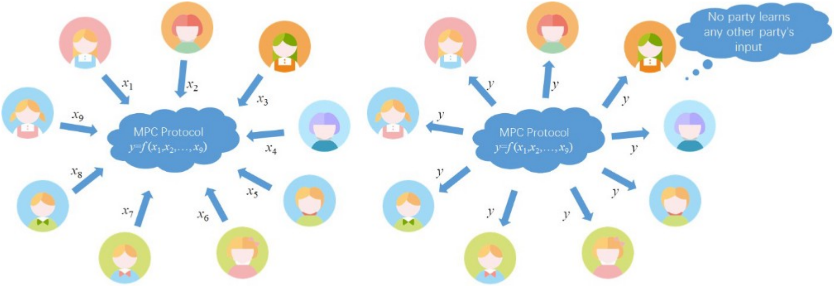
Secret Sharing

SMPC for private ML

Secure Multiparty Computation

Secure Multiparty Computation (SMPC)

SA allows parties to jointly compute an aggregated value without revealing their individual values.



Source:`https://alibaba-gemini-lab.github.io/docs/blog/pvc/`

Characteristics of SMPC

Straightforward Solution: *Trusted Third Party* (TTP)

- Users send their private inputs to a server that computes the function and send back the result, without revealing the inputs
- ... in many realistic setting, this is not feasible!

SMPC Solution: no TTPs

- Users need to collaborate and interact through a protocol
- Typical adversarial models:
  - Semi-honest
  - Malicious
  - Colluding

A metaphor: let's play cards...

This Lecture

Intro to SMPC

Oblivious Transfer

Garbled Circuits

Secret Sharing

SMPC for private ML

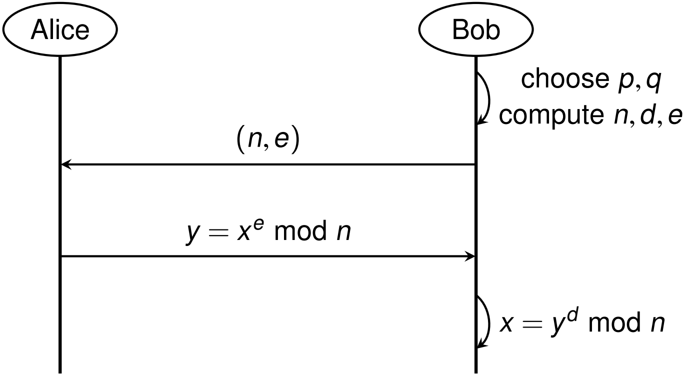
Oblivious Transfer

- Probably the first "proper" protocol of SMPC
- Invented by Rabin in 1981[1], refined by Even, Goldreich and Lempel in 1985[2]
- **Idea**: Alice sends a message (bit) to Bob, but does not know what she is sending
- **Implementation:** through RSA, let's review it...

[1] Michael O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981

[2] S. Even, O. Goldreich, and A. Lempel, "A Randomized Protocol for Signing Contracts", Communications of the ACM, Volume 28, Issue 6, pg. 637–647, 1985.

RSA – Scheme



- Finding d from (n, e) requires *factorizing* n

Hard-core Predicates

Definition

A function $hc : \{0,1\}^* \to \{0,1\}$ is a *hard-core predicate* for a one-way permutation $f : \{0,1\}^* \to \{0,1\}^*$ if:

1. hc can be computed by a polynomial time algorithm.
2. For all PPT algorithm A there is a negligible function $negl : \mathbb{N} \to \mathbb{R}$ such that

$$Pr[A(f(x)) = hc(x)] \leq \frac{1}{2} + negl(n)$$

   where x is sampled with uniform probability from $\{0,1\}^n$.

- Informally, the output bit of a hard-core predicate cannot be predicted with probability significantly larger than $\frac{1}{2}$.

Hard-core Predicate for RSA

- Assume that we have to encrypt one bit at the time, using RSA
- Problem: for any choice of the public key (n, e), it holds:

$$0^e \equiv 0 \bmod n$$
$$1^e \equiv 1 \bmod n$$

  hence, $y = x$, for all $x \in \mathbb{Z}_n$!
- It can be proved that the least significant bit of the plaintext x ($lsb(x)$) is a *hard-core* predicate for the modular exponentiation $x^e \bmod n$
- *Idea*: we put the plaintext bit into $lsb(x)$, and we choose the other bits at random

Randomized RSA

Assume that Alice wants to send the bit $b \in \{0, 1\}$ to Bob. Then, she performs the following steps:

1. Take Bob's public key (n_B, e_B)
2. Choose at random an integer $x < n_B/2$ (hence, $2x < n_B$)
3. Send to Bob $y = (2x + b)^{e_B} \bmod n_B$

When receiving y, Bob does the following to decrypt it:

1. Compute $y^{d_B} \bmod n_B = 2x + b$
2. Takes the least significant bit of the result

Randomized RSA

- Remark: it is not known whether the other bits of x (in particular, how many of them, and which ones) are hard-core predicates for RSA
- Hence, to encrypt in a very secure way a plaintext message x, we can encrypt every bit of x with the above randomized version of RSA
- Cryptanalysis becomes very difficult
- However, if the message is long, this method is very inefficient

1-2 Oblivious Transfer from RSA

Alice:

1. Starts with secret bits $m_0, m_1 \in \{0,1\}$
2. Generates key pair $((N,e),d)$ and sends (N,e) to Bob
3. Sends two random bits $x_0, x_1 \in \{0,1\}$ to Bob

Bob:

1. Chooses $b \in \{0,1\}$ and generates random $k \in \{0,1\}$
2. *Blinding*: sends $v = (x_b + k^e) modN$ to Alice

Alice:

1. Compute $k_0 = (v - x_0)^d modN$
2. Computes $k_1 = (v - x_1)^d modN$
3. Sends $m'_0 = m_0 + k_0$ and $m'_1 = m_1 + k_1$

Bob:

- Retrieves $m_b = m'_b - k$

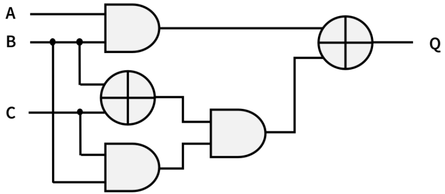
Secure AND with 1-2 OT

- Alice has 2 bits, 0 and x (private input)
- Bob has b (private input)
- Alice and Bob execute $1-2$ Oblivious Transfer
- Bob in the end gets:
  - 0 when $b=0$
  - 1 when $b=1$

This Lecture

Garbled Circuits

- Introduced by Yao in 1986[3]
- **Idea**: convert the function $f(x, y)$ in a Boolean circuit

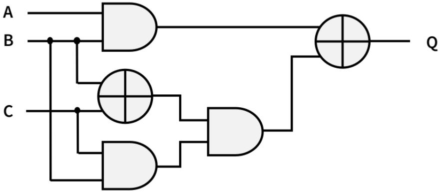


- Let's start simple, with the AND function

[3]Yao, A: How to generate and exchange secrets. Foundations of Computer Science, 1986 pp. 162–167 (1986)

General GC: Idea

- Alice garbles the table of *each* gate composing the circuit
- The output of each gate is used as an input for the next one
- **Remark**: OT is only needed in the first layer of the circuit
- For the rest, only AES is needed

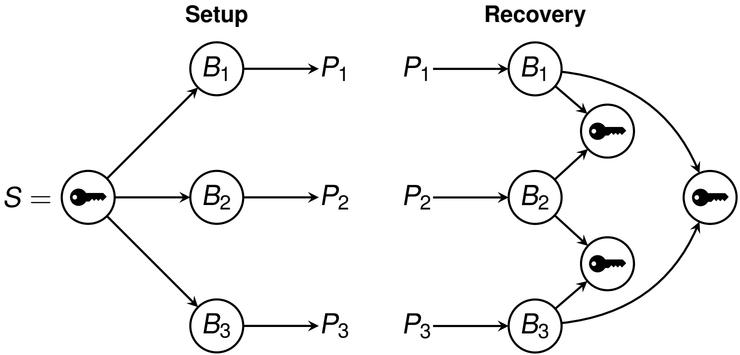


This Lecture

Secret Sharing Schemes (SSS)

(k, n) Threshold Secret Sharing Scheme: a procedure enabling a dealer to share a secret S among n players so that at least k players out of n can recover S.

Example: $(2,3)$–scheme



Remark: $(2,2)$–scheme $\Leftrightarrow$ Latin square

Secret Sharing in Practice

1	1	1	1
1	2	2	2
1	3	3	3
2	1	2	3
2	2	3	1
2	3	1	2
3	1	3	2
3	2	1	3
3	3	2	1

- We saw what is the combinatorial structure underlying threshold SSS: **orthogonal arrays** (OA)
- But how to construct an OA in practice?
- **Additive** (n, n) **SSS**:

$$S \in \mathbb{Z}_N = \{0, \cdots, N-1\}$$

$$S = B_1 + B_2 + \cdots + B_n \ mod\ N$$

- *All* shares are required to reconstruct S

This Lecture

Intro to SMPC

Oblivious Transfer

Garbled Circuits

Secret Sharing

SMPC for private ML

The importance of data for ML

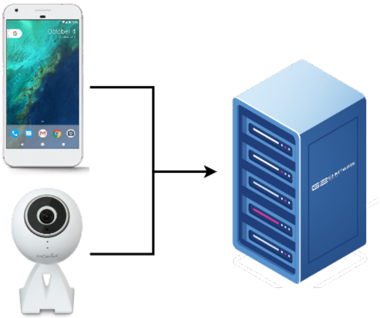
Data is born at the edge:

- Smartphones, connected devices, and IoT devices constantly generate (and share) data.
- Data enables better products and smarter models.

The importance of data for ML

Data is then shared from the device to the server for further processing, e.g., training and data mining ....

- However, data sharing incurs data **privacy issues**



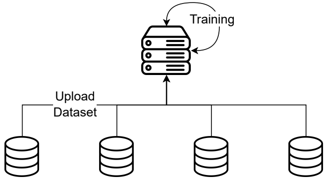
Collaborative learning

Collaborative learning allow training ML models in **decentralized** settings.

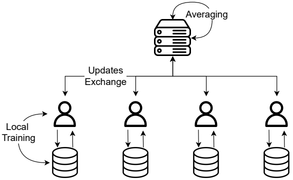
- Data remains in the device.
- But, how?
- **Federated Learning**

FL [4] enables training ML models without data sharing.

- Each device (clients) trains a small model **locally**.
- The model is then shared with the server (aggregator).
- The server merges the models using **FedAvg**.
- The aggregated model is sent back to each client.
- Repeat.



(a) ML



(b) FL

[4] McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017, April). Communication-efficient learning of deep networks from decentralized data. In Artificial intelligence and statistics (pp. 1273-1282). PMLR.

Federated Averaging

The Federated Averaging algorithm

Until convergence:

1. Select a random subset of clients n.
2. Send current model parameters θ_t^k to each client $k \in \{1, \cdots, n\}$. For each client $k \in \{1, \cdots, n\}$:
   2.1 Train on θ_t and get θ'_{t+1}.
   2.2 Return $\theta'_{t+1} - \theta_t$.
3. The server aggregates all the models $\theta_{t+1} = \frac{1}{n} \sum_{k=0}^{n} \theta_{t+1}^k$.