CSE 410 Fall 2025
Privacy-Enhancing Technologies

Marina Blanton

Department of Computer Science and Engineering
University at Buffalo

Lecture 13: Protecting Data during Computation V
Other Techniques

Secure Computation Techniques

We discussed three main types of computing on private data

- secret sharing
- garbled circuit evaluation
- homomorphic encryption

They are fundamentally different and have different properties

We primarily covered secret sharing and will discuss other techniques in this lecture

Secret Sharing Properties

Important properties of secret sharing

- the number of computational parties $n \geq 2$
- techniques are information-theoretic (don't depend on security parameter)
- the number of communication rounds depends on the computation
- the main bottleneck is communication
  - two important metrics: communication volume and the number of rounds

Other techniques have different properties

Garbled Circuit Evaluation

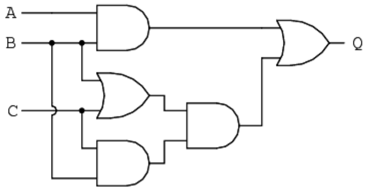
Garbled circuit evaluation is two-party computation

The steps at the high level are:

- we represent the computation as a Boolean circuit
- one party, the garbler, creates a garbled representation of the circuit
- the parties enter their inputs into the computation
- the other party, the evaluator, evaluates the circuit in garbled form
- the parties interpret the result of the computation

Garbled Circuit Evaluation

Step 1: Represent the computation as a Boolean circuit



- example: determining if two objects are similar using the Hamming distance

Garbled Circuit Evaluation

Step 2: Create a garbled representation of the circuit

The garbler associates two random labels ℓ_i^0 and ℓ_i^1 with each wire i of the circuit

- ℓ_i^0 corresponds to value 0 of the wire and ℓ_i^1 to value 1
- the bitlength of the labels is based on a security parameter

A garbled representation of each gate is formed as a set of ciphertexts

- it is based on the truth table of the gate
- it uses the labels of the input and output wires

Garbled Circuit Representation

Example: OR gate

The truth table:

0	0	0
0	1	1
1	0	1
1	1	1

Let the inputs be wires i and j and the output wire k
Given ℓ_i^0, ℓ_i^1, ℓ_j^0, ℓ_j^1, ℓ_k^0, ℓ_k^1, the garbler computes

$$c_1 = \mathsf{Enc}_{\ell_j^0}(\mathsf{Enc}_{\ell_i^0}(\ell_k^0)) \quad c_2 = \mathsf{Enc}_{\ell_j^0}(\mathsf{Enc}_{\ell_i^1}(\ell_k^1))$$

$$c_3 = \mathsf{Enc}_{\ell_j^1}(\mathsf{Enc}_{\ell_i^0}(\ell_k^1)) \quad c_4 = \mathsf{Enc}_{\ell_j^1}(\mathsf{Enc}_{\ell_i^1}(\ell_k^1))$$

The ciphertexts $c_1, \ldots, c_4$ are given to the evaluator in a random order

Garbled Circuit Evaluation

Step 3: The parties enter their inputs into the computation

Let the evaluator have n_1 inputs (wires 1 through n_1) and the garbler have n_2 inputs (wires $n_1 + 1$ through $n_1 + n_2$)

The garbler knows the input-label mapping and can send its input's labels to the evaluator

- if the garbler's input i is b_i, it sends the label $\ell^{b_i}_{n_i+i}$ to the evaluator

Garbled Circuit Evaluation

Step 3: The parties enter their inputs into the computation

Let the evaluator have n_1 inputs (wires 1 through n_1) and the garbler have n_2 inputs (wires $n_1 + 1$ through $n_1 + n_2$)

The garbler knows the input-label mapping and can send its input's labels to the evaluator

- if the garbler's input i is b_i, it sends the label $\ell_{n_i+i}^{b_i}$ to the evaluator

This approach doesn't work for the evaluator's inputs

- the evaluator needs to obtain a label for its private bit without the garbler learning its bit
- doing so requires a new tool – Oblivious Transfer

Garbled Circuit Evaluation

Oblivious Transfer (OT) is a useful cryptographic tool

1-out-of-2 Oblivious Transfer has the following functionality:

- there are two parties: the sender and the receiver
- the sender has two messages m_0, m_1
- the receiver has a bit b
- the receiver learns m_b
- the sender learns nothing

Garbled Circuit Evaluation

Oblivious Transfer (OT) is a useful cryptographic tool

1-out-of-2 Oblivious Transfer has the following functionality:

- there are two parties: the sender and the receiver
- the sender has two messages m_0, m_1
- the receiver has a bit b
- the receiver learns m_b
- the sender learns nothing

OT is an interactive protocol between two parties

- it uses cryptography
- communication is linear in the number of the sender's messages

Garbled Circuit Evaluation

Oblivious transfer in our case:

- the garbler has labels ℓ_i^0, ℓ_i^1
- the evaluator has input bit b_i
- the evaluator learns $\ell_i^{b_i}$

This is performed for each input bit of the evaluator

Garbled Circuit Evaluation

Step 4: The evaluator evaluates the circuit

The evaluator now has:

- garbled gates
- input wires' labels

The evaluator evaluates each gate in turn on its inputs

- given two input labels, the evaluator is able to decrypt one ciphertext
- the decrypted value corresponds to the label for the output wire

Once this is done, the evaluator holds labels for the output wires

Garbled Circuit Evaluation

Step 5: The parties interpret the result

The evaluator sends the labels it computed for the output wires to the garbler

The garbler announces their meaning to the evaluator

Garbled Circuit Evaluation

Properties of garbled circuit evaluation

- computational security
- constant-round communication
- communication volume is $O(n \cdot \kappa)$, where n is the number of gates and $\kappa = 128$ is a security parameter

Why does security hold?

Homomorphic Encryption

Recall that homomorphic encryption permits computation on plaintext by manipulating ciphertexts

$$\mathsf{Enc}_k(m_1) \otimes \mathsf{Enc}_k(m_2) = \mathsf{Enc}_k(m_1 \oplus m_2)$$

Homomorphic Encryption

Recall that homomorphic encryption permits computation on plaintext by manipulating ciphertexts

$$\mathsf{Enc}_k(m_1) \otimes \mathsf{Enc}_k(m_2) = \mathsf{Enc}_k(m_1 \oplus m_2)$$

Most popular types:

- partially homomorphic encryption
  $\mathsf{Enc}_k(m_1) \cdot \mathsf{Enc}_k(m_2) = \mathsf{Enc}_k(m_1 + m_2)$

Homomorphic Encryption

Recall that homomorphic encryption permits computation on plaintext by manipulating ciphertexts

$$\mathsf{Enc}_k(m_1) \otimes \mathsf{Enc}_k(m_2) = \mathsf{Enc}_k(m_1 \oplus m_2)$$

Most popular types:

- partially homomorphic encryption
  $\mathsf{Enc}_k(m_1) \cdot \mathsf{Enc}_k(m_2) = \mathsf{Enc}_k(m_1 + m_2)$
- fully homomorphic encryption (FHE)
  - both additions and multiplications are supported

Homomorphic Encryption

Recall that homomorphic encryption permits computation on plaintext by manipulating ciphertexts

$$\mathsf{Enc}_k(m_1) \otimes \mathsf{Enc}_k(m_2) = \mathsf{Enc}_k(m_1 \oplus m_2)$$

Most popular types:

- partially homomorphic encryption
  $\mathsf{Enc}_k(m_1) \cdot \mathsf{Enc}_k(m_2) = \mathsf{Enc}_k(m_1 + m_2)$
- fully homomorphic encryption (FHE)
  - both additions and multiplications are supported
- somewhat homomorphic encryption
  - an unlimited number of additions and a limited number of sequential ultiplications are supported

Homomorphic Encryption

Current FHE schemes have the following structure

- ciphertexts incorporate a certain amount of noise
- noise grows slowly during addition ($\text{noise}_1 + \text{noise}_2$)
- noise is expanded significantly during multiplications ($\text{noise}_1 \cdot \text{noise}_2$)
- after a few sequential multiplications noise starts interfering with data
- a re-encryption called bootstrapping is needed
  - bootstrapping is an expensive operation

Homomorphic Encryption

Homomorphic encryption is computationally secure and uses large parameters and large plaintext space

To improve efficiency, packing is used

- a ciphertext encrypts a vector of values
- this permits performing a number of operations on a vector at once
- different packing strategies are needed for different operations
- it is possible to re-encrypt from one type to another

Beyond Secure Computation

In addition to secure computing on private data, it is possible to protect and release the result of (local) computation on private data

The concept for achieving it is called differential privacy

- the fundamental difference is that only the result is protected
- the computation itself proceeds as usual
- differential privacy can be combined with secure computation

Differential Privacy

Differential privacy was designed for releasing statistical information about private data sets

Differential Privacy

Differential privacy was designed for releasing statistical information about private data sets

Conceptually, it guarantees that a statistical query result is minimally impacted by a single individual

- obtain the result if you are in the dataset
- obtain the result if you are not in the dataset
- one should not be able to meaningfully tell the difference

Differential Privacy

There is a database $\mathcal{D}$ containing sensitive information about individuals

The database owner is able to evaluate and answer statistical queries about the dataset

The way in which the answers are formed is called a mechanism

We want the mechanism not to leak sensitive information about the individuals

Differential Privacy

Two databases $\mathcal{D}_1$ and $\mathcal{D}_2$ are neighboring if they differ by a single record

- the target user is present in one and not present in the other

Differential Privacy

Two databases $\mathcal{D}_1$ and $\mathcal{D}_2$ are neighboring if they differ by a single record

- the target user is present in one and not present in the other

A mechanism $\mathcal{M}$ is differentially private if responses don't let us determine which database was used

Differential Privacy

Two databases $\mathcal{D}_1$ and $\mathcal{D}_2$ are neighboring if they differ by a single record

- the target user is present in one and not present in the other

A mechanism $\mathcal{M}$ is differentially private if responses don't let us determine which database was used

The definition is parameterized by a security parameter ε:

- a mechanism $\mathcal{M}$ is ε-differentially private if for any possible set of responses $\mathcal{R}$

$$\Pr[\mathcal{M}(\mathcal{D}_1) \in \mathcal{R}] \leq e^{\varepsilon} \cdot \Pr[\mathcal{M}(\mathcal{D}_2) \in \mathcal{R}]$$

Differential Privacy

The mechanism typically depends on the type of function/computation being performed

It consists of adding noise to the disclosed information to achieve differential privacy for every individual in the dataset

There is a trade-off between privacy and utility:

- the more protection we want, the more noise needs to be added

Differential privacy has been applied to far more functions than statistical queries

If one wants to release a differentially private dataset, meaningful privacy may not be achievable

Summary

We finished secure computation!