CSE 715 Fall 2025
Design and Implementation of Application for Computing with Private Data

Marina Blanton

Department of Computer Science and Engineering
University at Buffalo

Lecture 2: Secure Computation

Data Protection

Cryptographic techniques are used for

- Protecting data at rest
- Protecting data in transmission
- Protecting data during computation

Computing with Private Data

Computing on private data can be accomplished if the data does not leave the premises of where it was collected

- analysis of one's medical history collected by a doctor's office
- internally evaluating a company's performance
- computing a student's GPA

Computing with Private Data

Computing on private data can be accomplished if the data does not leave the premises of where it was collected

- analysis of one's medical history collected by a doctor's office
- internally evaluating a company's performance
- computing a student's GPA

This holds even if the data is a subject of legal restrictions

- legal restrictions often control data sharing
- they can also regulate protection of data at rest

Computing with Private Data

Computing on private data can be accomplished if the data does not leave the premises of where it was collected

- analysis of one's medical history collected by a doctor's office
- internally evaluating a company's performance
- computing a student's GPA

This holds even if the data is a subject of legal restrictions

- legal restrictions often control data sharing
- they can also regulate protection of data at rest

Privacy-preserving computation (or secure computation) refers to computing on private data across different trust domains

Computing with Private Data

Computing across different trust domains can take many forms

- evaluating predisposition to a genetic disease
  - a patient has a DNA, a service provider has genetic pattern

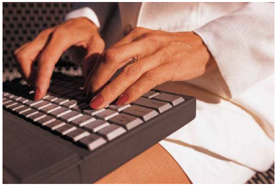
- determining the best treatment for a rare condition by multiple hospitals
  - hospitals cannot easily share patients' data

- computation outsourcing to a cloud provider
  - offloading an expensive task, e.g.., pharmaceutical evaluation of new drug

The Famous Millionaires Problem

- Alice and Bob would like to determine who is richer without revealing their worth to each other

Alice
private x



$$\longrightarrow$$
$$\longleftarrow$$
$$\longrightarrow$$
$$\longleftarrow$$
$$\longrightarrow$$

Bob
private y

output $x < y$

Secure Computation

Secure multi-party computation refers to the ability of multiple individuals to evaluate a function on their respective private inputs without disclosing them

- Each participant holds their own private data
- Participants jointly perform the computation on cryptographically protected private data
- Private data can only leave the owner after applying proper cryptographic protection
- Only the outcome is revealed to the intended output recipients

Secure Computation

- This is very different from traditional ways of computing:
  - obtain access to private data and promise to comply with data usage requirements
  - or private data is accessible to software, but the software doesn't let users to "see" the data

Secure Computation

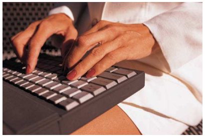
- This is very different from traditional ways of computing:
  - obtain access to private data and promise to comply with data usage requirements
  - or private data is accessible to software, but the software doesn't let users to "see" the data
- Contrast this with secure computation:
  - nothing about private data is recoverable by others throughout the computation
  - observed information is the same as if the computation was performed by a trusted third party

Secure Computation

The security expectations are as if a trusted third party performed the computation and handed the result to the participants

$b = (x < y)$

Alice



input x, output b

Bob

input y, output b

Secure Computation

Secure computation solutions can be categorized into 3 major types:

- those based on homomorphic encryption
- those based on secret sharing
- those based on garbled circuit evaluation

Homomorphic Encryption

Homomorphic encryption is a special type of encryption that, given ciphertexts, permits computation on the underlying plaintexts

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

Contrast this with conventional encryption we previously considered

- changes to a ciphertext often garble the data

Additional properties of homomorphic encryption restrict the type of arithmetic that can be used

Homomorphic Encryption

Different types of homomorphic encryption are known:

- partially homomorphic encryption
- fully homomorphic encryption

Partially homomorphic encryption

- supports a single operation on ciphertexts

Homomorphic Encryption

Different types of homomorphic encryption are known:

- partially homomorphic encryption
- fully homomorphic encryption

Partially homomorphic encryption

- supports a single operation on ciphertexts
- additively homomorphic encryption
  $\mathsf{Enc}_k(m_1) \cdot \mathsf{Enc}_k(m_2) = \mathsf{Enc}_k(m_1 + m_2)$
- multiplicatively homomorphic encryption
  $\mathsf{Enc}_k(m_1) \cdot \mathsf{Enc}_k(m_2) = \mathsf{Enc}_k(m_1 \cdot m_2)$
- intuition

Homomorphic Encryption

Fully homomorphic encryption (FHE)

- supports two operations on ciphertexts: addition and multiplication
- allows for any functionality to be evaluated on encrypted data
- this representation is called an arithmetic circuit

Homomorphic encryption enables computation on encrypted data and results in efficient protocols for certain problems

- it is well suited for secure computation outsourcing

Secret Sharing

Another way to compute on private values is by splitting them into multiple shares

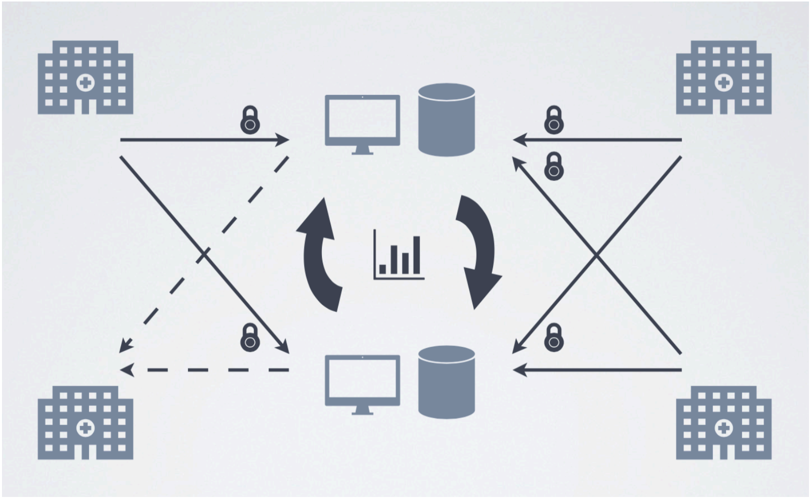
- this is called secret sharing
- given value s, generate shares $s_1, s_2, \ldots, s_n$
- each s_i is stored in a different place and doesn't reveal the secret
- access to enough shares allows for s to be reconstructed, but individual shares don't reveal anything
- specifically, (n, t) threshold secret sharing means
  - a secret s is divided into n shares
  - access to $\leq t$ shares reveals nothing about s
  - access to $> t$ shares allows for s to be reconstructed

Computation Using Secret Sharing

A number of participants would like to perform joint computation on their private inputs

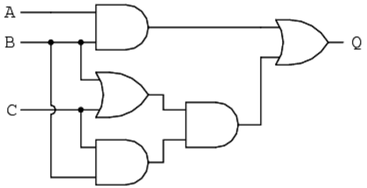
- secret sharing: each input owner creates shares of its private inputs and communicates a share to each party running the computation
- secure computation: computation parties evaluate the function on shares one operation at a time
- once the result is computed, shares are communicated to the output recipients
- output reconstruction: each output recipient reconstruct its output from the received shares

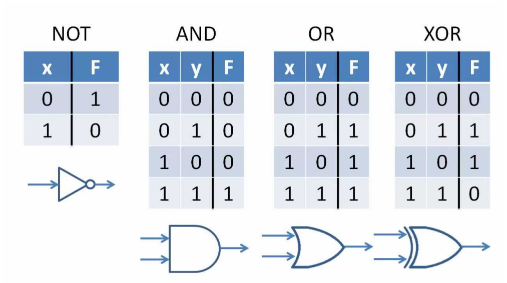
Computation Using Secret Sharing



Garbled Circuit Evaluation

With garbled circuit evaluation, the computation is represented as Boolean circuit





NOT

x	F
0	1
1	0

AND

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

Garbled Circuit Evaluation

The computation is performed by two parties:

- one party plays the role of a circuit garbler
- the other party is the circuit evaluator

The circuit garbler associates a random label with each value of each wire

The circuit evaluator evaluates the circuit on private inputs without knowing the meaning of values it handles

The idea is to decouple evaluation from its meaning

Secure Multi-Party Computation

Today is prime time for secure computation

- speed and abilities of secure computation techniques have improved dramatically

Secure Multi-Party Computation

Today is prime time for secure computation

- speed and abilities of secure computation techniques have improved dramatically
- a variety of tools have been developed

Secure Multi-Party Computation

Today is prime time for secure computation

- speed and abilities of secure computation techniques have improved dramatically
- a variety of tools have been developed
- a number of companies now offer this as a product

sharemind Roseman Labs PARTISIA inpher

Google J.P.Morgan

Real-World Uses

Danish sugar beet auction

- country-wide sealed bid auction that cryptographically protected farmer's bids

Tax fraud detection in Estonia

- citizens pay taxes on goods acquired abroad
- the government can compute taxes without access to purchases

Correlation between student employment and college graduation

- employment information was extracted from tax records
- it was combined with university records

Real-World Uses

City of Boston gender wage gap study

- analysis of wages by gender and race in the Greater Boston area
- significantly larger participation the second year

Effectiveness of Google ads

- Google knows on what ads a user clicked
- the store knows how much a user spent at the store
- this allows for determining effectiveness of ads

Privacy-Preserving Data Analytics

Data collection and analysis are all around us

- this allows for personalized medicine, targeted advertisement, ...
- this also brings unintended data leakage and disclosure

Privacy-Preserving Data Analytics

Data collection and analysis are all around us

- this allows for personalized medicine, targeted advertisement, ...
- this also brings unintended data leakage and disclosure

Privacy-preserving data mining and machine learning are receiving a significant amount of attention

- data can be distributed across multiple entities or organizations
- a model can be trained by one party, but queried by others
- the task of building a model can be outsourced

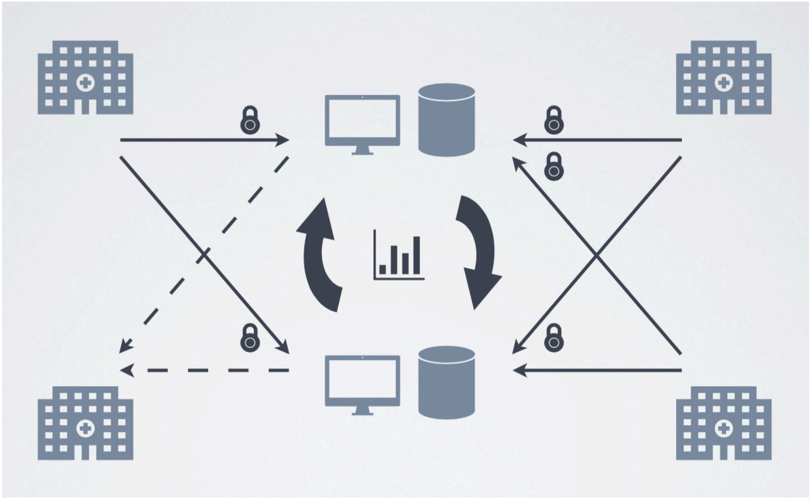
Secure Computation Summary

Computing on private data requires protection when the data resides in different trust domains

Secure multi-party computation is making rapid progress and is entering our lives

- it is an additional mechanism to keep our data protected
- it allows for our data to be used without disclosing it

Computation Using Secret Sharing



Secret Sharing

With (n, t) secret sharing, a private value s is split into n shares $s_1, \ldots, s_n$

Access to t or fewer shares reveals no information about s

Access to $t + 1$ or more shares permits reconstruction of the secret

Computational parties operate on shares, which translates to operations on the corresponding secrets

Computation takes place over a finite ring or field, modulo some modulus N

Secret Sharing

Example: additive secret sharing with $n = 2$ parties

- additive means we use addition to produce shares
- access to a single share reveals no information about a secret
- our secret is $0 \leq x < N$
- to generate shares:
  - choose random r from $\mathbb{Z}_N$ and set the first share $x_1 = r$
  - compute the second share $x_2 = (x - r) \bmod N$
- to reconstruct, compute $x = (x_1 + x_2) \bmod N$
- example

Security of Secret Sharing

Unlike encryption, secret sharing is unbreakable

- secret sharing enjoys information theoretic security and achieves perfect secrecy
- this goes back to Shannon's work in the 1940s

Let's examine the two-party secret sharing above

Why Secret Sharing is Secure

- One party holds random r
  - clearly this cannot reveal anything about secret x

Why Secret Sharing is Secure

- One party holds random r
  - clearly this cannot reveal anything about secret x
- The other party holds $x - r \bmod N$
  - this also doesn't reveal anything about x

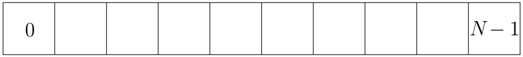
Why Secret Sharing is Secure

- One party holds random r
  - clearly this cannot reveal anything about secret x
- The other party holds $x - r \bmod N$
  - this also doesn't reveal anything about x
  - when we draw random r, all N options are equally likely

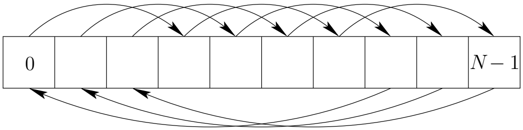
0									$N-1$

Why Secret Sharing is Secure

- One party holds random r
  - clearly this cannot reveal anything about secret x
- The other party holds $x - r$ mod N
  - this also doesn't reveal anything about x
  - when we draw random r, all N options are equally likely



  - when we add x to it, all N options are still equally likely



Why Secret Sharing is Secure

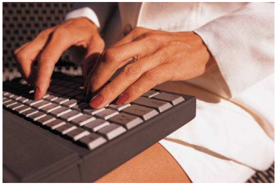
- The above means the outcome of protecting one value of x is identical to the outcome of protecting another value of x
  - this means that we learn no information about that value
- The above holds regardless of our computational capabilities
  - encryption requires that the keys and ciphertexts are sufficiently long to maintain security
  - information-theoretic techniques, on the other hand, can be used with arbitrarily small numbers

Computing with Secret Shared Values

Most types of secret sharing permit addition to be performed directly on local shares

- Addition $z = x + y$
  - assume 2-party additive secret sharing with modulus N
  - party i holds x_i, y_i and computes $z_i = (x_i + y_i) \bmod N$

Alice
x_1, y_1



$z_1 = (x_1 + y_1) \bmod N$

Bob
x_2, y_2

$z_2 = (x_2 + y_2) \bmod N$

Computing with Secret Shared Values

Multiplication $x \cdot y$

- multiplication cannot be computed using only local shares
- with two shares per value, we need to compute

$$z = x_1y_1 + x_2y_1 + x_1y_2 + x_2y_2 = z_1 + z_2 \pmod{N}$$

- two terms (x_1y_1 and x_2y_2) can be computed locally, while others require additional tools
- this requires interaction or tools such as homomorphic encryption

Computing with Secret Shared Values

Multiplication $x \cdot y$

- multiplication cannot be computed using only local shares
- with two shares per value, we need to compute

$$z = x_1y_1 + x_2y_1 + x_1y_2 + x_2y_2 = z_1 + z_2 \pmod{N}$$

- two terms (x_1y_1 and x_2y_2) can be computed locally, while others require additional tools
- this requires interaction or tools such as homomorphic encryption

(Integer) addition and multiplication are sufficient to compute any desired functionality

Computing with Secret Shared Values

Multiplication $x \cdot y$

Additive Sharing Multiplication

Now suppose that the number of parties $n > 2$

- there are n random shares x_i subject to $x = (x_1 + \ldots + x_n) \bmod N$
- how do we perform multiplication now?

Additive Sharing Multiplication

Now suppose that the number of parties $n > 2$

- there are n random shares x_i subject to $x = (x_1 + \ldots + x_n) \bmod N$
- how do we perform multiplication now?

A common solution is to use so-called Beaver triples

- precompute secret-shared a, b, and $c = ab$ for random a and b
- reconstruct $x - a$ and $y - b$
- locally compute shares of xy

Beaver Triple Multiplication

To compute additively shared $z = xy$ using a precomputed additively shared $(a, b, c = ab)$

- P_i computes and broadcasts $d_i = x_i - a_i$
- P_i computes and broadcasts $e_i = y_i - b_i$
- each party reconstructs d and e
- P_i locally computes $z_i = c_i + e \cdot a_i + d \cdot y_i + d \cdot e$

Beaver Triple Multiplication

To compute additively shared $z = xy$ using a precomputed additively shared $(a, b, c = ab)$

- P_i computes and broadcasts $d_i = x_i - a_i$
- P_i computes and broadcasts $e_i = y_i - b_i$
- each party reconstructs d and e
- P_i locally computes $z_i = c_i + e \cdot a_i + d \cdot y_i + d \cdot e$

Using $[x]$ to denote secret-shared x, we have

$$\begin{aligned}[xy] &= [(a+d)(b+e)] = [ab + ae + db + de] \\ &= [ab] + [ae] + [db] + [de] = [c] + e[a] + d[b] + de\end{aligned}$$

Replicated Secret Sharing

Replicated secret sharing (RSS) supplies more than one share to a party

- shares are still produced in an additive form
- the number of shares and their distribution follow an access structure Γ

Replicated Secret Sharing

Replicated secret sharing (RSS) supplies more than one share to a party

- shares are still produced in an additive form
- the number of shares and their distribution follow an access structure Γ

We are interested in (n, t) threshold secret sharing

- any t parties cannot learn any information about the secret
- any $t + 1$ parties can recover the secret

Replicated Secret Sharing

Replicated secret sharing (RSS) supplies more than one share to a party

- shares are still produced in an additive form
- the number of shares and their distribution follow an access structure Γ

We are interested in (n, t) threshold secret sharing

- any t parties cannot learn any information about the secret
- any $t + 1$ parties can recover the secret

Create one share for each maximal unqualified set T

- it is each set of parties of size t in our case

Distribute the share to all parties not in the set T

Replicated Secret Sharing Setup

Replicated secret sharing works over any finite ring

A ring R is a set of elements together with two binary operations $+$ and $\cdot$ such that

- R is an abelian (commutative) group under $+$
  - $a + (b + c) = (a + b) + c$ for all $a, b, c \in R$ (associative under $+$)
  - $a + b = b + a$ for all $a, b \in R$ (commutative under $+$)
  - there is the additive identity element 0 such that $a + 0 = 0 + a = a$ for every $a \in R$
  - every element a has a unique additive inverse $-a$ in R such that $a + (-a) = 0$
- there is the multiplicative identity element 1 such that $a \cdot 1 = 1 \cdot a = a$ for every $a \in R$
- $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all $a, b, c \in R$ (associative under $\cdot$)

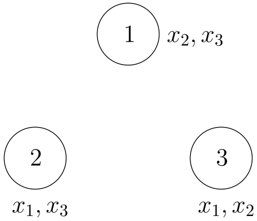
Replicated Secret Sharing

Example of $(4, 2)$ RSS

Replicated Secret Sharing

Suppose we set up RSS with $n = 3$ and $t = 1$

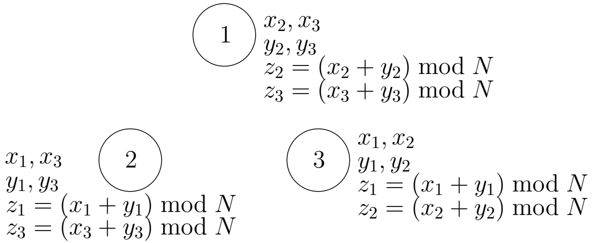
- when $t < n/2$, the setting is called honest majority and enables efficient computation



Replicated Secret Sharing

As before, addition $z = x + y$ is local

- compute each share z_i as $x_i + y_i \bmod N$


1
x_2, x_3
y_2, y_3
$z_2 = (x_2 + y_2) \bmod N$
$z_3 = (x_3 + y_3) \bmod N$
2
x_1, x_3
y_1, y_3
$z_1 = (x_1 + y_1) \bmod N$
$z_3 = (x_3 + y_3) \bmod N$
3
x_1, x_2
y_1, y_2
$z_1 = (x_1 + y_1) \bmod N$
$z_2 = (x_2 + y_2) \bmod N$

Replicated Secret Sharing

Multiplication $z = x \cdot y$ involves the following:

- note that $z = \sum_{i,j} x_i y_j$ for $i, j \in \{1, 2, 3\}$

Replicated Secret Sharing

Multiplication $z = x \cdot y$ involves the following:

- note that $z = \sum_{i,j} x_i y_j$ for $i, j \in \{1, 2, 3\}$
- the parties can jointly compute the sum of products

Replicated Secret Sharing

Multiplication $z = x \cdot y$ involves the following:

- note that $z = \sum_{i,j} x_i y_j$ for $i, j \in \{1, 2, 3\}$
- the parties can jointly compute the sum of products
  - e.g., party 1 computes $x_2 \cdot y_2 + x_2 \cdot y_3 + x_3 \cdot y_2$
  - party 2 computes $x_3 \cdot y_3 + x_3 \cdot y_1 + x_1 \cdot y_3$
  - party 3 computes $x_1 \cdot y_1 + x_1 \cdot y_2 + x_2 \cdot y_1$

Replicated Secret Sharing

Multiplication $z = x \cdot y$ involves the following:

- note that $z = \sum_{i,j} x_i y_j$ for $i, j \in \{1, 2, 3\}$
- the parties can jointly compute the sum of products
  - e.g., party 1 computes $x_2 \cdot y_2 + x_2 \cdot y_3 + x_3 \cdot y_2$
  - party 2 computes $x_3 \cdot y_3 + x_3 \cdot y_1 + x_1 \cdot y_3$
  - party 3 computes $x_1 \cdot y_1 + x_1 \cdot y_2 + x_2 \cdot y_1$
- the problem is that the resulting shares are in a different form
- this is where communication comes in place

Replicated Secret Sharing

Multiplication $z = x \cdot y$ involves the following:

- each party computes a partial sum and reshares it
- in the simplest three-party version, each party communicates 2 messages
  - illustration on the board
- this can be reduced to one message using pseudo-random values

Shamir Secret Sharing

The main disadvantage of RSS is that the number of shares grows exponentially with the number of parties

Shamir secret sharing doesn't have this drawback

- each participant stores only a single share

Computation is carried out over a finite field

- for our purposes, it often means computation modulo a prime

Shamir Secret Sharing Setup

A field F is a set of elements together with two binary operations $+$ and $\cdot$ such that for all $a, b, c \in F$

- $+$ and $\cdot$ are associative: $a + (b + c) = (a + b) + c$ and $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
- $+$ and $\cdot$ are commutative: $a + b = b + a$ and $a \cdot b = b \cdot a$
- there exists unique additive identity 0 such that $a + 0 = a$ and unique multiplicative identity 1 such that $a \cdot 1 = a$
- every $a \in F$ has a unique additive inverse $-a$ such that $a + (-a) = 0$
- every $a \in F \setminus \{0\}$ has a unique multiplicative inverse a^{-1} such that $a \cdot a^{-1} = 1$

Rings vs. Fields

What are the differences?

Shamir Secret Sharing

Each secret is represented as a polynomial of degree t with random coefficients (in F)

- given secret s, choose random $a_1, \ldots, a_t$
- let $f(x) = a_t x^t + \ldots + a_1 x + s$
- evaluate the polynomial on n distinct non-zero points that serve the purpose of shares
  - e.g., party 1 obtains $s_1 = f(1)$, party 2 obtains $s_2 = f(2)$, etc.

Shamir Secret Sharing

Each secret is represented as a polynomial of degree t with random coefficients (in F)

- given secret s, choose random $a_1, \ldots, a_t$
- let $f(x) = a_t x^t + \ldots + a_1 x + s$
- evaluate the polynomial on n distinct non-zero points that serve the purpose of shares
  - e.g., party 1 obtains $s_1 = f(1)$, party 2 obtains $s_2 = f(2)$, etc.
- access to t of fewer shares reveals no information about s
- access to $t + 1$ or more shares permits secret reconstruction via polynomial interpolation

Shamir Share Reconstruction

Reconstructing a shared secret involves interpolating the polynomial

- given pairs $(x_0, f(x_0)), \ldots (x_k, f(x_k))$ with $x_i \neq x_j$, compute

$$\ell_i(x) = \prod_{0 \le j \le k, j \neq i} \frac{x - x_j}{x_i - x_j} \text{ and } L(x) = \sum_{i=0}^{k} y_i \ell_i(x)$$

- to only determine $f(0)$, the computation simplifies to

$$\ell_i = \prod_{0 \le j \le k, j \neq i} \frac{x_j}{x_j - x_i} \text{ and } L(0) = \sum_{i=0}^{k} y_i \ell_i$$

Shamir Share Generation and Reconstruction

Example

Shamir Secret Sharing

Computing on Shamir secret shares follows a similar structure

- addition $z = x + y$ is local
  - each party i locally computes $z_i = x_i + y_i$

Shamir Secret Sharing

Computing on Shamir secret shares follows a similar structure

- addition $z = x + y$ is local
  - each party i locally computes $z_i = x_i + y_i$
- multiplication $z = x \cdot y$ is interactive
  - each party i locally computes $z_i = x_i \cdot y_i$
  - the issue is that the resulting polynomial is of degree $2t$
  - the parties re-share and lower the polynomial degree in the process
  - this dictates $n > 2t$

Simple Multiplication

Simple multiplication for Shamir secret sharing:

- let polynoimal f_x encode secret x and f_y encode secret b
- player P_i holds $f_x(i)$, $f_y(i)$ and obtains $f_z(i)$ for $z = x \cdot y$
- each P_i shares $f_x(i) \cdot f_y(i)$ by choosing a degree t polynomial h_i such that $h_i(0) = f_x(i) \cdot f_y(i)$
- P_i sends $h_i(j)$ to each P_j $(1 \leq j \leq 2t + 1)$
- each P_i computes its share $f_z(i)$ as $H(i) = \sum_{j=1}^{2t+1} \lambda_j h_j(i)$
- the constants λ_j are fixed for any t

Secret Sharing Summary

Additive secret sharing

- $t < n$, dishonest majority
- typically have $t = n - 1$
- does it require a ring or field?

Replicated secret sharing

- $t < n/2$, honest majority
- typically have $n = 2t + 1$

Shamir secret sharing

- $t < n/2$, honest majority
- typically have $n = 2t + 1$

Secret Sharing

What difference does computing over a ring vs. a field make?

Secret Sharing Summary

Secret sharing can be realized using a variety of techniques

- they are information theoretic in nature
- the setting with honest majority achieves the best performance
- addition is local, while multiplication requires interaction

We build on elementary operations to create more complex protocols

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 now will discuss other techniques

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

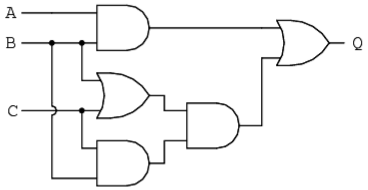
Garbled circuit evaluation is two-party computation

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:

$$\begin{array}{cc|c} 0 & 0 & 0 \\ 0 & 1 & 1 \end{array} \qquad \begin{array}{cc|c} 1 & 0 & 1 \\ 1 & 1 & 1 \end{array}$$

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 $\ell_{n_i+i}^{b_i}$ 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 $\ell^{b_i}_{n_i+i}$ 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 multiplications 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

Threshold Homomorphic Encryption

Variants of threshold homomorphic encryption exist

- each party holds a share of the decryption key
- encryption can be performed by anyone using the combined public key
- decryption uses partial decryptions by at least $t + 1$ partial keys

Final Summary

We discussed the concept of secure multi-party computation

It can be realized using a variety of techniques

- secret sharing
- homomorphic encryption
- garbled circuit evaluation

Security definitions don’t change, while performance properties vary