CSE 410 Fall 2025
Privacy-Enhancing Technologies

Marina Blanton

Department of Computer Science and Engineering
University at Buffalo

Lecture 10: Protecting Data during Computation II
Secret Sharing

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

Modular Arithmetic

Computation is over a finite set modulo some N

- the result of $a \bmod b$ is between 0 and $b - 1$
- recall that $\mathbb{Z}_N$ is the set of integers $\{0, \ldots, N-1\}$
- what is $-3 \bmod 10$?

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

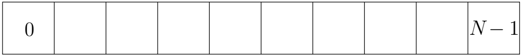
Why Secret Sharing is Secure

- 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

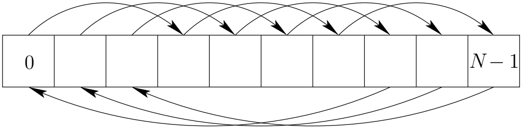
0									$N-1$

Why Secret Sharing is Secure

- One party holds random r
  - clearly this cannot reveal anything about secret x
- The other party holds $x - r \bmod N$
  - 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, 2) 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 $x \cdot y$

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

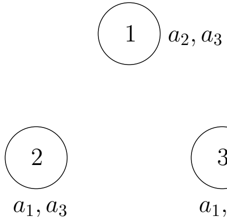
Replicated Secret Sharing

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 $c = a + b$ is local

- compute each share c_i as $a_i + b_i \bmod N$

1

a_2, a_3
b_2, b_3
$c_2 = (a_2 + b_2) \bmod N$
$c_3 = (a_3 + b_3) \bmod N$

2

a_1, a_3
b_1, b_3
$c_1 = (a_1 + b_1) \bmod N$
$c_3 = (a_3 + b_3) \bmod N$

3

a_1, a_2
b_1, b_2
$c_1 = (a_1 + b_1) \bmod N$
$c_2 = (a_2 + b_2) \bmod N$

Replicated Secret Sharing

Multiplication $c = a \cdot b$ involves the following:

- note that $c = \sum_{i,j} a_i b_j$ for $i, j \in \{1, 2, 3\}$

Replicated Secret Sharing

Multiplication $c = a \cdot b$ involves the following:

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

Replicated Secret Sharing

Multiplication $c = a \cdot b$ involves the following:

- note that $c = \sum_{i,j} a_i b_j$ for $i, j \in \{1, 2, 3\}$
- the parties can jointly compute the sum of products
  - e.g., party 1 computes $a_2 \cdot b_2 + a_2 \cdot b_3 + a_3 \cdot b_2$
  - party 2 computes $a_3 \cdot b_3 + a_3 \cdot b_1 + a_1 \cdot b_3$
  - party 3 computes $a_1 \cdot b_1 + a_1 \cdot b_2 + a_2 \cdot b_1$

Replicated Secret Sharing

Multiplication $c = a \cdot b$ involves the following:

- note that $c = \sum_{i,j} a_i b_j$ for $i, j \in \{1, 2, 3\}$
- the parties can jointly compute the sum of products
  - e.g., party 1 computes $a_2 \cdot b_2 + a_2 \cdot b_3 + a_3 \cdot b_2$
  - party 2 computes $a_3 \cdot b_3 + a_3 \cdot b_1 + a_1 \cdot b_3$
  - party 3 computes $a_1 \cdot b_1 + a_1 \cdot b_2 + a_2 \cdot b_1$
- the problem is that the resulting shares are in a different form
- this is where communication comes in place

Replicated Secret Sharing

Multiplication $c = a \cdot b$ 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 means computation modulo a prime

Shamir Secret Sharing

Each secret is represented as a polynomial of degree t with random coefficients (modulo p)

- 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 (modulo p)

- 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 Secret Sharing

Computing on Shamir secret shares follows a similar structure

- addition $c = a + b$ is local
  - each party i locally computes $c_i = a_i + b_i$

Shamir Secret Sharing

Computing on Shamir secret shares follows a similar structure

- addition $c = a + b$ is local
  - each party i locally computes $c_i = a_i + b_i$
- multiplication $c = a \cdot b$ is interactive
  - each party i locally computes $c_i = a_i \cdot b_i$
  - 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 $n > 2t$

Secret Sharing Summary

Additive secret sharing

- $t < n$, dishonest majority
- typically have $t = n - 1$

Replicated secret sharing

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

Shamir secret sharing

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

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