CS 6501 Week 10: Multiparty Computation

So far, we have looked at two-party computation. What if we have more than two parties?
Many protocols for general "multiparty computation (MPC)" — in this course, we will focus on MPC based on "secret sharing"

Secret sharing: Suppose we have a secret and want to distribute it among n parties such that any t of them can subsequently recover the secret and any $(t-1)$ subset cannot [e.g., Board of directors at Coca-Cola want to protect Coca-Cola recipe]

Definition. A (t,n)-secret sharing scheme over a message space $\mathcal{M}$ and share space $\mathcal{S}$ consists of two efficient algorithms:

$$\text{Share}: \mathcal{M} \to \mathcal{S}^n$$

$$\text{Reconstruct}: \mathcal{S}^t \to \mathcal{M}$$

with the following properties:

Correctness: Any t shares can be used to reconstruct m:

$$\forall m \in \mathcal{M} : (s_1, \ldots, s_n) \leftarrow \text{Share}(m)$$

$$\forall S \subseteq \{s_1, \ldots, s_n\} \text{ where } |S| = t : \text{Reconstruct}(S) = m$$

Security: Need at least t shares to learn the secret

$$\forall m_0, m_1 \in \mathcal{M},\ \forall I \subseteq [n] \text{ where } |I| < t :$$

$$\{(s_1, \ldots, s_n) \leftarrow \text{Share}(m_0) : s_i \text{ for all } i \in I\} \equiv \{(s_1, \ldots, s_n) \leftarrow \text{Share}(m_1) : s_i \text{ for all } i \in I\}$$

can relax to computational indistinguishability (in which case, Share takes in additional security parameter)

notion here is information-theoretic

Examples: Additive secret sharing [n out of n] : $\mathcal{M} = \mathbb{Z}_p$, $\mathcal{S} = \mathbb{Z}_p^n$

any ring will work (p need not be prime)

Provides information-theoretic security — check this!

- Share(m): Sample $r_1, \ldots, r_{n-1} \xleftarrow{R} \mathbb{Z}_p$ and set $r_n = m - \sum_{i=1}^{n-1} r_i \in \mathbb{Z}_p$
- Reconstruct$(r_1, \ldots, r_n)$: Output $\sum_{i=1}^{n} r_i$

Combinatorial secret sharing [t out of n]: Will use a symmetric encryption scheme over $\{0,1\}^*$ (e.g. AES-CTR)

satisfying CPA-security

- Share(m): Sample n keys $k_1, \ldots, k_n$ for encryption scheme

  For every t-subset $\{i_1, \ldots, i_t\} \subseteq [n]$, encrypt m using $k_{i_1}, \ldots, k_{i_t}$ (e.g. $\text{Enc}(k_{i_1}, \text{Enc}(k_{i_2}, \cdots, \text{Enc}(k_{i_t}, m) \cdots))$)

  Let $\{ct\}$ be the collection of ciphertexts ← very large shares!

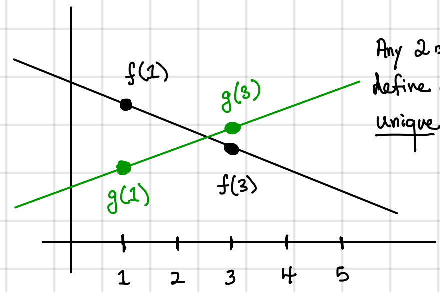
  Output shares $((k_1, \{ct\}), \ldots, (k_n, \{ct\}))$

- Reconstruct$((k_1, \{ct\}), \ldots (k_t, \{ct\}))$: by construction, there is one $ct \in \{ct\}$ encrypted under $k_1, \ldots, k_t$, so decrypt accordingly

relies on computational assumptions

Shamir secret sharing [t out of n]: $\mathcal{M} = \mathbb{F}_p$, $S = \mathbb{F}_p^n$ (require $p > n$)

↳ beautiful construction based on polynomials (very useful mechanism for sharing data)



Any 2 points define a <u>unique</u> line

<u>Key idea</u>: Any d points uniquely define a degree-$(d-1)$ polynomial over a field

- e.g. 2 points define a line, 3 points define a parabola, etc.
- given d points, can <u>efficiently</u> obtain entire polynomial [Lagrange interpolation]

- Share(m): Choose $y_1, \dots, y_{t-1} \xleftarrow{R} \mathbb{F}_p$

  Let $f: \mathbb{F}_p \to \mathbb{F}_p$ be the <u>unique</u> polynomial of degree $t-1$

  $$f(0) = m \text{ and } f(i) = y_i \;\; \forall i \in [t-1] \qquad \text{[t points \underline{uniquely} define the polynomial } f\text{]}$$

  Output shares $(i, f(i))$ $\forall i \in [n]$ Each share is just 2 field elements (independent of threshold t or # parties n)

- Reconstruct$((i_1, y_1), \dots, (i_t, y_t))$: Interpolate the unique polynomial f of degree $(t-1)$ defined by the points $(i_1, y_1), \dots, (i_t, y_t)$

  Output $f(0)$

A little more detail... how to construct the polynomial f. Lagrange interpolation.

Let $(x_0, y_0), \dots, (x_t, y_t)$ be a collection of $t+1$ points. To find the polynomial f of degree t that interpolates these points, we can write

$$f(x) = a_0 + a_1 x + \cdots + a_t x^t, \text{ where } a_0, \dots, a_t \in \mathbb{F}_p$$

Then, we can write

$$\begin{aligned} f(x_0) &= a_0 + a_1 x_0 + \cdots + a_t x_0^t = y_0 \\ &\vdots \\ f(x_t) &= a_0 + a_1 x_t + \cdots + a_t x_t^t = y_t \end{aligned} \implies \underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^t \\ 1 & x_1 & x_1^2 & \cdots & x_1^t \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_t & x_t^2 & \cdots & x_t^t \end{bmatrix}}_{\text{"Vandermonde matrix" of dimension } t+1} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_t \end{bmatrix}$$

Interpolating a polynomial over $\mathbb{F}_p$ just corresponds to solving a linear system over $\mathbb{F}_p$. A unique solution exists as long as the Vandermonde matrix is invertible. It turns out that you can show (via linear algebra) that

$$\det \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^t \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_t & x_t^2 & \cdots & x_t^t \end{pmatrix} = \prod_{0 \le i < j \le t} (x_j - x_i)$$

[If x_i, x_j are all distinct, and we work over a finite field (e.g., there are no non-trivial divisors of 0), then this matrix is invertible and we can interpolate efficiently]

Let us now analyze the properties of Shamir's secret sharing scheme:

<u>Correctness</u>: Follows by uniqueness of interpolating polynomial (e.g., t shares uniquely define a polynomial of degree $t-1$)

<u>Security</u>: Given any subset of $(t-1)$ shares $(i_1, y_1), \dots, (i_{t-1}, y_{i-1})$, and any message $m \in \mathbb{F}_p$, there is a <u>unique</u> polynomial f of degree $t-1$ where

$$f(i_1) = y_1, \dots, f(i_{t-1}) = y_{i-1} \text{ and } f(0) = m$$

Thus, any $(t-1)$ shares can be consistent with secret-sharing of any message $m \implies$ information-theoretic security

<u>Efficiency</u>: Both share-generation and share reconstruction consist of polynomial evaluation and interpolation, both of which are efficiently computable (see above)

Shamir secret sharing is a <u>linear</u> secret sharing scheme: namely share-reconstruction is a <u>linear</u> function of the shares

- Suppose we have shares $(i_1, y_1) \dots (i_t, y_t)$ from a t-out-of-n secret sharing of a message m.
- Then, we can recover polynomial f of degree $t-1$ that interpolates these points by solving the Vandermonde system

$$Va = y \implies a = V^{-1}y \in \mathbb{F}_p^t \qquad [f(x) = a_0 + a_1 x + \dots + a_t x^t]$$

where V is the Vandermonde matrix (corresponding to share coefficients $i_1, \dots, i_t$), a is the vector of coefficients of f, and y is the vector of values $(y_1, \dots, y_t)$.

- Given coefficients a, notice that $m = f(0) = a_0 = e_1^T a$ where $e_1^T = [1, 0, 0, \dots, 0]$ is the first standard basis vector. This means that we can write

$$m = e_1^T a = (e_1^T V^{-1}) y$$

Oftentimes, we write $e_1^T V^{-1} = [\lambda_0, \lambda_1, \dots, \lambda_t]$, in which case

$$m = \sum_{i \in [t]} \lambda_i y_i,$$

which is a <u>linear</u> function of the shares of m.

Linear secret sharing is very useful for building threshold cryptosystems. Here, we describe one example with threshold BLS signatures:

$\text{Setup}(1^\lambda)$: $k \xleftarrow{R} \mathbb{Z}_p$ $\quad$ vk: (g, g^k)

$\qquad\qquad\qquad\qquad\qquad$ sk: k

$\text{Sign}(sk, m)$: Output $\sigma \leftarrow H(m)^k$

$\text{Verify}(vk, m, \sigma)$: Output 1 if $e(g, \sigma) = e(g^k, H(m))$.

Suppose we apply a t-out-of-n secret sharing to the signing key k. Using Shamir secret sharing, this yields shares $(x_1, y_1), \dots, (x_n, y_n)$.

this is itself a BLS signature, just with respect to the verification key g^{y_i}

Each signing party P_i has a share (x_i, y_i). To sign a message m, the party outputs $(x_i, H(m)^{y_i})$.

Suppose one has t shares of a signature on m:

$$(x_{i_1}, H(m)^{y_{i_1}}), \dots (x_{i_t}, H(m)^{y_{i_t}})$$

Shamir secret sharing has a linear reconstruction procedure, so given $x_{i_1}, \dots, x_{i_t}$, we can write the signing key as

$$k = e_1^T V_x y = \sum_{j \in [t]} \lambda_j y_{i_j}$$

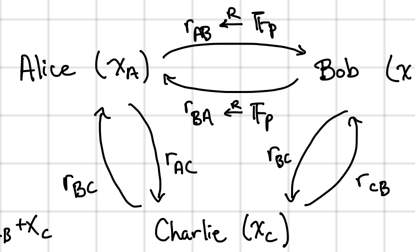
where V_x is the Vandermonde matrix associated with $(x_{i_1}, \dots, x_{i_t})$ and $y = (y_{i_1}, \dots, y_{i_t})$. Importantly, the Lagrange coefficients λ_j only depend on the x-coordinates. Thus, given $(x_{i_1}, H(m)^{y_{i_1}}), \dots, (x_{i_t}, H(m)^{y_{i_t}})$, one can compute

$$\prod_{j \in [t]} \left[H(m)^{y_{i_j}}\right]^{\lambda_j} = H(m)^{\sum_{j \in [t]} \lambda_j y_{i_j}} = H(m)^k,$$

which is a BLS signature on the message m under the signing key k.

gates are addition and multiplication over $\mathbb{F}_p$

Computing on secret-shared data: Another paradigm for 2PC (and MPC) - better-suited for evaluating arithmetic circuits



$f(x_A, x_B, x_C) = x_A + x_B + x_C$

Alice: Chooses $r_{AB}, r_{AC} \xleftarrow{R} \mathbb{F}_p$ and sends r_{AB} to Bob, r_{AC} to Charlie

Alice's share — Charlie's share

$\hookrightarrow$ **Observation:** $(x_A - r_{AB} - r_{AC},\ r_{AB},\ r_{AC})$ is additive secret sharing of Alice's input x_A

Bob's share

We will write $[x_A]$ to denote additive secret sharing of x_A

Computing on shares: Given shares of x_A and x_B,

$$[x_A + x_B] = [x_A] + [x_B] \qquad \text{(component-wise addition)}$$

Specifically if $[x_A] = (x_{A,1}, x_{A,2}, x_{A,3})$ where $x_{A,1} + x_{A,2} + x_{A,3} = x_A \in \mathbb{F}_p$

$[x_B] = (x_{B,1}, x_{B,2}, x_{B,3})$ where $x_{B,1} + x_{B,2} + x_{B,3} = x_B \in \mathbb{F}_p$

then $[x_A + x_B] = (x_{A,1} + x_{B,1},\ x_{A,2} + x_{B,2},\ x_{A,3} + x_{B,3})$ and $(x_{A,1} + x_{B,1}) + (x_{A,2} + x_{B,2}) + (x_{A,3} + x_{B,3}) = x_A + x_B \in \mathbb{F}_p$

More generally:
1. Share addition: $[x_A + x_B] = [x_A] + [x_B]$
2. Scalar multiplication: $[k x_A] = k \cdot [x_A]$
3. Addition by constant: $[x_A + k] = (x_{A,1} + k,\ x_{A,2},\ x_{A,3})$

Multiplication of secret-shared values is more challenging. We will first assume that parties have a "hint" – a secret sharing of a random multiplication tuple (idea due to Beaver — "Beaver multiplication triples"):

each party only has a share of a, b, c: no one knows actual values!

Suppose parties have a secret-sharing of a random product: $[a], [b], [c]$ where $c = ab \in \mathbb{F}_p$

$a, b \xleftarrow{R} \mathbb{F}_p$ (a, b are uniformly random values)

Then, given $[x]$ and $[y]$, we proceed as follows:
1. Each party computes $[x-a]$ and publishes their share of $x-a$
2. Each party computes $[y-b]$ and publishes their share of $y-b$
3. All of the parties compute non-interactively:

$$[z] = [c] + [x](y-b) + [y](x-a) - (x-a)(y-b)$$

Claim: $z = xy$. Follows by following calculation:

$$\begin{aligned} z &= c + x(y-b) + y(x-a) - (x-a)(y-b) \\ &= \cancel{ab} + xy - \cancel{bx} + \cancel{xy} - \cancel{ay} - \cancel{xy} + \cancel{bx} + \cancel{ay} - \cancel{ab} \\ &= xy \end{aligned}$$

Observe: Parties only see $x-a$ and $y-b$ in this protocol. Since a, b are uniformly random and unknown to the parties, $x-a$ and $y-b$ is a one-time pad encryption of x and y. Resulting protocol provides information-theoretic privacy for parties' inputs.

Assuming we have access to Beaver multiplication triples, we can evaluate any arithmetic circuit as follows (among n-parties):

1. Every party secret shares their input with every other party
2. For each addition gate in the circuit, parties locally compute on their shares
3. For each multiplication gate in the circuit, parties run Beaver's multiplication protocol (using <u>different</u> triple each time!)
4. Every party publishes share of the output; parties run share reduction to obtain output.

Where do Beaver triples come from?

- Generated by a trusted dealer (say, implemented using secure hardware like Intel SGX)
  - ↳ Notice that these are <u>random</u> multiplication triples and <u>input-independent</u> (the dealer does <u>not</u> see any party's secret inputs)
- Using oblivious transfers. Suppose the field is polynomial size (e.g. $p = \text{poly}(\lambda)$). We can use a 1-out-of-p^2 OT to generate a multiplication triple.

<u>sender</u>

$[a]_1, [b]_1, [c]_1 \xleftarrow{R} \mathbb{F}_p$

for $i, j \in \mathbb{F}_p$, let

$$m_{ij} = ([a]_1 + i)([b]_1 + j) - [c]_1 \in \mathbb{F}_p$$

<u>receiver</u>

$[a]_2, [b]_2 \xleftarrow{R} \mathbb{F}_p$

$$\xleftrightarrow{\text{OT for message } ([a]_2, [b]_2)}$$

By construction, receiver's message is $([a]_1 + [a]_2)([b_1] + [b_2]) - [c_1] \in \mathbb{F}_p$ and so $[a], [b], [c]$ is precisely a Beaver multiplication triple. Next, 1-out-of-p^2 OT can be implemented using $O(\log p)$ 1-out-of-2 OTs (via a tree-based construction), but communication grows with $O(p^2)$.

↳ Another method is to use Yao's garbled circuits to generate Beaver triple. Input is $[a]_1, [b]_1, [c]_1$ and $[a]_2, [b]_2$, and output is $[c]_2$. Communication now grows with polylog(p), so this method works even for superpolynomial p.

- Using somewhat homomorphic encryption:

<u>sender</u>

$[a]_1, [b]_1 \xleftarrow{R} \mathbb{F}_p$

$$\xrightarrow{pk,\ \text{Enc}(pk, [a]_1),\ \text{Enc}(pk, [b]_1)}$$

<u>receiver</u>

$[a]_2, [b]_2, [c]_2 \xleftarrow{R} \mathbb{F}_p$

$$\xleftarrow{\text{Enc}(pk, ([a]_1 + [a]_2)([b]_1 + [b]_2) - [c]_2)}$$

↓

decrypt to obtain $[c]_1$

In all these cases, Beaver triples can be generated in a <u>separate</u> "preprocessing" phase (before the parties come online and the inputs to the computation are know). $\boxed{\text{MPC with } \underline{\text{preprocessing}} \text{ model.}}$

↳ We can similarly preprocess oblivious transfers to reduce its online cost (see HW4).

MPC protocol comparison:

	Secret-Sharing (GMW)	Yao
Type of computation	Arithmetic circuits ($\mathbb{F}_p$)	Boolean circuits
Number of parties	Arbitrary (n)	2
Round complexity	Depth of circuit	2
Communication	$\sim 2n \log p$ bits per multiplication gate*	~ 256 bits per AND gate*
Security	Information-theoretic (with Beaver triples)	Computational

* Can be improved further!

* leverages several optimizations (half-gates + free XOR)

Main takeaway: OT is sufficient for n-party MPC (with security against all-but-1 corrupted parties)

↳ i.e., use OT to generate Beaver triples and compute on secret-shared values

"Anything that can be computed with a trusted party can be computed without, even if all but a single party is controlled by an adversary" [assuming OT]

Handling malicious parties: Protocols described so far are in the semi-honest model (parties agree to follow the protocol as written)

Many approaches to support malicious parties:

- Develop new protocols tailored for malicious security
- The GMW approach: take any protocol with security against semi-honest adversaries and have parties attach a zero-knowledge proof to each message to prove that they are following the protocol specification

↳ Theoretical view: Malicious security is no more difficult to achieve than semi-honest security

↳ Practice: Generic ZKPs are quite expensive, so oftentimes, other techniques will enable better efficiency

MPC without cryptographic assumptions? Information-theoretic MPC is also possible if we assume that there are sufficiently many honest parties.

- Honest majority: security against semi-honest adversaries
- Honest supermajority ($2/3$ honest): security against malicious adversaries

↳ Advantages: no cryptographic assumptions needed!

oftentimes very efficient since there are no cryptographic operations

Disadvantage: stronger trust assumption (need majority of parties to be honest)

↳ potentially quite reasonable in large-scale protocols (e.g., blockchains/Bitcoin make similar assumption on distribution of computational power)