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:

- **Share:** $\mathcal{M} \to \mathcal{S}^n$
- **Reconstruct:** $\mathcal{S}^t \to \mathcal{M}$

with the following properties:

1. **Correctness:** Any t shares can be used to reconstruct $\mathcal{M}$:
   
   $\forall m \in \mathcal{M} : (s_1, \ldots, s_n) \leftarrow \text{Share}(m) \Rightarrow \exists s \in \mathcal{M} : (s_1, \ldots, s_n) \leftarrow \text{Reconstruct}(S)$

2. **Security:** Need at least t shares to learn the secret:
   
   $\forall m_0, m_1 \in \mathcal{M}, \forall I \subseteq [n] \text{ where } |I| < t :$
   
   $\{(s_1, \ldots, s_n) \leftarrow \text{Share}(m_0) : s_i \text{ for all } i \in I\} \equiv \text{Nothing learned on } \mathcal{M}$
   
   $\Rightarrow$ notation here is information-theoretic.

**Examples:**

- **Additive secret sharing [n out of n]:** $\mathcal{M} = \mathbb{Z}_p^*, \mathcal{S} = \mathbb{Z}_p^n$
  
  - **Share(m):** Sample $r_1, \ldots, r_n \in \mathbb{Z}_p$ 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 $\mathbb{Z}_p^*$ (e.g., AES-CTR)

  - **Share(m):** Sample n keys $k_1, \ldots, k_n$ for encryption scheme
  
  For every $t$-subset $\{i_1, \ldots, i_t\} \subseteq \mathbb{Z}_p$ define
  
  - **Reconstruct(k_1, \ldots, k_t):** by construction, there is one ct $\in \mathcal{S}$ encrypted under $k_1, \ldots, k_t$, so decrypt accordingly

  
  Provides information-theoretic security — check this!

  Satisfying CPA-security

  Very large shares:

  Relies on computational assumptions
Shamir secret sharing [$k$ out of $n$]: $M = Tp$, $S = Tp^n$ (requires $p > n$)

- Key idea: Any $d$ points uniquely define a degree-$d$ polynomial over a field.
  - e.g., 2 points define a line, 3 points define a parabola, etc.
  - given $d$ points, can efficiently obtain entire polynomial (Lagrange interpolation)

- Share ($n$): Choose $y_1, ..., y_n \in Tp$
  
  Let $f: Tp \rightarrow Tp$ be the unique polynomial of degree $t-1$ such that:
  
  $f(0) = m_1$ and $f(i) = y_i \forall i \in [t-1]$ [it points uniquely define the polynomial $f$]

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

- Reconstruct $(x_1, y_1), ..., (x_t, y_t)$: Interpolate the unique polynomial $f$ of degree $t-1$ defined by the points $(i, y_i), ..., (i, y_t)$
  
  Output $f(0)$

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

Let $(x_1, y_1), ..., (x_t, y_t)$ be a collection of $t+1$ points. To find the polynomial $f$ of degree $t$ that interpolates these points, we can write

$$f(x) = a_0 + a_1 x + \cdots + a_t x^t,$$

where $a_0, ..., a_t \in Tp$

Then, we can write

$$f(x) = a_0 + a_1 x + \cdots + a_t x^t = y_0,$$

$$f(x) = a_0 + a_1 x + \cdots + a_t x^t = y_t,$$

So,

$$f(x) = a_0 + a_1 x + \cdots + a_t x^t = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^t \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_t & x_t^2 & \cdots & x_t^t \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_t \end{bmatrix}$$

"Vandermonde matrix" of dimension $t+1$

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

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

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:

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

Security: Given any subset of $(t-1)$ shares $(i_1, y_1), ..., (i_{t-1}, y_{t-1})$, and any message $m \in Tp$, there is a unique polynomial $f$ of degree $t-1$ where

$$f(i_1) = y_1, ..., f(i_{t-1}) = y_{t-1}$$

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

Efficiency: 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 linear secret sharing scheme: namely, share-reconstruction is a linear function of the shares.

- Suppose we have shares \((x_1, y_1), \ldots, (x_t, y_t)\) 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:

\[
V_i y = \Rightarrow a = V^{-1}_i y \in \mathbb{F}_p^t
\]

where \(V\) is the Vandermonde matrix (corresponding to share coordinates \(x_1, \ldots, x_t\)), \(a\) is the vector of coefficients of \(f\), and \(y\) is the vector of values \((y_1, \ldots, y_t)\).

- Given coefficients \(a\), notice that \(m = f(0) = a_0 = \mathbf{e}_1^T a\) where \(\mathbf{e}_1 = [1, 0, \ldots, 0]\) is the first standard basis vector. This means that we can write

\[
m = \mathbf{e}_1^T a = (\mathbf{e}_1^T V^{-1}) y
\]

Oftentimes, we write \(\mathbf{e}_1^T V^{-1} = [\lambda_0, \lambda_1, \ldots, \lambda_t]\), in which case

\[
m = \sum_{i=0}^t \lambda_i y_i,
\]

which is a linear 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:

\[
\text{Setup}(\mathbb{G}): k \in \mathbb{Z}_p \quad \text{vk}: (y_0, y_1) \quad \text{sk}: k
\]

\[
\text{Sign}(\text{sk}, m): \text{Output } \sigma = H(m)^k
\]

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

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

\((x_1, y_1), \ldots, (x_t, y_t)\).

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 \(m\):

\((x_1, H(m)^{y_1}), \ldots, (x_t, H(m)^{y_t})\)

Shamir secret sharing has a linear reconstruction procedure, so given \(x_1, \ldots, x_t\), we can write the signing key as

\[
k = \sum_{j=0}^t \lambda_j y_j
\]

where \(V\) is the Vandermonde matrix associated with \((x_1, \ldots, x_t)\) and \(y = (y_1, \ldots, y_t)\). Importantly, the Lagrange coefficients \(\lambda_j\) only depend on the \(x\)-coordinates. Thus, given \((x_1, H(m)^{y_1}), \ldots, (x_t, H(m)^{y_t})\), one can compute

\[
\prod_{j \in \{0, \ldots, t\}} H(m)^{y_1} \lambda_j y_j = H(m)^k
\]

which is a BLS signature on the message \(m\) under the signing key \(k\).
Computing on secret-shared data: Another paradigm for 2PC (and MPC) - better suited for evaluating arithmetic circuits.

Computing on shares: Given shares of $X_A$ and $X_B$,

$$[X_A + X_B] = [X_A] + [X_B] \quad \text{(component-wise addition)}$$

Specifically if $[X_A] = (X_{A1}, X_{A2}, X_{A3})$ where $X_{A1} + X_{A2} + X_{A3} = X_A \in \mathbb{F}_p$

$$[X_B] = (X_{B1}, X_{B2}, X_{B3}) \quad \text{where} \quad X_{B1} + X_{B2} + X_{B3} = X_B \in \mathbb{F}_p$$

then $[X_A + X_B] = (X_{A1} + X_{B1}, X_{A2} + X_{B2}, X_{A3} + X_{B3})$ and $(X_{A1} + X_{B1}) + (X_{A2} + X_{B2}) + (X_{A3} + X_{B3}) = X_A + X_B \in \mathbb{F}_p$

More generally:
1. Share addition: $[X_A + X_B] = [X_A] + [X_B]$
2. Scalar multiplication: $[kX_A] = k \cdot [X_A]$
3. Addition by constant: $[X_A + k] = (X_{A1} + k, X_{A2}, X_{A3})$

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”):

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

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

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

$$z = \frac{c + x \cdot (y \cdot b)}{a} = (x \cdot a) \cdot (y \cdot b)$$

$$= xy + b^2 + (x \cdot y) \cdot b - xy + b \cdot y + \frac{a}{b} \cdot b$$

$$= xy$$

Observe: Parties only see $x \cdot a$ and $y \cdot b$ in this protocol. Since $a, b$ are uniformly random and unknown to the parties, $x \cdot a$ and $y \cdot b$ 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 different 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 (e.g., implemented using secure hardware like Intel SGX)
  \(\Rightarrow\) Notice that these are random multiplication triples and input-independent (the dealer does not see any party’s secret inputs)
- Using oblivious transfers. Suppose the field is polynomial size (e.g. \(p = \text{poly}(n)\)). We can use a 1-out-of-\(p^2\) OT to generate a multiplication triple.

\[
\begin{align*}
\text{sender} & & \text{receiver} \\
[a], [b], [c] \xleftarrow{\$} F_p & & [a], [b], [c] \xleftarrow{\$} F_p \\
\text{for } i, j \in F_p, \text{ let} & & \Rightarrow \text{OT for message } ([a], [b], [c]) \\
M_{ij} = ([a] + i)([b] + j) - [c], & & \in F_p
\end{align*}
\]

By construction, receiver’s message is \(([a] + [b])([b] + [c]) - [c] \in F_p\) 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 \(O(p^3)\).

\(\Rightarrow\) Another method is to use Yao’s garbled circuits to generate Beaver triple. Input is \([a], [b], [c]\), and output is \([c]\). Communication now grows with \(\text{poly}(p)\), so this method works even for superpolynomial \(p\).
- Using somewhat homomorphic encryption:

\[
\begin{align*}
\text{sender} & & \text{receiver} \\
[a], [b] \xleftarrow{\$} F_p & & \text{Enc}(p, [c]) \\
\text{for } i, j \in F_p, \text{ let} & & T_{ik}, \text{Enc}(p, [a]) \\
\text{and } [a], [b], [c] \xleftarrow{\$} F_p & & \Rightarrow \text{OT for message } ([a], [b], [c]) \\
\text{Enc}(p, [a]) & & \text{Enc}(p, [a]) + [b] \xleftarrow{\$} F_p \\
\text{decrypt to obtain } [c]\end{align*}
\]

In all these cases, Beaver triples can be generated in a separate “preprocessing” phase (before the parties come online and the inputs to the computation are known). We can similarly preprocess oblivious transfers to reduce its online cost (see [foo]).
<table>
<thead>
<tr>
<th>MPC protocol comparison:</th>
<th>Secret-Share (GMW)</th>
<th>Yao</th>
<th><em>Can be improved further!</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of computation</td>
<td>Arithmetic circuits (F_p)</td>
<td>Boolean circuits</td>
<td>*Leverages several optimizations (half-gates + free XOR)</td>
</tr>
<tr>
<td>Number of parties</td>
<td>Arbitrary (n)</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Round complexity</td>
<td>Depth of circuit</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>&gt;2n log bits per multiplication + AND gate</td>
<td>&lt;25% bits per Xor</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>Information-theoretic (with Beaver triples)</td>
<td>Computational</td>
<td></td>
</tr>
</tbody>
</table>

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

Advantage: no cryptographic assumptions needed!

- Advantageous in 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)