CMP_SC 8001 - Introduction to Secure Multiparty Computation

Fundamental MPC Protocols - Part 1

Wei Jiang

Department of Electrical Engineering and Computer Science

University of Missouri

Outline

Outline

What is Zero-Knowledge (ZK) Proof

- **Completeness**: if the statement is true, a prover can convince an honest verifier that the statement is true
- **Soundness**: if the statement is false, a prover can convince an honest verifier to accept this fact with negligible probability
- **Zero-knowledge**: if the statement is true, no verifier can learn anything, except the fact that the statement is true

How to Show a Proof is ZK

- Similar to the read-ideal paradigm
- There exists a simulator for any verifier,
  - given only the statement to be proved
  - it can produce view indistinguishable from an interaction between the honest prover and the verifier

3-Colorable Graph

Definition

A graph $G = \langle V, E \rangle$ is 3-colorable if V can be colored with three different colors, such that for any two vertices v_i and v_j connected via an edge e_{ij}, $\mathrm{Color}(v_i) \neq \mathrm{Color}(v_j)$

Key Facts

- If G 3-colorable, permuting its three colors results another valid 3-coloring
- If G not 3-colorable, there exists at least a pair of adjacent vertices having the same color
- Graph 3-coloring is a NP-Complete problem

ZK-Proof for Graph 3-Coloring

- Public input:
  - $G = \langle V, E \rangle$
  - H: a secure commitment function
- Private input:
  - **Prover**: w, a 3-coloring of G
  - **Verifier**: $\perp$

ZK-Proof for Graph 3-Coloring

1. **Prover**:
   - randomly permute the 3 colors of w to produce w'
   - send $H(w')$ to the verifier, where $H(w') = \{H(v_i, \mathrm{Color}(v_i)) | \forall i, v_i \in V\}$
2. **Verifier**:
   - randomly select $e_{ij} \in E$ (or $(i,j) \in E$)
   - send e_{ij} or (i,j) to the prover
3. **Prover**: send $\langle v_i, \mathrm{Color}(v_i)\rangle$ and $\langle v_j, \mathrm{Color}(v_j)\rangle$ to the verifier
4. **Verifier**: if the commitments can be verified and $\mathrm{Color}(v_i) \neq \mathrm{Color}(v_j)$, return **accept**; otherwise, return **reject**

ZK-Proof for Graph 3-Coloring

- **Completeness**: if w is a valid 3-coloring of G, an honest verifier will always return **accept**
- **Soundness**:
  - if w is not a valid 3-coloring, the probability that an honest verifier returns **accept** is bounded by $\frac{|E|-1}{|E|}$ or $1 - \frac{1}{|E|}$
  - the above probability is also called **soundness error**

Making the Soundness Error Negligible

- The previous proof is not very sound:
  - the accept probability or the soundness error $1 - \frac{1}{|E|}$ is too high when w is not a valid 3-coloring
  - how to make the soundness error negligible?

Making the Soundness Error Negligible

- Run the above proof $n|E|$ times independently
- The verifier returns **accept** if all $n|E|$ executions returns accept
- Soundness error:
$$\left(1 - \frac{1}{|E|}\right)^{n|E|} \leq \frac{1}{e^n}$$
using the following inequality: $(1+x)^t \leq e^{tx}$ for any real number x and t with $t > 0$

Zero Knowledge

- To prove the previous proof is zero-knowledge, we need to build a simulator S
- Based on the public information, S generates a simulated view of the interaction between a prover (P) and a verifier (V)
- If the simulated view is computationally indistinguishable from the real interaction or execution of the proof, the proof is zero-knowledge when V is computationally bounded

Zero Knowledge - the Simulator

Simulator for graph 3-coloring

1. S randomly chooses an edge $e_{ij} \in E$ and colors v_i and v_j with different colors, and colors the rest the same color to produce $\hat{w}$. Then S commits $\hat{w}$, denoted by $H(\hat{w})$
2. S simulates V using $H(\hat{w})$, and receives (i', j'), the first message V sends
   - If $(i, j) = (i', j')$, then S can honestly answer the query and simulate the rest of the protocol, and outputs the transcript:

     $$\texttt{View}_S = \{H(\hat{w}), \langle v_i, \mathrm{Color}(v_i)\rangle, \langle v_j, \mathrm{Color}(v_j)\rangle, \texttt{accept}\}$$

   - If $(i, j) \neq (i', j')$, then S restarts from the beginning with a newly chosen (i, j)

Zero Knowledge - the Simulator

- Note that $\mathrm{Prob}((i,j) = (i',j')) \approx \frac{1}{|E|}$, since the selection of (i',j') is only based on the commitments, which cannot bias the decision due to the hiding property
- Thus, S succeeds with probability about $1/|E|$, and the expected number of iterations to terminate is $|E|$

Zero Knowledge - Real vs Ideal (Simulator)

- $\texttt{View}_\pi = \{H(w'), \langle v_i, \mathrm{Color}(v_i)\rangle, \langle v_j, \mathrm{Color}(v_j)\rangle, \texttt{accept}\}$
- $\texttt{View}_S = \{H(\hat{w}), \langle v_i, \mathrm{Color}(v_i)^*\rangle, \langle v_j, \mathrm{Color}(v_j)^*\rangle, \texttt{accept}\}$
- $\texttt{View}_\pi$ and $\texttt{View}_S$ are computationally indistinguishable:
  - $H(w')$ and $H(\hat{w})$ are computationally indistinguishable due to the hiding property of H
  - $\langle \mathrm{Color}(v_i), \mathrm{Color}(v_j)\rangle$, and $\langle \mathrm{Color}(v_i)^*, \mathrm{Color}(v_j)^*\rangle$ are identically distributed since the colors are randomly permuted for each execution of the proof

Hamiltonian Cycle

- Given a graph $G = \langle V, E \rangle$, a Hamiltonian cycle includes every vertex $v_i \in V$ exactly once
- The problem of finding a Hamiltonian cycle in G is known to be NP-Complete

ZK-Proof for Hamiltonian Cycle

- Public information: G is known to both Alice and Bob
- Private inputs:
  - **Alice**: $C \subseteq E$ a Hamiltonian cycle in G
  - **Bob**: $\perp$
- ZK-proof: Alice proves to Bob that she knows C without disclosing any information about it to Bob

ZK-Proof for Hamiltonian Cycle

1. Alice chooses a random permutation (on the vertices of G): $\pi(G) \to G'$, and sends $H(G')$ and $H(\pi)$, the commitments of G and π, to Bob
2. Bob randomly chooses $b \in \{0, 1\}$, and sends it to Alice
3. Alice performs the following, based on the challenge b:
   - $b = 0$: open G' and π
   - $b = 1$: compute $C' \leftarrow \pi(C)$ and open only the commitments related to C'
4. Bob returns accept if either verification below succeeds:
   - $b = 0$: verify the commitments and check if $G' = \pi(G)$
   - $b = 1$: verify the commitments related to C' and check if C' is a Hamiltonian cycle

Completeness

- If Alice does know a Hamiltonian cycle in G, she can easily satisfy Bob's either challenge:
  - the graph isomorphic mapping π producing G' from G, or
  - a Hamiltonian cycle C' in G' produced based on π

Soundness

- If Alice does not know C, she can guess which question Bob will ask to generate either
  - a graph isomorphic to G, or
  - a Hamiltonian cycle for an unrelated graph
- However, since she does not know a Hamiltonian cycle for G, she cannot do both
- Soundness error: $\frac{1}{2}$
- Similar to graph 3-coloring, the soundness error can be reduced to $\frac{1}{2^n}$ by executing the proof n times

Soundness

- Conversely, if Alice has prior knowledge about the challenge bit b, she can fool Bob without knowing a valid C
- If Alice knew $b = 0$, she would commit to an arbitrary permutation $\pi(G)$ and still pass the challenge
- If Alice knew $b = 1$, she would commit to a complete graph with $|G|$ vertices not a permutation of G, and she would then reveal any arbitrary Hamiltonian cycle on the complete graph to Bob

Zero Knowledge

- Alice's answers do not reveal the original Hamiltonian cycle C
  - Each round, Bob only learns G' is isomorphic to G or a Hamiltonian cycle in G'
  - He would need both answers for a single G' to discover the cycle C in G
- Thus, C remains unknown as long as Alice can generate a distinct G' every round

Zero Knowledge

- Prove there exists a probabilistic-polynomial time (PPT) simulator S for every PPT malicious verifier V^* such that
  - the output distribution of the interaction between S and each V^* is computationally indistinguishable from that of the interaction between each V^* and an honest prover P
- S predicts the challenge bit b' and commits either to a valid graph permutation or the complete graph with $|G|$ vertices with a trivial Hamiltonian cycle
- If the predicted challenge bit matches the actual challenge bit $b' = b$, then S proceeds by successfully responding to the challenge; otherwise, S rewinds the transcript and tries again

Simulator Complexity

- The probability of guessing b' correctly is $\frac{1}{2}$
- Thus, the expected number of iterations is 2
- In other words, the simulator runs in expected polynomial time

Outline

Division Theorem

Theorem (Division Algorithm)

If a and b are integers such that $b > 0$, then there are unique integers q and r such that $a = bq + r$, where $0 \leq r < b$

Linear Combination

Definition (Linear Combination)

If a and b are integers, then a linear combination of a and b is a sum of the form $ax + by$, where both x and y are integers

Example

- What are the linear combinations of $9x + 15y$?
- $-3 = 9 \cdot (-2) + 15 \cdot 1$
- $0 = 9 \cdot 0 + 15 \cdot 0$
- $3 = 9 \cdot 2 + 15 \cdot (-1)$
- It can be shown that the set of all linear combinations of 9 and 15 is $\{\ldots, -12, -9, -6, -3, 0, 3, 6, 9, 12, \ldots\}$

The Greatest Common Divisor

Definition (Greatest Common Divisor)

The greatest common divisor (gcd) of two integers a and b, not both zero, is the largest of the common divisors of a and b

Theorem (GCD as a Linear Combination)

The greatest common divisor of the integers a and b, not both 0, is the least positive integer that is a linear combination of a and b

Some Facts related to GCD and Divisibility

- **Fact 1**: $d|a$ and $d|b \implies d|(am + bn)$
- **Fact 2**: $d|a$ and $d|b \implies d|\gcd(a, b)$
- **Fact 3**: $\gcd(0, 0) = 0$ and $\gcd(a, 0) = |a|$

The Euclidean Algorithm

Theorem

For any non-negative integer a and any positive integer b,
$\gcd(a, b) = \gcd(b, a \bmod b)$

Algorithm 1 Euclid(a, b)

Require: a and b are non-negative integers

```
if b = 0 then
    return a
else
    return Euclid(b, a mod b)
end if
```

Find the gcd of 30 and 72

- First we use the division theorem to write:

$$72 = 2 \cdot 30 + 12$$

- The Euclidean theorem tells us that

$$\begin{aligned}
\gcd(72, 30) &= \gcd(30, 72 \bmod 30) = \gcd(30, 12) \\
30 &= 2 \cdot 12 + 6 \\
\gcd(30, 12) &= \gcd(12, 30 \bmod 12) = \gcd(12, 6) \\
12 &= 2 \cdot 6 + 0 \\
\gcd(12, 6) &= \gcd(6, 12 \bmod 6) = \gcd(6, 0) = 6
\end{aligned}$$

- $\gcd(72, 30) = 6$

The Extended Euclidean

Find the GCD of 801 and 154

$$\begin{aligned}
801 &= 5 \cdot 154 + 31 && (1)\\
154 &= 4 \cdot 31 + 30 && (2)\\
31 &= 1 \cdot 30 + 1 && (3)\\
30 &= 30 \cdot 1 + 0 && (4)\\
1 &= 1 \cdot 1 + 0 && (5)
\end{aligned}$$

- $\gcd(801, 154) = \gcd(1, 0) = 1$

The Extended Euclidean

Find the linear combination of $\gcd(801, 154) = 801 \cdot x + 154 \cdot y$

- Starting with the GCD based Equation (5):

$$1 = 1 \cdot 1 + 0 \cdot 0 \qquad (g = 1, x = 1, y = 0)$$

- Replace 0 in the above according to Equation (4):

$$1 = 30 \cdot 0 + 1 \cdot 1 \qquad (g = 1, x = 0, y = 1)$$

- Replace 1 in the above according to Equation (3):

$$1 = 31 \cdot 1 + 30 \cdot (-1) \qquad (g = 1, x = 1, y = -1)$$

- Replace 30 in the above according to Equation (2):

$$1 = 154 \cdot (-1) + 31 \cdot 5 \qquad (g = 1, x = -1, y = 5)$$

- Replace 31 in the above according to Equation (1):

$$1 = 801 \cdot 5 + 154 \cdot (-26) \qquad (g = 1, x = 5, y = -26)$$

The Extended Euclidean

- The algorithm terminates with $b = 0$ and $a = g$; thus, from these parameters, the linear combination for g is $g = g \cdot 1 + 0 \cdot 0$
- Starting from these coefficients $(x, y) = (1, 0)$, we can go backwards up the recursive calls
- We need to figure out how the coefficients x and y change during the transition from (a, b) to $(b, a \bmod b)$

The Extended Euclidean

- Assuming we found the coefficients (x', y') for $(b, a \bmod b)$

$$g = b \cdot x' + (a \bmod b) \cdot y'$$

- We want to find the pair (x, y) for (a, b):

$$g = a \cdot x + b \cdot y$$

- We can represent $a \bmod b$ as:

$$a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b$$

- Replacing this in the coefficient equation for (x', y') gives:

$$g = b \cdot x' + (a \bmod b) \cdot y' = b \cdot x' + \left(a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b\right) \cdot y'$$

The Extended Euclidean

- After rearranging and combining the terms, we have:

$$g = a \cdot y' + b \cdot \left(x' - y' \cdot \left\lfloor \frac{a}{b} \right\rfloor\right)$$

- As a result, the values of x and y are:

$$x = y'$$
$$y = x' - y' \cdot \left\lfloor \frac{a}{b} \right\rfloor$$

The Extended Euclidean

Algorithm 2 Extended_Euclid(a, b)

Require: a and b are non-negative integers

1: **if** $b = 0$ **then**
2: return $(a, 1, 0)$
3: **else**
4: (g', x', y') = Extended_Euclid(b, $a \bmod b$)
5: $(g, x, y) = (g', y', x' - \lfloor a/b \rfloor y')$
6: return (g, x, y)
7: **end if**

Group Definition

A set of objects G along with a binary operation $(\bullet)$ is called a group if the following four properties hold:

- **Closure**: If $a, b \in G$, then $c = a \bullet b \in G$
- **Associativity**: $(a \bullet b) \bullet c = a \bullet (b \bullet c)$
- **Identity element**: There exists a unique element e in G, such that for every $a \in G$, we have $a \bullet e = e \bullet a = a$
- **Inverse**: For every $a \in G$, there exists $b \in G$ such that $a \bullet b = e$

A group is **commutative** or **abelian** if for any two elements $a, b \in G$, we have $a \bullet b = b \bullet a$

Group Examples

- Integers $\mathbb{Z}$ is a group under addition $(+)$, and real numbers $\mathbb{R}$ with either addition $(+)$ or multiplication $(\times)$ operation is a group
- $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ with addition modulo n $(+, \bmod n)$ is a group where n is a positive integer:
  - $a, b \in \mathbb{Z}_n$, then $c = a + b \bmod n$ is also in $\mathbb{Z}_n$
  - The identity element is 0, and the inverse of a is $n - a$
  - $7, 15 \in \mathbb{Z}_{16}$, then $7 + 15 \bmod 16 = 6$

Group Examples

- $\mathbb{Z}_p^+ = \{1, \dots, p-1\}$ with multiplication modulo p ($\times$, mod p) is a group where p is a prime:
  - $a, b \in \mathbb{Z}_p^+$, then $c = a \times b \bmod p$ is also in $\mathbb{Z}_p^+$
  - The identity element is 1
- Given $a \in \mathbb{Z}_p^+$, find the inverse of a (denoted by a^{-1}):
  - $\gcd(a, p) = a \cdot x + p \cdot y$
  - $a^{-1} = x \bmod p$

Ring and Field

Definition (Ring)

A **ring** is a set of elements with two binary operations, addition $(+)$ and multiplication $(\times)$:

- It is an abelian group with identify element 0 under addition
- Its multiplication is associative $a \times (b \times c) = (a \times b) \times c$ and distributive over addition $a \times (b + c) = a \times b + a \times c$ and $(b + c) \times a = b \times a + c \times a$

A ring is commutative if $a \times b = b \times a$ for every a and b

Ring and Field

Definition (Field)

A ring is called a **field** if its elements, except for 0, form a commutative group under $\times$

- $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ with $(+, \bmod p)$ and $(\times, \bmod p)$ is a field where p is a prime:
  - The identity element is 0 under $(+, \bmod p)$
  - The identity element is 1 under $(\times, \bmod p)$

Outline

Common Notations

- $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$
- P_i: a participating party indexed by $i \in \{1, \ldots, m\}$
- $[v] = \{[v]^1, \ldots, [v]^m\}$: a value $v \in \mathbb{Z}_n$ is secretly shared among the parties where $[v]^i$ ($1 \leq i \leq m$) is the share held by P_i
- $[v]_t = \{[v]_t^1, \ldots, [v]_t^m\}$: a value v is secretly shared using a t-degree polynomial over a finite field among the parties

Secret Sharing in $\mathbb{Z}_n$

- Suppose there are two parties P_1 and P_2, and each has a private value α and β in $\mathbb{Z}_n$ respectively
- To secretly share $\alpha \in \mathbb{Z}_n$ between P_1 and P_2, P_1 performs the following steps:
  - randomly select r from $\mathbb{Z}_n$
  - set $[\alpha]^1 = r$ and $[\alpha]^2 = \alpha - r \bmod n$
  - send $[\alpha]^2$ to P_2
- β can be secretly shared similarly by P_2

Secure Addition $[\alpha + \beta]$

- To derive $[\alpha + \beta]$, each party adds its local shares; that is,
  - P_1: $[\alpha + \beta]^1 \leftarrow [\alpha]^1 + [\beta]^1 \bmod n$
  - P_2: $[\alpha + \beta]^2 \leftarrow [\alpha]^2 + [\beta]^2 \bmod n$
- We will omit the $\bmod n$ operation where the context is clear

Secure Multiplication between $[\alpha]$ and a Constant c to Result $[c \cdot \alpha]$

- $c \in \mathbb{Z}_n$ (or in $\mathbb{Z}_p$) is known to both parties
- To derive $[c \cdot \alpha]$, each party multiplies its local shares of α with c:
  - P_1: $[c \cdot \alpha]^1 \leftarrow c \cdot [\alpha]^1$
  - P_2: $[c \cdot \alpha]^2 \leftarrow c \cdot [\alpha]^2$

Secure Multiplication $[\alpha\beta]$

- Suppose $\chi = \alpha + u$ and $\gamma = \beta + v$, and we have

$$\chi\gamma = \alpha\beta + v\alpha + u\beta + uv$$

- It is easy to see that

$$\alpha\beta = \chi\gamma - \chi v - \gamma u + uv$$

- We perform multiplication of $\alpha\beta$ based on χ, γ, u and v

Secure Multiplication $[\alpha\beta]$

- To compute $[\alpha\beta]$, we follow the relation below:

$$\begin{aligned} [\alpha\beta]^1 &= \chi\gamma - \chi[v]^1 - \gamma[u]^1 + [uv]^1 \\ [\alpha\beta]^2 &= -\chi[v]^2 - \gamma[u]^2 + [uv]^2 \end{aligned}$$

- This implies that if both P_1 and P_2 know χ, γ, $[u]$, $[v]$ and $[uv]$, then they can derive $[\alpha\beta]$
- $\langle [u], [v], [uv] \rangle$ is called a Beaver triple

Secure Multiplication $[\alpha\beta]$

- To compute $[\alpha\beta]$, we need an additional party P_3
- The purpose of using P_3 is to generate the Beaver triple $\langle [u], [v], [uv] \rangle$ shared between P_1 and P_2
- From $[u]$ and $[v]$, P_1 and P_2 can collaboratively derive χ and γ
- Then $[\alpha\beta]$ can be easily derived by both parties as shown earlier

Secure Multiplication $[\alpha\beta]$

- Input: $\langle P_1, [\alpha]^1, [\beta]^1\rangle, \langle P_2, [\alpha]^2, [\beta]^2\rangle, \langle P_3, \perp\rangle$
- Output: $\langle P_1, [\alpha\beta]^1\rangle, \langle P_2, [\alpha\beta]^2\rangle$
- Domain: $\mathbb{Z}_p$

Secure Multiplication $[\alpha\beta]$ based on Beaver triple

1. P_3 //generate Beaver triples and send shares to P_1 and P_2
   (a) randomly choose u and v from $\mathbb{Z}_p$ and generate the shares $([u]^1, [u]^2)$, $([v]^1, [v]^2)$ and $([uv]^1, [uv]^2)$
   (b) send $[u]^1, [v]^1, [uv]^1$ to P_1 and $[u]^2, [v]^2, [uv]^2$ to P_2
2. P_1 //generate P_1's shares of $[\chi]$ and $[\gamma]$ and send them to P_2
   (a) $[\chi]^1 \leftarrow [\alpha]^1 + [u]^1$ and $[\gamma]^1 \leftarrow [\beta]^1 + [v]^1$
   (b) send $[\chi]^1$ and $[\gamma]^1$ to P_2
3. P_2 //generate P_2's shares of $[\chi]$ and $[\gamma]$ and send them to P_1
   (a) $[\chi]^2 \leftarrow [\alpha]^2 + [u]^2$ and $[\gamma]^2 \leftarrow [\beta]^2 + [v]^2$
   (b) send $[\chi]^2$ and $[\gamma]^2$ to P_1

Secure Multiplication $[\alpha\beta]$ based on Beaver triple

4. P_1 //reconstruct χ and γ and derive P_1's share of $[\alpha\beta]$
   (a) $\chi \leftarrow [\chi]^1 + [\chi]^2$ and $\gamma \leftarrow [\gamma]^1 + [\gamma]^2$
   (b) $[\alpha\beta]^1 \leftarrow \chi\gamma - \chi[v]^1 - \gamma[u]^1 + [uv]^1$
5. P_2 //reconstruct χ and γ and derive P_2's share of $[\alpha\beta]$
   (a) $\chi \leftarrow [\chi]^1 + [\chi]^2$ and $\gamma \leftarrow [\gamma]^1 + [\gamma]^2$
   (b) $[\alpha\beta]^2 \leftarrow -\chi[v]^2 - \gamma[u]^2 + [uv]^2$

Important Properties for Polynomial

Two fundamental properties of polynomial

1. A non-zero polynomial of degree t has at most t roots
2. Given $t+1$ pairs $(x_1, y_1), \ldots, (x_{t+1}, y_{t+1})$, with all the x_i distinct, there is a unique polynomial $\theta(x)$ of degree (at most) t such that $\theta(x_i) = y_i$ for $1 \leq i \leq t+1$

Lagrange Interpolation

- Given $t+1$ pairs $(x_1, y_1), \ldots, (x_{t+1}, y_{t+1})$, with all the x_i distinct, construct a polynomial $\theta(x)$ such that $\theta(x_i) = y_i$ for $1 \leq i \leq t+1$
- Let consider a simpler problem first:
  - Suppose $y_1 = 1$ and $y_i = 0$ for $2 \leq i \leq t+1$, what is $\theta(x)$?

Lagrange Interpolation

- Let $q(x) = (x - x_2)(x - x_3)\cdots(x - x_{t+1})$: a polynomial of degree t (the x_i's are constants, and x appears t times)
  - We have $q(x_i) = 0$, for $2 \leq i \leq t + 1$
  - $q(x_1) = (x_1 - x_2)(x_1 - x_3)\cdots(x_1 - x_{t+1})$, which is some constant not equal to 0
- Thus, we have $\theta(x) = q(x)/q(x_1)$

Lagrange Interpolation

- Let generalize the previous problem to any arbitrary index i: $y_i = 1$ and $y_j = 0$ for all $j \neq i$
- Define $\delta_i(x)$ the degree t polynomial that goes through these $t + 1$ points:

$$\delta_i(x) = \frac{\Pi_{j \neq i}(x - x_j)}{\Pi_{j \neq i}(x_i - x_j)}$$

- It is easy to verify that

$$\begin{aligned} y_i &= \delta_i(x_i) = 1 \\ y_j &= \delta_i(x_j) = 0, \forall j \neq i \end{aligned}$$

Lagrange Interpolation

Given $t+1$ points $(x_1, y_1), \ldots, (x_{t+1}, y_{t+1})$ where x_i's are distinct, find a t-degree polynomial $\theta(x)$ going through these points:

1. Construct the $t+1$ polynomials: $\delta_1(x), \ldots, \delta_{t+1}(x)$
2. $\theta(x) = \sum_{i=1}^{t+1} y_i \delta_i(x)$

Polynomial over a Finite Field

- Suppose the finite field is $\mathbb{Z}_p$
- The coefficients of $\theta(x)$ and all operations (e.g., addition, multiplication) are in $\mathbb{Z}_p$
- The two important properties of polynomial still hold as well as the Lagrange Interpolation:

$$\delta_i(x) = \frac{\Pi_{j\neq i}(x - x_j)}{\Pi_{j\neq i}(x_i - x_j)} \Rightarrow \delta_i(x) = \Pi_{j\neq i}(x - x_j)(\Pi_{j\neq i}(x_i - x_j))^{-1} \bmod p$$

where $(\Pi_{j\neq i}(x_i - x_j))^{-1}$ is the multiplicative inverse of $\Pi_{j\neq i}(x_i - x_j)$ in $\mathbb{Z}_p$

Shamir Secret Sharing

- When discussing additive secret sharing, we assumed each party has a private value and wants to secretly share it with the other party
- There are many variations of how a value is secretly shared among the participating parties
- Here we use another variation to illustrate the Shamir secret sharing scheme

Shamir Secret Sharing

- Suppose there is a dealer who wants to secretly share α and β among m parties $P_1, \ldots, P_m$
- In practice, the dealer could be one of the parties and secretly shares its private input for the subsequent MPC

Shamir Secret Sharing

- The dealer randomly generates two t-degree polynomials in $\mathbb{Z}_p$:
  - $\theta_\alpha(x) = a_t x^t + a_{t-1} x^{t-1} + \cdots + a_1 x + \alpha \bmod p$
  - $\theta_\beta(x) = b_t x^t + b_{t-1} x^{t-1} + \cdots + b_1 x + \beta \bmod p$
- Note that $\theta_\alpha(0) = \alpha$ and $\theta_\beta(0) = \beta$
- To generate the shares of α and β, the dealer does the following:
  - $[\alpha]_t^i = \theta_\alpha(i) \bmod p$, for $1 \leq i \leq m$
  - $[\beta]_t^i = \theta_\beta(i) \bmod p$, for $1 \leq i \leq m$
- We will omit the $\bmod p$ operation where the context is clear
- The dealer sends $[\alpha]_t^i$ and $[\beta]_t^i$ to P_i

Share Reconstruction

- To discover the original values α and β, at least $t+1$ parties need to pool their shares together
- Suppose $P_{j_1}, \ldots, P_{j_{t+1}}$ are $t+1$ parties with shares $[\alpha]_t^{j_1}, \ldots, [\alpha]_t^{j_{t+1}}$ where $j_1, \ldots, j_{t+1} \subset \{1, \ldots, m\}$ and $t < \frac{m}{2}$
- These parties can share their shares, and each can local reconstruct the polynomial $\theta_\alpha(x)$ using Lagrange interpolation on the $t+1$ points: $(j_1, [\alpha]_t^{j_1}), \ldots, (j_{t+1}, [\alpha]_t^{j_{t+1}})$
- Then compute $\theta_\alpha(0)$ to retrieve α, and β can be derived similarly

Secure Addition $[\alpha + \beta]_t$

- To derive $[\alpha + \beta]_t$, each party adds its local shares; that is,

  P_i: $[\alpha + \beta]_t^i \leftarrow [\alpha]_t^i + [\beta]_t^i$

- This works because adding the two local points (on the y-coordinates) giving a point $(i, [\alpha + \beta]^i)$ on $\theta_{\alpha+\beta}(x)$:

$$\begin{aligned}\theta_{\alpha+\beta}(x) &= \theta_\alpha(x) + \theta_\beta(x) \\ &= (a_t + b_t)x^t + \cdots + (a_1 + b_1)x + (\alpha + \beta)\end{aligned}$$

- As discussed previously, the parties need to have at least $t + 1$ points $(j_1, [\alpha + \beta]_t^{j_1}), \ldots, (j_{t+1}, [\alpha + \beta]_t^{j_{t+1}})$ to reconstruct $\theta_{\alpha+\beta}(x)$
  - To retrieve $\alpha + \beta$, set $\theta_{\alpha+\beta}(0) = \alpha + \beta$

Secure Multiplication between $[\alpha]_t$ and a Constant c to Result $[c \cdot \alpha]_t$

- $c \in \mathbb{Z}_n$ (or in $\mathbb{Z}_p$) is known to all parties
- To derive $[c \cdot \alpha]_t$, P_i multiplies its local shares of α with c:
  - P_i: $[c \cdot \alpha]_t^i \leftarrow c \cdot [\alpha]_t^i$
- This works because multiplying the local point (on the y-coordinates) with c giving a point $(i, [c \cdot \alpha]_t^i)$ on $\theta_{c \cdot \alpha}(x)$:

$$\begin{aligned} \theta_{c \cdot \alpha}(x) &= c \cdot \theta_\alpha(x) \\ &= (c \cdot a_t)x^t + \cdots + (c \cdot a_1)x + (c \cdot \alpha) \end{aligned}$$

Secure Multiplication $[\alpha\beta]_t$

- If the parties need to perform this multiplication once, then they can just simply multiple their local shares to produce a valid point $(i, [\alpha\beta]_t^i)$ on $\theta'_{\alpha\beta}(x)$
- Since $\theta'_{\alpha\beta}(x)$ has a degree of $2t$ and $t < \frac{m}{2}$, we cannot use these shares to perform additional secure multiplications
- Otherwise, the original values cannot be retrieved due to the degree of the polynomials (resulting from these additional secure multiplications) would be equal to or greater than m
- **Key challenge**: after each multiplication, transform $\theta'_{\alpha\beta}(x)$ a $2t$-degree polynomial to $\theta_{\alpha\beta}(x)$ a t-degree polynomial

Secure Multiplication $[\alpha\beta]_t$

- To compute $[\alpha\beta]_t$ and solve the previously mentioned technical challenge, there are several protocols
- In what follows, we present an efficient protocol (based on DN07) under the semi-honest adversary model

Secure Multiplication $[\alpha\beta]_t$ - Main Steps of DN07

1. Each party locally multiplies its shares resulting a point on $\theta'_{\alpha\beta}(x)$, and obliviously randomizes it using $[r]_{2t}$ to produce a point on $\theta'_{\alpha\beta+r}(x)$
   - r is a random value in $\mathbb{Z}_p$, not known to the parties
2. Then each party sends its randomized point (on $\theta'_{\alpha\beta+r}(x)$) to a designated party, say P_1
3. P_1 performs Lagrange interpolation on $2t+1$ points to retrieve $\alpha\beta + r$ and sends it to the other parties
4. Each party subtracts the randomness to obtain a point on $\theta_{\alpha\beta}(x)$

Secure Multiplication $[xy]_t$ - DN07

- Input: $\langle P_i, [\alpha]_t^i, [\beta]_t^i \rangle$, for $1 \leq i \leq m$
- Output: $\langle P_i, [\alpha\beta]_t^i \rangle$, for $1 \leq i \leq m$
- Domain: $\mathbb{Z}_p$ and $1 \leq t < \frac{m}{2}$

Secure Multiplication $[xy]_t$ - DN07

1. P_i, for $1 \leq i \leq m$:
   - Randomly generate r_i from $\mathbb{Z}_p$ and use Shamir secret sharing to secretly share r_i with a random $2t$-degree polynomial and a random t-degree polynomial
   - At the end of the previous step, the party has $[r_1]^i_{2t}, \ldots, [r_m]^i_{2t}$ and $[r_1]^i_t, \ldots, [r_m]^i_t$)
   - Derive $[r]^i_{2t} \leftarrow [r_1]^i_{2t} + \cdots + [r_m]^i_{2t}$ and $[r]^i_t \leftarrow [r_1]^i_t + \cdots + [r_m]^i_t$
   - Derive $[\alpha\beta + r]^i_{2t} \leftarrow [\alpha]^i_t[\beta]^i_t + [r]^i_{2t}$ and send it to P_1

Secure Multiplication $[xy]_t$ - DN07

2. P_1:
   - Reconstruct $\theta'_{\alpha\beta+r}(x)$ based on $2t+1$ pairs of $(i, [\alpha\beta+r]^i_{2t})$
   - Derive $\alpha\beta + r \leftarrow \theta'_{\alpha\beta+r}(0)$ and generate a random t-degree polynomial $\theta_{\alpha\beta+r}$ to secretly share $\alpha\beta + r$
   - Send $[\alpha\beta + r]^i_t$ to P_i
3. P_i, for $1 \leq i \leq m$:
   - Derive $[\alpha\beta]^i_t \leftarrow [\alpha\beta + r]^i_t - [r]^i_t$

Comparison Circuit

- Suppose we compare α and β, and both are four-bit numbers
  - $\alpha \equiv \langle \alpha_3, \alpha_2, \alpha_1, \alpha_0 \rangle$
  - $\beta \equiv \langle \beta_3, \beta_2, \beta_1, \beta_0 \rangle$
- α_3 and β_3 indicate the most significant bits of α and β

Comparison Circuit

1. Compute the bitwise xor of α and β
   - $a_3 = \alpha_3 \oplus \beta_3$
   - $a_2 = \alpha_2 \oplus \beta_2$
   - $a_1 = \alpha_1 \oplus \beta_1$
   - $a_0 = \alpha_0 \oplus \beta_0$
2. Let j be the most significant bit location where $\alpha_j \neq \beta_j$, set $b_3 = 0, \ldots, b_{j+1} = 0$ and $b_j = 1, \ldots, b_0 = 1$
   - $b_3 = a_3$
   - $b_2 = a_2 \vee b_3$
   - $b_1 = a_1 \vee b_2$
   - $b_0 = a_0 \vee b_1$

Comparison Circuit

3. Let j be the most significant bit location where $\alpha_j \neq \beta_j$, set $c_j = 1$ and $c_i = 0$ where $i \neq j$
   - $c_3 = b_3$
   - $c_2 = b_2 \oplus b_3$
   - $c_1 = b_1 \oplus b_2$
   - $c_0 = b_0 \oplus b_1$
4. Multiple c and α bitwise
   - $d_3 = c_3 \wedge \alpha_3$
   - $d_2 = c_2 \wedge \alpha_3$
   - $d_1 = c_1 \wedge \alpha_2$
   - $d_0 = c_0 \wedge \alpha_1$

Comparison Circuit

5. Derive the comparison result
   - $e_2 = d_2 \vee d_3$
   - $e_1 = d_1 \vee e_2$
   - $e_0 = d_0 \vee e_1$

Comparison Circuit

Key Obervation

The comparison result is stored in e_0

- $e_0 = 1 \rightarrow \alpha > \beta$
- $e_0 = 0 \rightarrow \alpha \leq \beta$

Compute $\oplus$, $\vee$ and $\wedge$ in terms of $-$, $+$ and $\times$

- $x \oplus y \equiv x + y - 2xy$
- $x \vee y \equiv x + y - xy$
- $x \wedge y \equiv xy$

Since we know how to compute $-$, $+$ and $\times$ securely, we can evaluate the comparison circuit securely

Comparison Circuit

Secure Implementation of Step 1 of the Boolean Circuit

- $a_i = \alpha_i \oplus \beta_i$
- $a_i = \alpha_i + \beta_i - 2\alpha_i\beta_i$

The main steps with inputs $[\alpha_i]$ and $[\beta_i]$

1. $[\alpha_i\beta_i] \leftarrow \text{Secure\_Multiplication}([\alpha_i], [\beta_i])$
2. $[2\alpha_i\beta_i] \leftarrow 2[\alpha_i\beta_i]$
3. $[a_i] \leftarrow [\alpha_i] + [\beta_i] - [2\alpha_i\beta_i]$

Acknowledgment

- Adi Shamir, "How to share a secret", Communications of the ACM, 22:612 - 613, 1979
- Donald Beaver, "Efficient multiparty protocols using circuit randomization", In Advances in Cryptology - (CRYPTO 91), volume 576 of Lecture Notes in Computer Science, pages 420 - 432, 1991
- Ivan Damgard and Jesper Buus Nielsen, "Robust multiparty computation with linear communication complexity", In Alfred Menezes, editor, Advances in Cryptology (CRYPTO 2007), Lecture Notes in Computer Science. Springer-Verlag, 2007