Secure Multi-Party Computation

Patrick Ah-Fat

Imperial College London

pwa14@ic.ac.uk

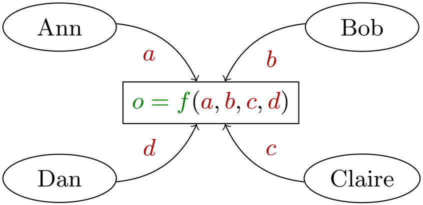
Overview

SMC:

- Compute $f(a, b, c, d)$
- Inputs are private
- No trusted party

Protocols:

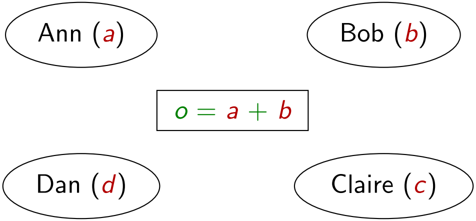
- Yao's garbled circuits (OT)
- Secret sharing schemes



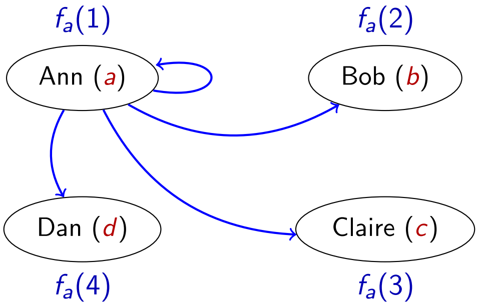
Security: During the protocol, nothing leaks about a, b, c and d *(apart from the public output o)*.

Example: Yao's Millionaires' problem

Question: How to *securely* compute $a + b$? (with $\leq t$ passive attackers)
Hint: Secret sharing

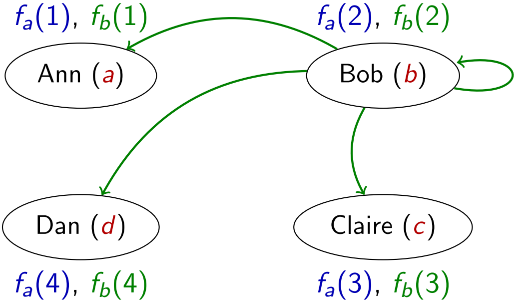


Question: How to *securely* compute $a + b$? (with $\leq t$ passive attackers)
Hint: Secret sharing



- Ann secret shares a: she distributes $[a, f_a]_t$.

Question: How to *securely* compute $a + b$? (with $\leq t$ passive attackers)
Hint: Secret sharing



- Ann secret shares a: she distributes $[a, f_a]_t$.
- Bob secret shares b: he distributes $[b, f_b]_t$.

Question: How to *securely* compute $a + b$? (with $\leq t$ passive attackers)
Hint: Secret sharing

$f_a(1) + f_b(1)$

Ann (a)

$f_a(2) + f_b(2)$

Bob (b)

Dan (d)

$f_a(4) + f_b(4)$

Claire (c)

$f_a(3) + f_b(3)$

- Ann secret shares a: she distributes $[a, f_a]_t$.
- Bob secret shares b: he distributes $[b, f_b]_t$.
- Parties **locally** add their shares.

Question: How to *securely* compute $a + b$? (with $\leq t$ passive attackers)
Hint: Secret sharing

$f_a(1) + f_b(1) = (f_a + f_b)(1)$ $\quad$ $f_a(2) + f_b(2) = (f_a + f_b)(2)$

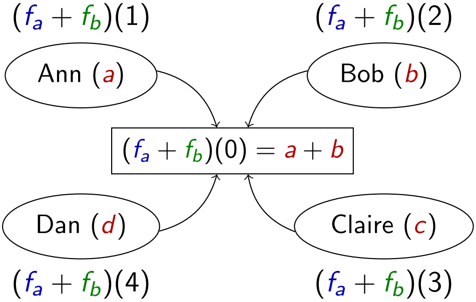
Ann (a) $\quad$ Bob (b)

Dan (d) $\quad$ Claire (c)

$f_a(4) + f_b(4) = (f_a + f_b)(4)$ $\quad$ $f_a(3) + f_b(3) = (f_a + f_b)(3)$

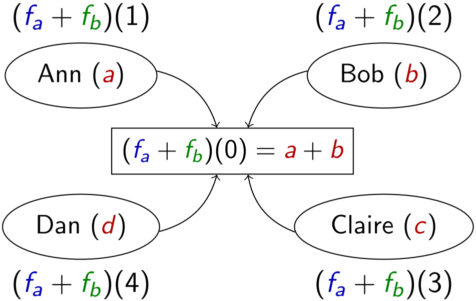
- Ann secret shares a: she distributes $[a, f_a]_t$.
- Bob secret shares b: he distributes $[b, f_b]_t$.
- Parties **locally** add their shares.
- They now share $[a + b, f_a + f_b]_t$!

Question: How to *securely* compute $a + b$? (with $\leq t$ passive attackers)
Hint: Secret sharing



- Ann secret shares a: she distributes $[a, f_a]_t$.
- Bob secret shares b: he distributes $[b, f_b]_t$.
- Parties **locally** add their shares.
- They now share $[a + b, f_a + f_b]_t$, and can **securely** recover $a + b$!

Question: How to *securely* compute $a + b$? (with $\leq t$ passive attackers)
Hint: Secret sharing



- Ann secret shares a: she distributes $[a, f_a]_t$.
- Bob secret shares b: he distributes $[b, f_b]_t$.
- Parties **locally** add their shares ($[a, f_a]_t + [b, f_b]_t = [a + b, f_a + f_b]_t$).
- They now share $[a + b, f_a + f_b]_t$, and can **securely** recover $a + b$!

Notation: When parties share a secret s via polynomial f of degree at most t, i.e. $[s, f]_t$, we write $[s, f(1), f(2), f(3)]_t$ to make shares explicit.

Example: Let us place ourselves in $\mathbb{F} = \mathbb{Z}_{11}$. Let $\mathcal{P} = \{P_1, P_2, P_3\}$.
Aim: P_1 holds secret 4 and P_2 holds secret 7 and they want to securely compute $a + b$, while allowing up to 1 passive adversary.
Means: Use SMC on Shamir scheme with polynomials of degree at most 1

P_1 generates $f_a = 4 + 3X$ and distributes $[4, 7, 10, 2]_1$.
P_2 generates $f_a = 7 + 10X$ and distributes $[7, 6, 5, 4]_1$.
Parties locally add their shares to get $[0, 2, 4, 6]_1$.
They now share $[0, f_a + f_b = 0 + 2X]_1$.

Note: Passive attacker learns nothing about secrets given his share only. Parties can together reconstruct $a + b$ by opening their shares and using Lagrange interpolation on any set of more than 1 party.

Let γ be a **public** constant. Ann holds private input a.
Question: How to *securely* compute $\gamma \cdot a$? (with $\leq t$ passive attackers)

Let γ be a **public** constant. Ann holds private input a.
Question: How to *securely* compute $\gamma \cdot a$? (with $\leq t$ passive attackers)

- Ann secret shares a: she distributes $[a, f_a]_t$.
- Each party P_i **locally** computes $\gamma \cdot f_a(i)$ ($\gamma \cdot [a, f_a]_t = [\gamma \cdot a, \gamma \cdot f_a]_t$).
- Parties now share $[\gamma \cdot a, \gamma \cdot f_a]_t$ and can recover $\gamma \cdot a$.

Let γ be a **public** constant. Ann holds private input a.
Question: How to *securely* compute $\gamma \cdot a$? (with $\leq t$ passive attackers)

- Ann secret shares a: she distributes $[a, f_a]_t$.
- Each party P_i **locally** computes $\gamma \cdot f_a(i)$ ($\gamma \cdot [a, f_a]_t = [\gamma \cdot a, \gamma \cdot f_a]_t$).
- Parties now share $[\gamma \cdot a, \gamma \cdot f_a]_t$ and can recover $\gamma \cdot a$.

Note: By learning $\gamma \cdot a$ we can deduce secret input a, but a can actually represent any secret shared value, which will be useful later.

Ann holds secret input a and Bob holds secret input b.
Question: How to *securely* compute $a \cdot b$? (with $\leq t$ passive attackers)

Ann holds secret input a and Bob holds secret input b.
Question: How to *securely* compute $a \cdot b$? (with $\leq t$ passive attackers)

Would the following protocol work ?

- Ann secret shares a: she distributes $[a, f_a]_t$.
- Bob secret shares b: he distributes $[b, f_b]_t$.
- Each party P_i locally computes $f_a(i) \cdot f_b(i)$.
- Parties now share $[a \cdot b, f_a \cdot f_b]_{??}$. **Can they recover** $a \cdot b$?

Ann holds secret input a and Bob holds secret input b.
Question: How to *securely* compute $a \cdot b$? (with $\leq t$ passive attackers)

Would the following protocol work ?

- Ann secret shares a: she distributes $[a, f_a]_t$.
- Bob secret shares b: he distributes $[b, f_b]_t$.
- Each party P_i locally computes $f_a(i) \cdot f_b(i)$.
- Parties now share $[a \cdot b, f_a \cdot f_b]_{\mathbf{2t}}$. **Can they recover** $a \cdot b$?

Problems:

- We need $2t + 1$ parties to recover $a \cdot b$ from $[a \cdot b, f_a \cdot f_b]_{\mathbf{2t}}$.

Ann holds secret input a and Bob holds secret input b.
Question: How to *securely* compute $a \cdot b$? (with $\leq t$ passive attackers)

Would the following protocol work ?

- Ann secret shares a: she distributes $[a, f_a]_t$.
- Bob secret shares b: he distributes $[b, f_b]_t$.
- Each party P_i locally computes $f_a(i) \cdot f_b(i)$.
- Parties now share $[a \cdot b, f_a \cdot f_b]_{\mathbf{2t}}$. **Can they recover** $a \cdot b$?

Problems:

- We need $2t + 1$ parties to recover $a \cdot b$ from $[a \cdot b, f_a \cdot f_b]_{\mathbf{2t}}$.
- It's okay if we set $t < n/2$. But how about computing $a \cdot b \cdot c$?

Ann holds secret input a and Bob holds secret input b.
Question: How to *securely* compute $a \cdot b$? (with $\leq t$ passive attackers)

Would the following protocol work ?

- Ann secret shares a: she distributes $[a, f_a]_t$.
- Bob secret shares b: he distributes $[b, f_b]_t$.
- Each party P_i locally computes $f_a(i) \cdot f_b(i)$.
- Parties now share $[a \cdot b, f_a \cdot f_b]_{\mathbf{2t}}$. **Can they recover $a \cdot b$?**

Problems:

- We need $2t + 1$ parties to recover $a \cdot b$ from $[a \cdot b, f_a \cdot f_b]_{\mathbf{2t}}$.
- It's okay if we set $t < n/2$. But how about computing $a \cdot b \cdot c$? Number of needed parties keeps growing !

Aim: We need a protocol for letting parties share $[a \cdot b, h]_{\mathbf{t}}$, where polynomial h has degree $\leq t$.

Let:

- $t < n/2$ be the maximal number of passive attackers.
- all parties in $Z = \{P_1, \cdots, P_n\}$ secret share $[a, f_a]_t$ and $[b, f_b]_t$.
- $r = (r_k)_{k \in Z}$ be the recombination vector introduced earlier, such that $P(0) = \sum_{k \in Z} r_k P(k)$ for all polynomial P of degree lower than $|Z|$.

Protocol Multiplication

Let:

- $t < n/2$ be the maximal number of passive attackers.
- all parties in $Z = \{P_1, \cdots, P_n\}$ secret share $[a, f_a]_t$ and $[b, f_b]_t$.
- $r = (r_k)_{k \in Z}$ be the recombination vector introduced earlier, such that $P(0) = \sum_{k \in Z} r_k P(k)$ for all polynomial P of degree lower than $|Z|$.

Protocol Multiplication

- Each party P_k locally computes the product $f_a(k) \cdot f_b(k) = (f_a f_b)(k)$. Parties now share $[ab, f_a f_b]_{2t}$.

Let:

- $t < n/2$ be the maximal number of passive attackers.
- all parties in $Z = \{P_1, \cdots, P_n\}$ secret share $[a, f_a]_t$ and $[b, f_b]_t$.
- $r = (r_k)_{k \in Z}$ be the recombination vector introduced earlier, such that $P(0) = \sum_{k \in Z} r_k P(k)$ for all polynomial P of degree lower than $|Z|$.

Protocol Multiplication

- Each party P_k locally computes the product $f_a(k) \cdot f_b(k) = (f_a f_b)(k)$. Parties now share $[ab, f_a f_b]_{2t}$.
- Each party P_k secretly shares his own share $(f_a f_b)(k)$ via a new polynomial g_k of degree at most t: he distributes $[(f_a f_b)(k), g_k]_t$.

Let:

- $t < n/2$ be the maximal number of passive attackers.
- all parties in $Z = \{P_1, \cdots, P_n\}$ secret share $[a, f_a]_t$ and $[b, f_b]_t$.
- $r = (r_k)_{k \in Z}$ be the recombination vector introduced earlier, such that $P(0) = \sum_{k \in Z} r_k P(k)$ for all polynomial P of degree lower than $|Z|$.

Protocol Multiplication

- Each party P_k locally computes the product $f_a(k) \cdot f_b(k) = (f_a f_b)(k)$. Parties now share $[ab, f_a f_b]_{2t}$.
- Each party P_k secretly shares his own share $(f_a f_b)(k)$ via a new polynomial g_k of degree at most t: he distributes $[(f_a f_b)(k), g_k]_t$.
- Parties invoke *addition* and *multiplication by constant* to compute:

$$\sum_{k \in Z} r_k [(f_a f_b)(k), g_k]_t$$

Protocol Multiplication

- Each party P_k locally computes the product $f_a(k) \cdot f_b(k) = (f_a f_b)(k)$. Parties now share $[ab, f_a f_b]_{2t}$.
- Each party P_k secretly shares his own share $(f_a f_b)(k)$ via a new polynomial g_k of degree at most t: he distributes $[(f_a f_b)(k), g_k]_t$.
- Parties invoke *addition* and *multiplication by constant* to compute:

$$\sum_{k \in Z} r_k [(f_a f_b)(k), g_k]_t$$

By definition of r and since $\deg(f_a f_b) < |Z|$, we have:

$$\sum_{k \in Z} r_k (f_a f_b)(k) = (f_a f_b)(0) = a \cdot b$$

Protocol Multiplication

- Each party P_k locally computes the product $f_a(k) \cdot f_b(k) = (f_a f_b)(k)$. Parties now share $[ab, f_a f_b]_{2t}$.
- Each party P_k secretly shares his own share $(f_a f_b)(k)$ via a new polynomial g_k of degree at most t: he distributes $[(f_a f_b)(k), g_k]_t$.
- Parties invoke *addition* and *multiplication by constant* to compute:

$$\sum_{k \in Z} r_k [(f_a f_b)(k), g_k]_t$$

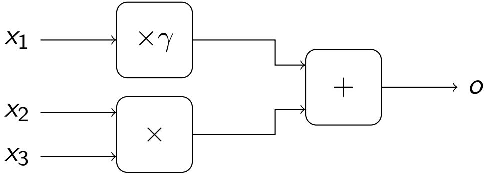
By definition of r and since $\deg(f_a f_b) < |Z|$, we have:

$$\sum_{k \in Z} r_k (f_a f_b)(k) = (f_a f_b)(0) = a \cdot b$$

Let $h = \sum_{k \in Z} r_k g_k$. We have $\deg(h) \leq t$. **Conclusion:** Parties share:

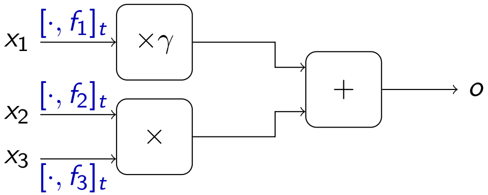
$$[\sum_{k \in Z} r_k (f_a f_b)(k), \sum_{k \in Z} r_k g_k]_t = [a \cdot b, h]_t$$

Question: How to compute an arithmetic function of n inputs ?
Example: $f(x_1, x_2, x_3) = \gamma * x_1 + x_2 * x_3$?



Protocol SMC with passive security

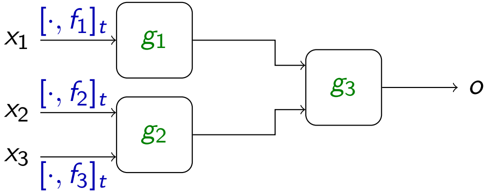
Question: How to compute an arithmetic function of n inputs ?
Example: $f(x_1, x_2, x_3) = \gamma * x_1 + x_2 * x_3$?



Protocol SMC with passive security

- Each party P_k secretly shares his input x_k via $[x_k, f_k]_t$.

Question: How to compute an arithmetic function of n inputs ?
Example: $f(x_1, x_2, x_3) = \gamma * x_1 + x_2 * x_3$?

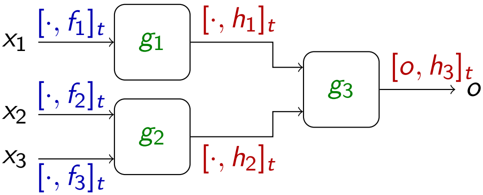


Protocol SMC with passive security

- Each party P_k secretly shares his input x_k via $[x_k, f_k]_t$.
- Sort all m gates such that $\forall j \geq i \implies$ no output of g_j is input of g_i (from left to right).

Question: How to compute an arithmetic function of n inputs ?

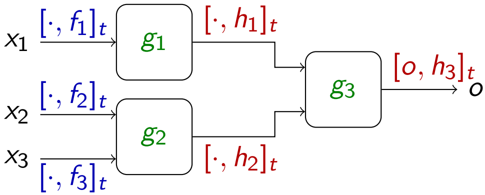
Example: $f(x_1, x_2, x_3) = \gamma * x_1 + x_2 * x_3$?



Protocol SMC with passive security

- Each party P_k secretly shares his input x_k via $[x_k, f_k]_t$.
- Sort all m gates such that $\forall j \geq i \implies$ no output of g_j is input of g_i (from left to right).
- For each ordered gate g_q, invoke secure operation to compute $[\cdot, h_q]_t$.

Question: How to compute an arithmetic function of n inputs ?
Example: $f(x_1, x_2, x_3) = \gamma * x_1 + x_2 * x_3$?



Protocol SMC with passive security

- Each party P_k secretly shares his input x_k via $[x_k, f_k]_t$.
- Sort all m gates such that $\forall j \geq i \implies$ no output of g_j is input of g_i (from left to right).
- For each ordered gate g_q, invoke secure operation to compute $[\cdot, h_q]_t$.
- Output recovery: Parties now share $[o, h_m]_t$ and can recover o !

Example: Let us place ourselves in $\mathbb{F} = \mathbb{Z}_{11}$. Let $\mathcal{P} = \{P_1, P_2, P_3\}$.
Aim: P_1 holds secret 4 and P_2 holds secret 7 and they want to securely compute $a \cdot b$, while allowing up to 1 passive adversary.

Step 1: Let us first compute the recombination vector $r = (r_1, r_2, r_3)$ that ensures that $\sum_{k=1}^{3} r_k \cdot f(k) = f(0)$ for all polynomial f of degree at most 2.
By setting $S = \{1, 2, 3\}$, r is defined as (Equation (1) from the notes):

$$r_1 = \frac{-2}{1-2} \cdot \frac{-3}{1-3} = 3$$
$$r_2 = \frac{-1}{2-1} \cdot \frac{-3}{2-3} = 8$$
$$r_3 = \frac{-1}{3-1} \cdot \frac{-2}{3-2} = 1$$

Example: Let us place ourselves in $\mathbb{F} = \mathbb{Z}_{11}$. Let $\mathcal{P} = \{P_1, P_2, P_3\}$.
Aim: P_1 holds secret 4 and P_2 holds secret 7 and they want to securely compute $a \cdot b$, while allowing up to 1 passive adversary.

Step 1: We have $r = (3, 8, 1)$.
Step 2: Let us now start the multiplication protocol.
P_1 generates $f_a = 4 + 3X$ and distributes $[4, 7, 10, 2]_1$.
P_2 generates $f_a = 7 + 10X$ and distributes $[7, 6, 5, 4]_1$.
Parties locally multiply their shares to get $[6, 9, 6, 8]_2$.
So parties share $[6, f_a \cdot f_b = 6 + 6X + 8X^2]_2$, now starts degree reduction.

Example: Let us place ourselves in $\mathbb{F} = \mathbb{Z}_{11}$. Let $\mathcal{P} = \{P_1, P_2, P_3\}$.
Aim: P_1 holds secret 4 and P_2 holds secret 7 and they want to securely compute $a \cdot b$, while allowing up to 1 passive adversary.

Step 1: We have $r = (3, 8, 1)$.
Step 2: Parties share $[6, 9, 6, 8]_2$.
Step 3: Degree reduction
P_1 generates $g_1 = 9 + 10X$ and distributes $[9, 8, 7, 6]_1$.
P_2 generates $g_2 = 6 + X$ and distributes $[6, 7, 8, 9]_1$.
P_3 generates $g_3 = 8 + 0X$ and distributes $[8, 8, 8, 8]_1$.
Each party P_j now **locally** computes $\sum_{k=1}^{3} r_k g_k(j)$:

$$3 * [9, 8, 7, 6]_1 + 8 * [6, 7, 8, 9]_1 + 1 * [8, 8, 8, 8]_1 = [6, 0, 5, 10]_1$$

Thus, parties now share secret $4 * 7 = 6$ with polynomial of degree at most 1, which can be shown to equal $\sum_{k=1}^{3} r_k \cdot g_k = 6 + 5X$.

- Compute a function on **private inputs** without trusted party
- Efficient scheme based on **secret sharing**
- Addition and multiplication by constants are **trivial** (no communication: only local computations)
- Multiplication require **communication** and protocol
- Arithmetic functions are decomposed into elementary operations