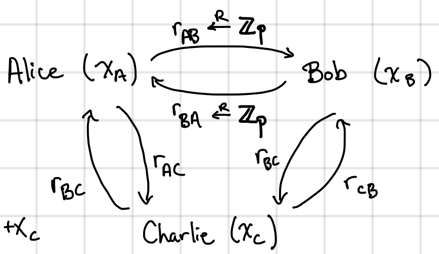
Computing on secret-shared data: Another paradigm for 2PC (and MPC) – better-suited for evaluating arithmetic circuits



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

Alice's share   Charlie's share

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

↳ **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,

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

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

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

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

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 $c = ab \in \mathbb{Z}_p$

$a, b \xleftarrow{R} \mathbb{F}_p$ (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 p is small (i.e., polynomial). We can use a 1-out-of-p^2 OT to generate a multiplication triple.

<u>sender</u>

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

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>

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

$\longleftarrow$ OT for message $([a]_2, [b]_2)$ $\longrightarrow$

By construction, receiver's message is $(([a]_1 + [a]_2)([b_1] + [b_2])) - [c_1] \in \mathbb{Z}_p$ 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.

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). $\boxed{\text{MPC with preprocessing model.}}$

<u>MPC protocol comparison:</u>

	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	~$2n \log p$ bits per multiplication gate*	~256 bits per AND gate
Security	Information-theoretic (with Beaver triples)	Computational

*Can be improved further!

Wrap-up: **This course:** secret-key cryptography

↳ parties have a **shared** secret key

↳ gold standard: authenticated encryption (confidentiality + integrity) ← AES-GCM

public-key cryptography

↳ mechanisms to negotiate a shared key

↳ main primitives: authenticated key exchange ← TLS 1.3, digital signatures ← ECDSA, Schnorr

} **protecting communication**

multiparty computation: protecting **computations**

↳ oblivious transfer ⟹ general MPC

↳ "Anything that can be computed with a trusted party can be computed **without**!"

What's next? Encryption schemes with more **flexibility**:

- Homomorphic encryption: computations on ciphertexts

$$[\,\mathrm{Enc}(pk, x) \longrightarrow \mathrm{Enc}(pk, f(x))$$

- Identity-based encryption: public keys can be **arbitrary** strings (e.g., a email address)
- Functional encryption: decryption outputs arbitrary **function** of the message

$$[\,\mathrm{Dec}(sk_f, \mathrm{Enc}(pk, x)) \longrightarrow f(x)\,]$$

New cryptographic assumptions:

- Pairing-based cryptography: exploiting additional structure on elliptic curves to enable "multiplication in the exponent" — 20th century mathematics!
- Lattice-based cryptography: cryptography with **post-quantum** security

Advanced cryptographic protocols:

- Private information retrieval (PIR): reading elements from a database without revealing query
- Differential privacy: protecting sensitive **inputs** to computations
- Succinct proofs: mimizing the **size** of a proof