Secure Multiparty Computation (MPC)

\[ f(x_1, x_2, x_3, x_4) = (y_1, y_2, y_3, y_4) \]

**Motivation:**
- framework for computation between parties who do not trust each other

**Examples:**
- elections, auctions, distributed data mining, database privacy:
  - evaluate a query on the database without revealing the query to the database owner,
  - evaluate a statistical query on the database without revealing the values of individual entries
Secure Multiparty Computation MPC

\[ f(x_1, x_2, x_3, x_4) = (y_1, y_2, y_3, y_4) = y \]

**Parties:**
- randomized distributed protocol
- synchronous rounds
- round complexity (maximal no of rounds before termination)

**Goals:**
- preserve the privacy of the player's inputs and guarantee the correctness of the computation.

**Diagram:**
- Particles labeled with \( x_1, x_2, x_3, x_4 \) and \( y \) for the computation.
The goal of secure multi-party computation is to achieve the same result without involving a trusted third party.

All of these tasks can be easily computed by a trusted third party.

The goal of secure multi-party computation is to achieve the same result without involving a trusted third party.
Secure Multiparty Computation MPC

A protocol is secure if it emulates an ideal setting where the parties hand their inputs to a “trusted party,” who locally computes the desired outputs and hands them back to the parties

\[ f(x_1, x_2, x_3, x_4) = (y_1, y_2, y_3, y_4) = y \]

Formally:

\[ f(x_1, x_2, x_3, x_4) = (y_1, y_2, y_3, y_4) = y \]

[Goldreich-Micali-Wigderson 87]
Secure Multiparty Computation MPC

Security:

- Provide an exact problem definition
  - Adversarial power
  - Network model
  - Meaning of security
- Prove that the protocol is secure
  - by reduction to an assumed hard problem (factoring large composites)

\[ f(x_1, x_2, x_3, x_4) = (y_1, y_2, y_3, y_4) = y \]
Defining security

Components of ANY security definition

- Adversarial power
- Network model
  - type of network
  - existence of trusted help
  - stand-alone versus composition
- Security guarantees
Components of ANY security definition
- Adversarial power
- Network model
  - type of network
  - existence of trusted help
  - stand-alone versus composition
- Security guarantees

Ex: Secure auction (with secret bids) requires:
  **Privacy:** the adversary may wish to learn the bids of all parties
  **Correctness:** the adversary may wish to win with a lower bid than the highest
  **Independence of inputs:** the adversary may also wish to ensure that it always gives the highest bid
Adversary: Models the coalitions of corrupted parties

**Passive / Active**
- **Passive:** *private computation*, Corrupted players follow the protocol but try to learn more.
- **Fail:** Corrupted player might stop sending messages at some point of the execution.
- **Active:** *secure computation*, Corrupted players can collaborate in any way and misbehave arbitrarily.

**Static / Adaptive**
- **Static:** set of corrupted parties fixed at onset
- **Adaptive:** can choose to corrupt parties at any time during computation.

**Unbounded / PPT**
- **Unbounded:** Unlimited power, information-theoretic model
- **PPT:** probabilistic polynomial-time with auxiliary input, non-uniform model of computation
Secure Multiparty Computation (MPC)

Network:

- **Synchronous**: communication proceeds in rounds, all messages received in same round.
- **Asynchronous**: messages can be delayed arbitrarily

- **Secure (authenticated) channels**: can be achieved using a public-key infrastructure of digital signatures
  - perfect information-theoretic security
  - unconditional security

- **Cryptographic model**: hardness assumption
Real vs. Ideal Paradigm

Real model:
- parties run a real protocol with no trusted help

Ideal model:
- parties send inputs to a trusted party $T$
  - $T$ computes the function and sends the outputs
Informally: a protocol is secure if any attack on a real protocol can be carried out (or simulated) in the ideal model.

Since essentially no attacks can be carried out in the ideal model, security is implied:

The adversary should not be able to do more damage in the **real model** than he is allowed in the **ideal model**
## Known results

Corrupts up to \( t \) out of \( n \) players

<table>
<thead>
<tr>
<th>Setting</th>
<th>Adversary</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>computational</td>
<td>passive</td>
<td>( t &lt; n )</td>
</tr>
<tr>
<td></td>
<td>active</td>
<td>( t &lt; n/2 )</td>
</tr>
<tr>
<td>information-theoretic</td>
<td>passive</td>
<td>( t &lt; n/2 )</td>
</tr>
<tr>
<td></td>
<td>active</td>
<td>( t &lt; n/3 )</td>
</tr>
<tr>
<td>i-t with broadcast</td>
<td>active</td>
<td>( t &lt; n/2 )</td>
</tr>
</tbody>
</table>

\( t < n/2 \) with “fail”
Ideal broadcast
Real broadcast (Active Adversary)
- **consistency**: All honest players have the same output (even from malicious senders).
- **validity**: If the sender is honest then all the honest players output $x_2$.
- **termination**: Every player ends with an output.
Consensus - Agreement achieved
Corrupts up to $t$ out of $n$ players

Active

<table>
<thead>
<tr>
<th>Setting</th>
<th>Type</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>computational</td>
<td>BC</td>
<td>$t &lt; n$</td>
</tr>
<tr>
<td></td>
<td>consens</td>
<td>$t &lt; n/2$</td>
</tr>
<tr>
<td>information-theoretic</td>
<td>BC</td>
<td>$t &lt; n$</td>
</tr>
<tr>
<td>PKI</td>
<td>consens</td>
<td>$t &lt; n/2$</td>
</tr>
<tr>
<td>information-theoretic</td>
<td>BC / consens</td>
<td>$t &lt; n/2$</td>
</tr>
</tbody>
</table>
## Methods of Secure MPC

<table>
<thead>
<tr>
<th>MPC Protocol</th>
<th>Setting</th>
<th>Adversary</th>
<th>Pre-processing</th>
<th>Rounds</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secret-Sharing based</td>
<td>unbounded (i-t)</td>
<td>passive</td>
<td>no</td>
<td>circuit depth</td>
<td>t &lt; n/2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>active</td>
<td>no</td>
<td>circuit depth</td>
<td>t &lt; n/3</td>
</tr>
<tr>
<td>BMR based (garbled circuits)</td>
<td>computational</td>
<td>passive</td>
<td>no</td>
<td>constant</td>
<td>t &lt; n</td>
</tr>
<tr>
<td>BMR cut-and-choose</td>
<td>computational</td>
<td>active</td>
<td>no</td>
<td>constant</td>
<td>t &lt; n</td>
</tr>
<tr>
<td>GMW Oblivious-Transfer based</td>
<td>computational</td>
<td>passive</td>
<td>yes</td>
<td>circuit depth</td>
<td>t &lt; n</td>
</tr>
<tr>
<td>SPDZ</td>
<td>computational</td>
<td>active</td>
<td>yes</td>
<td>circuit depth</td>
<td>t &lt; n</td>
</tr>
</tbody>
</table>
Outline

- OT
  - 2PC from OT

- Yao
  - 2PC from Yao Garbled Circuits

- Shamir
  - Secure Multiparty Computation from Secret Sharing

- Conclusion
  - Open Questions
Oblivious Transfer

2PC from OT

OT

2PC from Yao Garbled Circuits

Yao

Secure Multiparty Computation from Secret Sharing

Shamir

Open Questions

Conclusion
Overlook of MPC with preprocessing

Model of Computation:
• Represent the function as Arithmetic/Boolean circuit $C$
• Preprocessing phase
Overlook of MPC with preprocessing

Preprocessing:
• Independent of \( x, y \)
• Typically only depends on size of \( f \)
• Uses public key crypto technology (slower)

Online:
• Uses only information theoretic tools (order of magn. faster)
1-out-of-2 Oblivious Transfer (OT)

- A inputs two bits, B inputs the index of one of A’s bits
- B learns his chosen message (bit) \( m_i \)
- A learns nothing - A does not learn which bit B has chosen; B does not learn the value of the bit that he did not choose
- Generalizes to bitstrings, \( n \) instead of 2, etc.

[Rabin 1981]
Facts: Oblivious Transfer (OT)

- Introduced by Rabin [Rab81], Even, Goldreich and Lempel [EGL85]
- OT is equivalent to random OT [Cré88]
- OT is symmetric [WW06]
- OT is “complete” for secure multiparty computation [Kil88, IPS08]
- Black-box construction of OT from one-way permutations implies $P \neq NP$ [IR89]
- Perfect OT cannot be constructed using quantum mechanics [Lo97]
- OTs can be extended under computational assumptions [IKNP03]
- OTs cannot be extended using quantum mechanics [SSS09, WW10]
- OT implies PKE, but not vice-versa [GKM+00]

Constructions:
- PIR [CMO00]
- DDH [NP01]
- Projective hash proofs [Kal05, HK07]
- Blind signatures (requires RO) [CNS07]
- Bilinear assumptions [GH07]
- Dual-mode encryption [PVW08]
- Noisy Channels [IKO+11]
Oblivious Transfer (OT) from Encryption

Let \((G,E,D)\) be a public-key encryption scheme

Key-generation: \((pk,sk) \leftarrow G\)

Encryption: \(c = E_{pk}(m)\)

Decryption: \(m = D_{sk}(c)\)

[Reference: Rabin 1981]
Passive Protocol Oblivious Transfer (OT)

Sender sees two keys \( \text{pk}_0, \text{pk}_1 \) (no clue what \( i \) is)

Sends to receiver: \( c_0 = \text{E}_{\text{pk}}(m_0), c_1 = \text{E}_{\text{pk}_1}(m_1) \)

Receiver (with input \( i \))

- Chooses one key-pair \( (\text{pk}, \text{sk}) \) and one public-key \( \text{pk}' \) (no secret-key).
  - Receiver sets \( \text{pk}_i = \text{pk}, \text{pk}_{1-i} = \text{pk}' \)
  - Note: receiver can decrypt for \( \text{pk}_i \) but not for \( \text{pk}_{1-i} \)
  - Receiver sends \( \text{pk}_0, \text{pk}_1 \) to sender
Not secure against **active** Adversaries

Sender \((m_0, m_1)\)

Receiver (with input \(i\))

- Receiver does not follow the protocol:
  - Chooses one key-pair \((pk, sk)\) and key-pair \((pk', sk')\).
  - Receiver can decrypt for \(pk_i\) and for \(pk_{1-i}!!!\)
Can define 1-out-of-k oblivious transfer

Protocol remains the same:

- Choose \( k-1 \) public keys for which the secret key is unknown
- Choose 1 public-key and secret-key pair
MPC from OT

- **Stage 1**: each party randomly shares its input with the other party
● **Stage 1:** each party randomly shares its input with the other party

● **Stage 2:** compute gates of circuit as follows:

Given random shares to the input wires, compute random shares of the output wires
MPC from OT

- **Stage 1**: each party randomly shares its input with the other party

- **Stage 2**: compute gates of circuit as follows:
  
  Given random shares to the input wires, compute random shares of the output wires

- **Stage 3**: combine shares of the output wires in order to obtain actual output
• Represent \( f \) as an arithmetic circuit with addition and multiplication gates (over \( \text{GF}[2] \)).

• **Aim** – compute gate-by-gate, revealing only **random shares** each time:
  Let \( a \) be some value:
  - A holds a random value \( a_1 \)
  - B holds \( a_2 = a - a_1 \)
  \((a_1, a_2) = \text{random shares of } a\): without knowing \( a_1 \), \( a - a_1 \) is just a random value revealing nothing of \( a \)

*For simplicity – consider two-party case*
Addition gates: computed locally

\[ c_1 = a_1 + b_1 \]
\[ c_2 = a_2 + b_2 \]

Input wires to gate have values \( a \) and \( b \):
- A has shares \( a_1 \) and \( b_1 \)
- B has shares \( a_2 \) and \( b_2 \)
- Note: \( a_1 + a_2 = a \) and \( b_1 + b_2 = b \)

To compute random shares of output \( c = a+b \)
- A locally computes \( c_1 = a_1 + b_1 \)
- B locally computes \( c_2 = a_2 + b_2 \)
- Note: \( c_1 + c_2 = a_1 + a_2 + b_1 + b_2 = a+b=c \)
Input wires to gate have values $a$ and $b$:

- A has shares $a_1$ and $b_1$
- B has shares $a_2$ and $b_2$
- Wish to compute $c = ab = (a_1 + a_2)(b_1 + b_2)$

B’s values are unknown to A, but there are only 4 possibilities (depending on correspondence to 00, 01, 10, 11)
Multiplication gates?

How do we compute multiplication gates?

- Wish to compute $c = ab = (a_1 + a_2)(b_1 + b_2)$

- Method 1: Oblivious Transfer (OT) [GMW87]
- Method 2: Beaver triples [Beaver, CRYPTO91]
A prepares a table as follows:
A chooses a random bit \( r \)

- Row 1 contains the value \( a \cdot b + r \) when \( a_2 = 0, b_2 = 0 \) → B’s input 00
- Row 2 contains the value \( a \cdot b + r \) when \( a_2 = 0, b_2 = 1 \) → B’s input 01
- Row 3 contains the value \( a \cdot b + r \) when \( a_2 = 1, b_2 = 0 \) → B’s input 10
- Row 4 contains the value \( a \cdot b + r \) when \( a_2 = 1, b_2 = 1 \) → B’s input 11
<table>
<thead>
<tr>
<th>Row</th>
<th>B’s shares</th>
<th>Output value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a_2=0, b_2=0$</td>
<td>$(0+0)\cdot(1+0)+1=1$</td>
</tr>
<tr>
<td>2</td>
<td>$a_2=0, b_2=1$</td>
<td>$(0+0)\cdot(1+1)+1=1$</td>
</tr>
<tr>
<td>3</td>
<td>$a_2=1, b_2=0$</td>
<td>$(0+1)\cdot(1+0)+1=0$</td>
</tr>
<tr>
<td>4</td>
<td>$a_2=1, b_2=1$</td>
<td>$(0+1)\cdot(1+1)+1=1$</td>
</tr>
</tbody>
</table>

$$a_1=0, \ b_1=1, \ r=1$$
The parties run a 1-out-of-4 oblivious transfer protocol

A plays the sender: message $i$ is row $i$ of the table.

B plays the receiver: it inputs $1$ if $a_2=0$ and $b_2=0$, $2$ if $a_2=0$ and $b_2=1$, …

- B receives $c_2=c+r$ – this is its output
- A outputs $c_1=r$

Note: $c_1$ and $c_2$ are random shares of $c$, as required
### Multiplication gates with OT

<table>
<thead>
<tr>
<th>Row</th>
<th>B’s shares</th>
<th>Output value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a_2=0, b_2=0$</td>
<td>$a \cdot b + r = 1$</td>
</tr>
<tr>
<td>2</td>
<td>$a_2=0, b_2=1$</td>
<td>$a \cdot b + r = 1$</td>
</tr>
<tr>
<td>3</td>
<td>$a_2=1, b_2=0$</td>
<td>$a \cdot b + r = 0$</td>
</tr>
<tr>
<td>4</td>
<td>$a_2=1, b_2=1$</td>
<td>$a \cdot b + r = 1$</td>
</tr>
</tbody>
</table>

**Stage 3:**
- By computing each gate these way, at the end the parties hold shares of the output wires.
- Function output generated by simply sending shares to each other.
The above protocol is **not** secure against **active** adversaries:

- An active adversary may **learn more** than it should.
- An active adversary can cause the honest party to receive **incorrect output**.

### Table of Shares and Output Values

<table>
<thead>
<tr>
<th>Row</th>
<th>B’s shares</th>
<th>Output value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a_2 = 0, b_2 = 0$</td>
<td>$a \cdot b + r = 1$</td>
</tr>
<tr>
<td>2</td>
<td>$a_2 = 0, b_2 = 1$</td>
<td>$a \cdot b + r = 1$</td>
</tr>
<tr>
<td>3</td>
<td>$a_2 = 1, b_2 = 0$</td>
<td>$a \cdot b + r = 0$</td>
</tr>
<tr>
<td>4</td>
<td>$a_2 = 1, b_2 = 1$</td>
<td>$a \cdot b + r = 1$</td>
</tr>
</tbody>
</table>
Security of 2PC from OT

- Reduction to the oblivious transfer protocol
- Assuming security of the OT protocol, parties only see random values until the end. Therefore, simulation is straightforward.
- Note: correctness relies heavily on semi-honest (passive) behavior (otherwise A and B can modify shares).
From Passive to Active Adversary

- The semi-honest model is often used as a tool for obtaining security against malicious parties.
- In many settings, security against semi-honest adversaries does not suffice.
  - In some settings, it may suffice.
  - One example: hospitals that wish to share data.
Three goals:

- Force the adversary to use a **fixed input**
  
  Furthermore, make it possible for the ideal-model simulator/adversary to **extract** this input.

- Force the adversary to use a **uniform random tape**

- Force the adversary to follow the protocol exactly (consistently with their fixed input and random tape)

Techniques:

- Commitments (hiding and binding)
- Coin Tossing protocol [Bloom]
- Zero-Knowledge proofs
GMW paradigm:

- First, construct a protocol for semi-honest adv.
- Then, compile it so that it is secure also against malicious adversaries

There are many other ways to construct secure protocols – some of them significantly more efficient.

Efficient protocols against semi-honest adversaries are far easier to obtain than for malicious adversaries.
Review:

- Inputs are secret-shared between participants.
- The circuit is evaluated gate-by-gate.
- Each gate acts only on shares, so no information is revealed.
- Addition gates can be computed locally (no communication between participants).
- Multiplication requires a 1-out-of-4 OT.
- OT requires public-key operations.
- Public-key operations are computationally expensive.
- Computation requires a number of rounds equal to the multiplicative depth of the circuit.
Multiplication gates with Beaver triples

How do we compute multiplication gates?

- Wish to compute $c = ab = (a_1+a_2)(b_1+b_2)$

$$c = (a_1 + a_2)(b_1 + b_2)$$
How do we compute multiplication gates?

- Wish to compute $c = ab = (a_1+a_2)(b_1+b_2)$

$$c = (a_1+a_2)(b_1+b_2)$$

$$= a_1 b_1 + a_1 b_2 + a_2 b_1 + a_2 b_2$$
Multiplication gates with Beaver triples

How do we compute multiplication gates?

- Wish to compute $c = ab = (a_1 + a_2)(b_1 + b_2)$

$$c = (a_1 + a_2)(b_1 + b_2)$$

$$= a_1 b_1 + a_1 b_2 + a_2 b_1 + a_2 b_2$$

Alice can compute $\underline{a_1 b_1}$, $\underline{a_1 b_2}$, $\underline{a_2 b_1}$, Bob can compute $\underline{a_2 b_2}$. 
Multiplication gates with Beaver triples

How do we compute multiplication gates?

- Wish to compute \( c = ab = (a_1 + a_2)(b_1 + b_2) \)

\[
c = (a_1 + a_2)(b_1 + b_2) = a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2
\]

- Method 2: Beaver triples [Beaver, CRYPTO91]
Get Beavers shares \([x],[y],[z]\) with \(z=xy\) from trusted dealer

\[
e = \text{Open}([a]+[x])
\]
\[
d = \text{Open}([b]+[y])
\]

Compute \([c] = [z] + e[b] + d[a] - ed\)

\[
xy + (xb+ab) + (ya+ab) -(xy+xb+ya+ab)=ab
\]
Trusted dealer: **Active Security?**

**Privacy?**
– even a malicious Bob does not learn anything

**Correctness?**
– a corrupted Bob can change his share during any “Open” (both final result or during multiplication) leading the final output to be incorrect
MAC-then-Compute for Active Security

**Solution:** Enhance representation of \([a], [b]\)

\[
[a] = (a_1, k_{1a}, m_{1a}), (a_2, k_{2a}, m_{2a})
\]

\[
m_{1a} = a_1 \Delta_2 + k_{2a} \text{ (symmetric for } m_2)\]

\(\Delta_1, \Delta_2\) are the same for all wires
MAC-then-Compute for Active Security

Given

\[ [a] = (a_1, k_{1a}, m_{1a}), (a_2, k_{2a}, m_{2a}) \]
\[ [b] = (b_1, k_{1b}, m_{1b}), (b_2, k_{2b}, m_{2b}) \]

Compute

\[ [c] = (c_1 = a_1 + b_1, k_{1c} = k_{1a} + k_{1b}, m_{1c} = m_{1a} + m_{1b}), \]
\[ (c_2 = a_2 + b_2, k_{2c} = k_{2a} + k_{2b}, m_{2c} = m_{2a} + m_{2b}) \]

and \[ m_{1c} = a_1 \Delta_2 + k_{2a} + b_1 \Delta_2 + k_{2b} = c_1 \Delta_2 + k_{2c} \]
Output Gates:
- Exchange shares and MACs
- Abort if MAC does not verify

Input Gates:
- Get a random \([r]\) from trusted dealer
- Alice open \(r\)
- Alice sends Bob \(d = x - r\),
- Compute \([x] = [r] + d\)
Output Gates:

- Each party must store a mac/key pair for each other party
  - quadratic complexity!
  - SPDZ for linear complexity.

- MAC is only as hard as guessing key!
  - kMACs in parallel give security $1/|F|^k$
  - TinyOT $F=FG[2]$, then MACs/Keys are k-bit strings
  - MiniMACs for constant overhead
(SPDZ)/BeDOZa/TinyOT online phase

- Storage: linear $\#$number of AND gates
- Communication: linear $\#$number of AND gates
- $\#$rounds = depth of the circuit

...and add enough MACs to get active security
One-Time Truth Table:

Preprocessing:
1. Write the truth table $T$ of the function $f$ you want to compute
2. Pick random $(r,s)$ and rotate rows and columns
3. Random secret share the Truth table $T_1, T_2 = T - T_1$

Online:

$$f(x,y) = T_1[u,v] + T_2[u,v]$$

- Storage: $\exp(\text{input size})$
- Communication: $O(\text{input size})$
- 1 round -- Optimal communication complexity
Garbled Circuits

OT

2PC from OT

2PC from Yao Garbled Circuits

Yao

Secure Multiparty Computation from Secret Sharing

Shamir

Open Questions

Conclusion
Alice garbels each gate:

- Alice will generate key pairs for every wire
Alice garbels each gate:

- Alice will generate key pairs for every wire
- Alice generates garbled truth tables for each gate using wire keys: pair of randomly colored keys for each wire, one for 0, one for 1
Garbled gates

- A gives B the garbled gates (but not the wire keys)
A gives B the key corresponding to her input $G=1$
Garbled Circuits

\[ G = 1 \]

A’s labels
- \( G_0 \)
- \( E_1 \)

B’s labels
- \( E_0 \)
- \( G_1 \)

garbler

evaluator

• A gives B the key corresponding to her input \( G=1 \)
• A gives B the key corresponding to her input $G=1$
• A and B use OT to get B the keys he needs for his input $E=1$
A gives B the key corresponding to her input $G=1$
A and B use OT to get B the keys he needs for his input $E=1$
Now B has one wire key for each input wire
A gives B the key corresponding to her input $G=1$
A and B use OT to get B the keys he needs for his input $E=1$
Now B has one wire key for each input wire
Garbled Circuits

To compute more complex functions, express them as circuits made up of individual gates.
To compute more complex functions, express them as circuits made up of individual gates. Then, encrypt labels for the next gates.
Implementing locks
- (one-time) symmetric encryption
  ➔ computational privacy, works for any circuit
- one-time pads
  ➔ information-theoretic privacy, efficient only for log-depth circuits
Example: Garbled Circuits

- Each wire carries a string
- Strings on input wires allow decryption of string on output wire
High-Level Approach

- Public function is written as a circuit
- Alice will “garble” the circuit
- Alice will send the “garbled” circuit to Bob
- Alice and Bob will engage in OT to get Bob his “garbled” inputs
- Bob will use the garbled inputs to compute circuit gate-by-gate
Garbled Circuit Review

- Alice and Bob want to compute a public function
  a circuit of size $|C|$ with $n$ inputs
- Alice will generate key pairs for every wire
- Alice will generate garbled truth tables for every gate
  Calculate: 8 symmetric encryptions per gate
- Alice will give Bob the entire garbled circuit
  Communicate: 4 symmetric encryptions per gate
- Alice will give secrets corresponding to her inputs
- They will use OT to get Bob secrets corresponding to his inputs
  One OT per input
- Bob will evaluate the entire garbled circuit gate by gate
  Calculate: 2 symmetric decryptions per gate

**Overall:**
- Calculation: $O(|C|)$ symmetric encryptions for each party
- Communication: $O(|C|)$ symmetric ciphertexts
- OT: $n$ parallel oblivious transfers
- Entire protocol is only two rounds
Garbled Circuit Implementation

- **Fairplay:** a garbled circuit compiler [MNPS04]
- **Garbled circuits for malicious adversaries** [LPS08]
  Computation of $>$ for 16-bit integers takes 135-360 seconds on Intel Core 2 2.13Ghz where running time increases as security parameter increases
- **TASTY:** a compiler which compiles into a mix of garbled circuits and homomorphic encryption [HOS+10]
- **900-bit hamming distance calculation** in .051s on Inter Core Duo 3Ghz [HEKM11, HSE+11]
  (.019s online time, approximately 10μs per gate)
- **Billion-gate circuits against malicious adversaries** [KSS12]
  - Garble time: 64,000 - 84,000 gates / sec
  - Secure evaluation of AES (50K gates): 29.4s on 4 cores, 1.3s on 256 cores
  - 4095-bit edit distance (5.9B gates) in 8.2 hours (82K gates/sec)
MPC with Shamir Secret Sharing

OT

2PC from OT

Yao

2PC from Yao Garbled Circuits

Shamir

Secure Multiparty Computation from Secret Sharing

Conclusion

Open Questions
The GMW/BGW Approach:

- The (public) function being computed is written as a circuit
- Each participant secret-shares their private input
- The circuit is evaluated gate-by-gate on the shares (this requires communication between participants)
- Answer is reconstructed from final shares
Random Secret Sharing for bits

The GMW/BGW Approach:
- The (public) function being computed is written as a circuit
- Each participant secret-shares their private input
- The circuit is evaluated gate-by-gate on the shares (this requires communication between participants)
- Answer is reconstructed from final shares

Already seen - Random Sharing:
- Simple linear sharing scheme
- Share \( s \) in GF[2], among two players
  - Pick a random bit \( r \)
  - Create the shares \( s+r \) and \( r \)
  - Keep \( s+r \)
  - Send \( r \) to other player
- \( s_1 = s+r \)
- \( s_2 = r \)
- \( s = s_1 + s_2 \)
Shamir Secret Sharing

[Sha79]

How it works:

- A Dealer holds a secret value $s$ in $\mathbb{Z}_p^*$, where $p > n$ is a prime.
- Dealer chooses a random polynomial $f()$ over $\mathbb{Z}_p^*$ of degree at most $t$, such that $f(0)=s$:
  
  $$f(x) = s + a_1 x + a_2 x^2 + \ldots + a_t x^t$$

- Dealer sends $s_i = f(i)$ privately to $P_i$.

$s_1 = f(1)$

$s_2 = f(2)$

$s_3 = f(3)$
Shamir Secret Sharing

[Sha79]

Properties: \( \text{deg}(f)=t \)

- Any subset of at most \( t \) players has no information on \( s \)
- Any subset of at least \( t+1 \) players can easily compute \( s \) – can be done by taking a linear combination of the shares they know.

Lagrange interpolation: recover \( f \) from \( t+1 \) shares

A consequence – the reconstruction vector:

There exists a reconstruction vector \( (r_1, \ldots, r_n) \) such that for any polynomial \( h() \) of degree less than \( n \):

\[
h(0) = r_1 h(1) + \ldots + r_n h(n)
\]

\( s=f(0) \)
Notation: $a \xrightarrow{f()} a_1, a_2, \ldots, a_n$

means: value $a$ has been shared using polynomial $f()$, resulting in shares $a_1, \ldots, a_n$, where player $P_i$ knows $a_i$. 
Properties: Linearly Homomorphic

Inputs: \( s \xrightarrow{f()} s_1, \ldots, s_n \) \quad \text{and} \quad \( t \xrightarrow{g()} t_1, \ldots, t_n \)

Addition: \( s + t \xrightarrow{(f + g)()} s_1 + t_1, \ldots, s_n + t_n \)

Adding two random polynomials \( f() \) and \( g() \) of degree \( \leq t \) produces random polynomial \( (f + g)() \) of degree \( \leq t \)
MPC from Shamir Secret Sharing

Sharing:

Each $P_i$ shares each of his input value using a random polynomial of degree at most $t$, sends a share of each input to each player.

$s_1 = f(1)$

$s_2 = f(2)$

$s_3 = f(3)$
MPC from Shamir Secret Sharing

Sharing:
Each $P_i$ shares each of his input value using a random polynomial of degree at most $t$, sends a share of each input to each player.

Computation:

Addition Gates

Input:
- $a \xrightarrow{f_a()} a_1, \ldots, a_n$
- $b \xrightarrow{f_b()} b_1, \ldots, b_n$

Desired Output:
- $c = a + b \xrightarrow{f_c()} c_1, \ldots, c_n$

Each player sets $c_i := a_i + b_i$.

This is a share of $a + b \xrightarrow{f_c()} c_1, \ldots, c_n$, with $f_c() = f_a() + f_b()$. 
Multiplication Gates

Input: \[ a \xrightarrow{f_a()} a_1, \ldots, a_n \]
\[ b \xrightarrow{f_b()} b_1, \ldots, b_n \]

Desired Output:
\[ c = ab \xrightarrow{f_c()} c_1, \ldots, c_n \]

Each player sets \( d_i := a_i b_i \).

If we set \( h() = f_a() f_b() \), then
\[ d_i = f_a(i) f_b(i) = h(i). \text{ Also } h(0) = ab = c \]

Unfortunately, \( h() \) may have degree up to \( 2t \), and is not even a random polynomial of degree at most \( 2t \). :(
**Multiplication Gates**

*Input:* \( a \xrightarrow{f_a} a_1, \ldots, a_n \) \( b \xrightarrow{f_b} b_1, \ldots, b_n \)

*Desired Output:*

\( c = ab \xrightarrow{f_c} c_1, \ldots, c_n \)

Each player sets \( d_i : = a_i b_i \).

**Reshare \( c = h(0) \):**

\[ d_i = f_a(i) f_b(i) = h(i) \]

We have public reconstruction vector \((r_1, \ldots, r_n)\) – know that

\[ c = h(0) = r_1 h(1) + \ldots + r_n h(n) \]

\[ = r_1 d_1 + \ldots + r_n d_n \text{ - since } \deg(h) \leq 2t < n \]

Each player \( P_i \) creates \( d_i \xrightarrow{h_i} c_{i1}, c_{i2}, \ldots, c_{in} \).
Multiplication Gates

Each player $P_i$ creates

$$d_i \rightarrow h_i() \rightarrow c_{i1}, c_{i2}, \ldots, c_{in}.$$  

So we have:

$$d_1 \rightarrow h_1() \rightarrow c_{11}, c_{12}, \ldots, c_{1n}.$$  

$$d_2 \rightarrow h_2() \rightarrow c_{21}, c_{22}, \ldots, c_{2n}.$$  

$$\ldots$$  

$$d_n \rightarrow h_n() \rightarrow c_{n1}, c_{n2}, \ldots, c_{nn}.$$
Multiplication Gates

Each player $P_i$: $d_i \rightarrow h_i() \rightarrow c_{i1}, c_{i2}, \ldots, c_{in}$.

$P_1$  $P_2$  $\ldots$  $P_n$

$r_1c_{11}$  $+ \quad r_1c_{12}$  $+ \quad r_1c_{1n}$

$r_2c_{21}$  $+ \quad r_2c_{22}$  $+ \quad r_2c_{2n}$

$\ldots$  $\quad \ldots$  $\quad \ldots$

$r_nc_{n1}$  $= \quad r_nc_{n2}$  $= \quad r_nc_{nn}$

$c \rightarrow f_c() \rightarrow c_1 \quad c_2 \quad \ldots \quad c_n$
MPC from Shamir Secret Sharing

Multiplication Gates

Each player $P_i : d_i \rightarrow h_i() \rightarrow c_{i1}, c_{i2}, \ldots, c_{in}$.

$c$ is now shared using polynomial $f_c()$, where

$$f_c() = \sum r_i h_i()$$
Output opening:

Having made our way through the circuit, we have for each output value $y$:

$$ y \xrightarrow{f_y()} y_1, \ldots, y_n $$

If $y$ is to be received by player $P_i$, each $P_j$ sends $y_j$ to $P_i$.

$P_i$ reconstructs in the normal way.
Security against Passive Adversary

Review of GMW/BGW Approach:

- The (public) function being computed is written as a circuit
- Each participant secret-shares their private input
- The circuit is evaluated gate-by-gate on the shares (this requires communication between participants)
- Answer is reconstructed from final shares

Security Intuition:

Outputs trivially correct, since all players follow the protocol

For every input from an honest player, intermediate result and outputs of honest players, Adv sees at most t shares. These are always t random field elements, so reveal no information.
General idea: use protocol for passive case, but make players prove that they send correct information.

Main tool for this: commitment scheme

Intuition: Committer $P_i$ puts secret value $s$ “in a locked box” and puts it on the table. Later, $P_i$ can choose to open the box, by releasing the key.

**Hiding** – no one else can learn $s$ from the commitment

**Binding** – having given away the box, $P_i$ cannot change what is inside.
More Approaches

- Shamir
- Yao
- 2PC from Yao Garbled Circuits
- Secure Multiparty Computation from Secret Sharing
- Other Techniques
- Open Questions
MPC using FHE

Intuition: only FHE

- Low round complexity
- Low communication complexity
  - Independent of the function $f$
  - Independent of Sally’s input $b$
- Low computation
  - Charlie’s work is independent of $f$
- A simple template
Results - MPC with TFHE

1. [Cramer Damgard Nielsen 01] – MPC using threshold HE
2. [Gentry 09] – MPC using threshold FHE
3. [Bendlin Damgard 10] – threshold version for LWE
4. [Katz Ostrovsky 04] – lower bound of 5 rounds for MPC in the plain model
**MPC using FHE**

**Idea:** Share the secret key (decryption)

- **Threshold Key Gen**
- **Encrypt and Evaluate**
- **Threshold Decryption**
MPC with TFHE

• **Threshold KeyGen** and **Threshold Dec** can be implemented using **generic MPC**

• Advantages:
  - Low communication complexity (even in malicious)
  - The homomorphic evaluation can be delegated / only one party

• Disadvantages:
  - Needs generic MPC techniques
  - Round complexity can be high
MPC with TFHE for Passive

• **Threshold KeyGen** and **Threshold Dec** algebraically [BV11b, BGV12] (based on LWE)

• Advantages:
  - Low communication complexity (even in malicious)
  - The homomorphic evaluation can be delegated / only one party (passive)
  - Simple: there is no need for generic MPC protocol
  - **Extremely low round complexity**
    - Only 3 broadcast rounds (CRS model)
    - 2 rounds reusable PKI – optimal(!)
Conclusions
Thank You