Methods of Secure Computation

Pille Pullonen pille.pullonen@cyber.ee

April 8, 2020
Topics

Generic Secure Computation
- SPDZ
- (F)HE
- GC
- TEE (e.g. SGX)

Special Purpose Protocols
- PIR
- OT
- PSI
- ORAM
Part I: Generic MPC Protocols
Security of MPC protocols

Discussion: What does it mean for an MPC protocol to be secure?
Security of MPC protocols

- Discussion: What does it mean for an MPC protocol to be secure?
- General idea: A protocol where the parties only learn the output and nothing else
- Discussion: How might we prove this?

We assume there exists a black box that takes the input and gives the output (a.k.a. the ideal functionality or trusted third party). Then we show that our protocol is indistinguishable from the black box. Actually showing this is very technical. In this lecture we only discuss the intuition of why the protocols seem to be secure. See [Lin16] for more details.
Security of MPC protocols

- **Discussion**: What does it mean for an MPC protocol to be secure?
- **General idea**: A protocol where the parties only learn the output and nothing else
- **Discussion**: How might we prove this?
- We assume there exists a black box that takes the input and gives the output (a.k.a. the ideal functionality or trusted third party)
- Then we show that our protocol is indistinguishable from the black box
Security of MPC protocols

- **Discussion:** What does it mean for an MPC protocol to be secure?
- **General idea:** A protocol where the parties only learn the output and nothing else

- **Discussion:** How might we prove this?

- We assume there exists a black box that takes the input and gives the output (a.k.a. the ideal functionality or trusted third party)

- Then we show that our protocol is indistinguishable from the black box

- Actually showing this is very technical. In this lecture we only discuss the intuition of why the protocols seem to be secure

- Take Cryptographic Protocols or Cryptography II courses or see [Lin16]
Active vs Passive Security

- Passive adversary (a.k.a Honest but Curious)
  - Follows the protocol
  - Thinks about everything it sees
  - Can derive new knowledge from its view

- Active adversary (a.k.a Malicious)
  - Can do whatever they like
  - May not follow the protocol
  - May modify their behaviour to learn the private values of other parties or simply break the execution
Active vs Passive Security

- Passive adversary (a.k.a Honest but Curious)
  - Follows the protocol
  - Thinks about everything it sees
  - Can derive new knowledge from its view

- Active adversary (a.k.a Malicious)
  - Can do whatever they like
  - May not follow the protocol
  - May modify their behaviour to learn the private values of other parties or simply break the execution
Additive Secret Sharing

Discussion Idea of secret sharing?

Idea of secret sharing?

\[ x = (x_1, \ldots, x_n) \]

For SPDZ protocol we have

\[ x \in \mathbb{F} \]

commonly

\[ \mathbb{F}_p \]

for some prime

\[ p \]

\[ x = \sum x_i \mod p \]

Field recap

Set of elements and two operations

\[ + \] addition and

\[ \cdot \] multiplication

Unit element (e.g. 0 and 1) for both operations

Set of elements is a commutative group with respect to both operations

For multiplicative group we exclude 0

Finite field means that the set of elements is finite
Additive Secret Sharing

Discussion Idea of secret sharing?

\[ [x] = (x_1, \ldots, x_n) \text{ where } x = \sum x_i \]

For SPDZ protocol we have \( x \in \mathbb{F} \)

- commonly \( \mathbb{F}_p \) for some prime \( p \)
- \( x = \sum x_i \mod p \)

Field recap

Set of elements and two operations \(+\) addition and \(\cdot\) multiplication

Unit element (e.g. 0 and 1) for both operations

Set of elements is a commutative group with respect to both operations

For multiplicative group we exclude 0

Finite field means that the set of elements is finite
Additive Secret Sharing

- **Discussion** Idea of secret sharing?
- \([x] = (x_1, \ldots, x_n)\) where \(x = \sum x_i\)
- For SPDZ protocol we have \(x \in \mathbb{F}\)
  - commonly \(\mathbb{F}_p\) for some prime \(p\)
  - \(x = \sum x_i \mod p\)

Field recap
- Set of elements and two operations + addition and \(\cdot\) multiplication
- Unit element (e.g. 0 and 1) for both operations
- Set of elements is a commutative group with respect to both operations
  - For multiplicative group we exclude 0
- Finite field means that the set of elements is finite
MPC with Preprocessing

- **Online phase**
  - Uses the private inputs
  - Computes the necessary functionality
  - Must be efficient

- **Offline phase**
  - Independent of the inputs
  - Independent of the desired computations
  - Can be used to prepare correlated randomness for the online phase

**Random values**

Beaver triples - 

\[
\begin{bmatrix}
a \\
b \\
c
\end{bmatrix}
\]

where \(a\) and \(b\) are random and \(c = a \cdot b\)

Efficiency is less important
MPC with Preprocessing

- **Online phase**
  - Uses the private inputs
  - Computes the necessary functionality
  - Must be efficient

- **Offline phase**
  - Independent of the inputs
  - Independent of the desired computations
  - Can be used to prepare correlated randomness for the online phase
    - Random values
    - Beaver triples - \([a], [b]\) and \([c]\) where \(a\) and \(b\) are random and \(c = a \cdot b\)
  - Efficiency is less important
Online Phase with Additive Secret Sharing

- Addition \([x + y] = [x] + [y] = (x_1 + y_1, \ldots, x_n + y_n)\)
Online Phase with Additive Secret Sharing

- Addition $[x + y] = [x] + [y] = (x_1 + y_1, \ldots, x_n + y_n)$
- Adding a public value $[x + c] = [x] + c = (x_1 + c, \ldots, x_n)$
Online Phase with Additive Secret Sharing

- Addition $[x + y] = [x] + [y] = (x_1 + y_1, \ldots, x_n + y_n)$
- Adding a public value $[x + c] = [x] + c = (x_1 + c, \ldots, x_n)$
- Multiplication by a public constant $[c \cdot x] = c \cdot [x] = (c \cdot x_1, \ldots, c \cdot x_n)$
Online Phase with Additive Secret Sharing

- Addition \([x + y] = [x] + [y] = (x_1 + y_1, \ldots, x_n + y_n)\)
- Adding a public value \([x + c] = [x] + c = (x_1 + c, \ldots, x_n)\)
- Multiplication by a public constant \([c \cdot x] = c \cdot [x] = (c \cdot x_1, \ldots, c \cdot x_n)\)
- Sharing a value - Party chooses random \(x_1, \ldots, x_{n-1}\) and computes
  \[x_n = x - \sum_{i=1}^{n-1} x_i,\] sends \(x_i\) to party \(i\)
Online Phase with Additive Secret Sharing

- Addition $[x + y] = [x] + [y] = (x_1 + y_1, \ldots, x_n + y_n)$
- Adding a public value $[x + c] = [x] + c = (x_1 + c, \ldots, x_n)$
- Multiplication by a public constant $[c \cdot x] = c \cdot [x] = (c \cdot x_1, \ldots, c \cdot x_n)$
- Sharing a value - Party chooses random $x_1, \ldots, x_{n-1}$ and computes $x_n = x - \sum_{i=1}^{n-1} x_i$, sends $x_i$ to party $i$
- Publishing a value - each party sends their share $x_i$, parties compute $x = \sum x_i$
Online Phase with Additive Secret Sharing

- Addition $[[x + y]] = [[x]] + [[y]] = (x_1 + y_1, \ldots, x_n + y_n)$
- Adding a public value $[[x + c]] = [[x]] + c = (x_1 + c, \ldots, x_n)$
- Multiplication by a public constant $[[c \cdot x]] = c \cdot [[x]] = (c \cdot x_1, \ldots, c \cdot x_n)$
- Sharing a value - Party chooses random $x_1, \ldots, x_{n-1}$ and computes $x_n = x - \sum_{i=1}^{n-1} x_i$, sends $x_i$ to party $i$
- Publishing a value - each party sends their share $x_i$, parties compute $x = \sum x_i$
- Multiplication $[[w]] = [[x \cdot y]] = [[x]] \cdot [[y]]$ with Beaver triple $[[a]], [[b]], [[c]]$, where $c = ab$
Online Phase with Additive Secret Sharing

- Addition $[x + y] = [x] + [y] = (x_1 + y_1, \ldots, x_n + y_n)$
- Adding a public value $[x + c] = [x] + c = (x_1 + c, \ldots, x_n)$
- Multiplication by a public constant $[c \cdot x] = c \cdot [x] = (c \cdot x_1, \ldots, c \cdot x_n)$
- Sharing a value - Party chooses random $x_1, \ldots, x_{n-1}$ and computes $x_n = x - \sum_{i=1}^{n-1} x_i$, sends $x_i$ to party $i$
- Publishing a value - each party sends their share $x_i$, parties compute $x = \sum x_i$
- Multiplication $[w] = [x \cdot y] = [x] \cdot [y]$ with Beaver triple $[a], [b], [c]$, where $c = ab$
  - Compute $[e] = [x] - [a]$ and $[d] = [y] - [b]$
  - Publish $e = \sum e_i$ and $d = \sum d_i$
  - Compute $[w] = [c] + e \cdot [b] + d \cdot [a] + ed$

April 8, 2020
Online Phase with Additive Secret Sharing

- Addition $[x + y] = [x] + [y] = (x_1 + y_1, \ldots, x_n + y_n)$
- Adding a public value $[x + c] = [x] + c = (x_1 + c, \ldots, x_n)$
- Multiplication by a public constant $[c \cdot x] = c \cdot [x] = (c \cdot x_1, \ldots, c \cdot x_n)$
- Sharing a value - Party chooses random $x_1, \ldots, x_{n-1}$ and computes $x_n = x - \sum_{i=1}^{n-1} x_i$, sends $x_i$ to party $i$
- Publishing a value - each party sends their share $x_i$, parties compute $x = \sum x_i$
- Multiplication $[w] = [x \cdot y] = [x] \cdot [y]$ with Beaver triple $[a], [b], [c]$, where $c = ab$
  - Compute $[e] = [x] - [a]$ and $[d] = [y] - [b]$
  - Publish $e = \sum e_i$ and $d = \sum d_i$
  - Compute $[w] = [c] + e \cdot [b] + d \cdot [a] + ed$
  - Correctness:
    $$ab + (x - a) \cdot b + (y - b) \cdot a + (x - a)(y - b) = ab + xb - ba + ya - ba + yx - ay - xb + ab = xy$$
But Active Security?

Discussion: What can go wrong in the previous protocol?

- Need some way to authenticate or verify the sharing or actions of the parties.
- We will use message authentication codes (MAC).
- Can be used to authenticate either the share or the secret value.
- Example: Each party has a share $x_i$ and some tag $t_i$ for a MAC that other party can check.
- Does not scale - need a tag for each party.
- Only works if the MAC is homomorphic.
- Because we need MACs on the computation results, can only get them from MACs on the inputs.
But Active Security?

- *Discussion*: What can go wrong in the previous protocol?
- Need some way to authenticate or verify the sharing or actions of the parties.
But Active Security?

- *Discussion*: What can go wrong in the previous protocol?
- Need some way to authenticate or verify the sharing or actions of the parties
- We will use message authentication codes (MAC)
- Can be used to authenticate either the share or the secret value
But Active Security?

- *Discussion*: What can go wrong in the previous protocol?
- Need some way to authenticate or verify the sharing or actions of the parties
- We will use message authentication codes (MAC)
- Can be used to authenticate either the share or the secret value
- Example:
  - Each party has a share $x_i$ and some tag $t_i$ for a MAC that other party can check
  - Does not scale - need a tag for each party
But Active Security?

- **Discussion:** What can go wrong in the previous protocol?
- Need some way to authenticate or verify the sharing or actions of the parties
- We will use message authentication codes (MAC)
- Can be used to authenticate either the share or the secret value
- Example:
  - Each party has a share $x_i$ and some tag $t_i$ for a MAC that other party can check
  - Does not scale - need a tag for each party
- Only works if the MAC is homomorphic
  - Because we need MACs on the computation results
  - Can only get them from MACs on the inputs
SPDZ style MAC

- Authenticate secret value not the separate shares
- \( \text{MAC}(x) = \alpha \cdot x \mod p \)
- Security - given \( x \) can we modify the message and create a valid MAC?
SPDZ style MAC

- Authenticate secret value not the separate shares
- $\text{MAC}(x) = \alpha \cdot x \mod p$
- Security - given $x$ can we modify the message and create a valid MAC?
  - Can not see $\alpha$ nor $\text{MAC}(x)$, find $e \neq 0$ and $e'$
  - $\text{MAC}(x + e) = \text{MAC}(x) + e'$
  - Need to find $e, e'$, we have $\alpha e - e' = 0$ for a successful attack
  - Probability $\frac{1}{|\mathbb{F}|}$ - equivalent to guessing the key
  - Need a big field
SPDZ style MAC

- Authenticate secret value not the separate shares
- \( \text{MAC}(x) = \alpha \cdot x \mod p \)
- Security - given \( x \) can we modify the message and create a valid MAC?
  - Can not see \( \alpha \) nor MAC(\( x \)), find \( e \neq 0 \) and \( e' \)
  - MAC(\( x + e \)) = MAC(\( x \)) + e'
  - Need to find \( e, e' \), we have \( \alpha e - e' = 0 \) for a successful attack
  - Probability \( \frac{1}{|F|} \) - equivalent to guessing the key
  - Need a big field
- Homomophic given MAC(\( x \)) and MAC(\( y \))
  - MAC(\( x + y \)) = MAC(\( x \)) + MAC(\( y \)) = \alpha(\( x + y \))
Share Representation with MAC

- Common key $\alpha$
- Additive shares of the secret and the MAC
  - $\lbrack x \rbrack = \langle (x_1, \text{MAC}(x)_1), \ldots, (x_n, \text{MAC}(x)_n) \rangle$
  - $x = \sum x_i$
  - $\text{MAC}(x) = \sum \text{MAC}(x)_i$
Share Representation with MAC

- Common key $\alpha$
- Additive shares of the secret and the MAC
  
  \[ [x] = (x_1, MAC(x)_1), \ldots, (x_n, MAC(x)_n) \]

\[ x = \sum x_i \]

\[ MAC(x) = \sum MAC(x)_i \]

- Key $\alpha$ is additively shared $\langle \alpha \rangle = (\alpha_1, \ldots, \alpha_n)$ where $\alpha = \sum \alpha_i$
Share Representation with MAC

- Common key $\alpha$
- Additive shares of the secret and the MAC
  - $[x] = ((x_1, \text{MAC}(x)_1), \ldots, (x_n, \text{MAC}(x)_n))$
  - $x = \sum x_i$
  - $\text{MAC}(x) = \sum \text{MAC}(x)_i$
- Key $\alpha$ is additively shared $\langle \alpha \rangle = (\alpha_1, \ldots, \alpha_n)$ where $\alpha = \sum \alpha_i$
- How do we compute?
Share Representation with MAC

- Common key $\alpha$
- Additive shares of the secret and the MAC
  - $[x] = ((x_1, \text{MAC}(x)_1), \ldots, (x_n, \text{MAC}(x)_n))$
  - $x = \sum x_i$
  - $\text{MAC}(x) = \sum \text{MAC}(x)_i$
- Key $\alpha$ is additively shared $\langle \alpha \rangle = (\alpha_1, \ldots, \alpha_n)$ where $\alpha = \sum \alpha_i$
- How do we compute?
- How and when do we verify the MAC?
Computing with Additive Shares + MAC

- Addition $[[x + y]] = [[x]] + [[y]] = ((x_1 + y_1, MAC(x)_1 + MAC(y)_1), \ldots, (x_n + y_n, MAC(x)_n + MAC(y)_n)$
Computing with Additive Shares + MAC

- Addition $[x + y] = [x] + [y] = ((x_1 + y_1, MAC(x)_1 + MAC(y)_1), \ldots, (x_n + y_n, MAC(x)_n + MAC(y)_n))$

- Multiplication with a constant
  $[c \cdot x] = c \cdot [x] = ((c \cdot x_1, c \cdot MAC(x)_1), \ldots, (c \cdot x_n, c \cdot MAC(x)_n))$
Computing with Additive Shares + MAC

- Addition $[x + y] = [x] + [y] =$ $((x_1 + y_1, \text{MAC}(x)_1 + \text{MAC}(y)_1), \ldots, (x_n + y_n, \text{MAC}(x)_n + \text{MAC}(y)_n))$

- Multiplication with a constant $[c \cdot x] = c \cdot [x] = ((c \cdot x_1, c \cdot \text{MAC}(x)_1), \ldots, (c \cdot x_n, c \cdot \text{MAC}(x)_n))$

- Adding a public value $[x + c] = [x] + c = ((x_1 + c, \text{MAC}(x)_1 + c \cdot \alpha_1), \ldots, (x_n, \text{MAC}(x)_n + c \cdot \alpha_n))$
Computing with Additive Shares + MAC

- Addition $[x + y] = [x] + [y] = ((x_1 + y_1, MAC(x)_1 + MAC(y)_1), \ldots, (x_n + y_n, MAC(x)_n + MAC(y)_n)$
- Multiplication with a constant $[c \cdot x] = c \cdot [x] = ((c \cdot x_1, c \cdot MAC(x)_1), \ldots, (c \cdot x_n, c \cdot MAC(x)_n)$
- Adding a public value $[x + c] = [x] + c = ((x_1 + c, MAC(x)_1 + c \cdot \alpha_1), \ldots, (x_n, MAC(x)_n + c \cdot \alpha_n)$
- Multiplication with Beaver triples remains the same as before
Computing with Additive Shares + MAC

- Addition $[x + y] = [x] + [y] = ((x_1 + y_1, \text{MAC}(x)_1 + \text{MAC}(y)_1), \ldots, (x_n + y_n, \text{MAC}(x)_n + \text{MAC}(y)_n)$

- Multiplication with a constant $[c \cdot x] = c \cdot [x] = ((c \cdot x_1, c \cdot \text{MAC}(x)_1), \ldots, (c \cdot x_n, c \cdot \text{MAC}(x)_n)$

- Adding a public value $[x + c] = [x] + c = ((x_1 + c, \text{MAC}(x)_1 + c \cdot \alpha_1), \ldots, (x_n, \text{MAC}(x)_n + c \cdot \alpha_n)$

- Multiplication with Beaver triples remains the same as before

- Sharing a value $x$ - we assume we have a random $[r]$ from preprocessing,
  - $[r]$ is published to the party
    - Can be done in preprocessing
    - Must be a verified opening
  - It computes and broadcast $x - r$
  - Players compute $[x] = [r] + (x - r)$
Computing with Additive Shares + MAC

- **Addition** $\llbracket x + y \rrbracket = \llbracket x \rrbracket + \llbracket y \rrbracket = ((x_1 + y_1, \text{MAC}(x)_1 + \text{MAC}(y)_1), \ldots, (x_n + y_n, \text{MAC}(x)_n + \text{MAC}(y)_n))$

- **Multiplication with a constant** $\llbracket c \cdot x \rrbracket = c \cdot \llbracket x \rrbracket = ((c \cdot x_1, c \cdot \text{MAC}(x)_1), \ldots, (c \cdot x_n, c \cdot \text{MAC}(x)_n))$

- **Adding a public value** $\llbracket x + c \rrbracket = \llbracket x \rrbracket + c = ((x_1 + c, \text{MAC}(x)_1 + c \cdot \alpha_1), \ldots, (x_n, \text{MAC}(x)_n + c \cdot \alpha_n))$

- **Multiplication with Beaver triples remains the same as before**

- **Sharing a value $x$ - we assume we have a random $\llbracket r \rrbracket$ from preprocessing,**
  - $\llbracket r \rrbracket$ is published to the party
  - Can be done in preprocessing
  - Must be a verified opening
  - It computes and broadcast $x - r$
  - Players compute $\llbracket x \rrbracket = \llbracket r \rrbracket + (x - r)$

- **Opening - how to verify the MAC?**
Opening

- Exchange shares $x_i$ of $\lfloor x \rfloor$ to learn $x = \sum x_i$
Opening

- Exchange shares $x_i$ of $[x]$ to learn $x = \sum x_i$
- Commit to shares $\alpha_i$ of the key
- Commit to shares $\text{MAC}(x)_i$ of the MAC
  - *Discussion* Commitments? Why?
Opening

- Exchange shares $x_i$ of $\lfloor x \rfloor$ to learn $x = \sum x_i$
- Commit to shares $\alpha_i$ of the key
- Commit to shares MAC($x$)$_i$ of the MAC
  - Discussion Commitments? Why?
- Open the commitments and compute $\alpha = \sum \alpha_i$
Opening

- Exchange shares $x_i$ of $[x]$ to learn $x = \sum x_i$
- Commit to shares $\alpha_i$ of the key
- Commit to shares $\text{MAC}(x)_i$ of the MAC
  - Discussion Commitments? Why?
- Open the commitments and compute $\alpha = \sum \alpha_i$
- Open the commitments and compute $\text{MAC}(x) = \sum \text{MAC}(x)_i$
Opening

- Exchange shares $x_i$ of $[x]$ to learn $x = \sum x_i$
- Commit to shares $\alpha_i$ of the key
- Commit to shares $\text{MAC}(x)_i$ of the MAC
- **Discussion** Commitments? Why?
- Open the commitments and compute $\alpha = \sum \alpha_i$
- Open the commitments and compute $\text{MAC}(x) = \sum \text{MAC}(x)_i$
- Check if $\text{MAC}(x) = \alpha x$
Opening

- Exchange shares $x_i$ of $[x]$ to learn $x = \sum x_i$
- Commit to shares $\alpha_i$ of the key
- Commit to shares $\text{MAC}(x)_i$ of the MAC
  - Discussion Commitments? Why?
- Open the commitments and compute $\alpha = \sum \alpha_i$
- Open the commitments and compute $\text{MAC}(x) = \sum \text{MAC}(x)_i$
- Check if $\text{MAC}(x) = \alpha x$
- The check passes if the opening was correct, or the adversary broke the MAC and knows the key
  - The adversary sees $x$
  - but needs to find a modifier for the MAC before seeing $\alpha$ or $\text{MAC}(x)$
Opening

- Exchange shares $x_i$ of $[x]$ to learn $x = \sum x_i$
- Commit to shares $\alpha_i$ of the key
- Commit to shares $\text{MAC}(x)_i$ of the MAC
  - Discussion Commitments? Why?
- Open the commitments and compute $\alpha = \sum \alpha_i$
- Open the commitments and compute $\text{MAC}(x) = \sum \text{MAC}(x)_i$
- Check if $\text{MAC}(x) = \alpha x$
- The check passes if the opening was correct, or the adversary broke the MAC and knows the key
  - The adversary sees $x$
  - but needs to find a modifier for the MAC before seeing $\alpha$ or $\text{MAC}(x)$
- This opening leaks the key $\alpha$ and therefore can be done only once. But we need two openings for each multiplication.
Opening v2

- Exchange shares $x_i$ of $[x]$ to learn $x = \sum x_i$
Opening v2

- Exchange shares $x_i$ of $\lfloor x \rfloor$ to learn $x = \sum x_i$
- Each party locally computes $d_i = \alpha_i x - \text{MAC}(x)_i$

How to check?

Players commit to $d_i$, then open.

Check that $\sum d_i = 0$

This is equivalent to the MAC security

Adversary has not seen $\alpha$ nor $\text{MAC}(x)$ before committing

Guessing a value to modify $d_i$ is the same as modifying ones MAC share

No need to publish $\alpha$ - can continue computing with these shares
Opening v2

- Exchange shares $x_i$ of $[x]$ to learn $x = \sum x_i$
- Each party locally computes $d_i = \alpha_i x - \text{MAC}(x)_i$
- $d = \sum d_i = \alpha x - \text{MAC}(x)$ need it to be 0.

How to check?
Players commit to $d_i$, then open.
Check that $\sum d_i = 0$
This is equivalent to the MAC security
Adversary has not seen $\alpha$ nor $\text{MAC}(x)$ before committing.
Guessing a value to modify $d_i$ is the same as modifying ones MAC share.
No need to publish $\alpha$ - can continue computing with these shares.
Opening v2

- Exchange shares $x_i$ of $\lfloor x \rfloor$ to learn $x = \sum x_i$
- Each party locally computes $d_i = \alpha_i x - \text{MAC}(x)_i$
- $d = \sum d_i = \alpha x - \text{MAC}(x)$ need it to be 0.
- How to check?
  - Players commit to $d_i$, then open
  - Check that $\sum d_i = 0$
Opening v2

- Exchange shares $x_i$ of $\lfloor x \rfloor$ to learn $x = \sum x_i$
- Each party locally computes $d_i = \alpha_i x - \text{MAC}(x)_i$
- $d = \sum d_i = \alpha x - \text{MAC}(x)$ need it to be 0.
- How to check?
  - Players commit to $d_i$, then open
  - Check that $\sum d_i = 0$
- This is equivalent to the MAC security
- Adversary has not seen $\alpha$ nor $\text{MAC}(x)$ before committing
  - Guessing a value to modify $d_i$ is the same as modifying ones MAC share
Opening v2

- Exchange shares $x_i$ of $[x]$ to learn $x = \sum x_i$
- Each party locally computes $d_i = \alpha_i x - \text{MAC}(x)_i$
- $d = \sum d_i = \alpha x - \text{MAC}(x)$ need it to be 0.
- How to check?
  - Players commit to $d_i$, then open
  - Check that $\sum d_i = 0$
- This is equivalent to the MAC security
- Adversary has not seen $\alpha$ nor $\text{MAC}(x)$ before committing
  - Guessing a value to modify $d_i$ is the same as modifying ones MAC share
- No need to publish $\alpha$ - can continue computing with these shares
SPDZ Execution

- Run secure setup to obtain the shares of the key $\alpha$

- Use preprocessing to obtain random values $[\[ r \]]$ and Beaver triples $[\[ a \]], [\[ b \]], [\[ c \]]$

- Run sharing protocol to obtain the inputs $[\[ x \]]$

- Run the desired computations (arithmetic operations) with partial openings (open without verification)

- Verify the partial openings of the computation phase all at once

- All multiplications can be done without verifying the openings

- Abort if the check on multiplication publishing fails

- Run the publishing phase, verify the correctness of opening

- Abort if the check fails

For more details on SPDZ see [Dam+12] and [Dam+13]
SPDZ Execution

- Run secure setup to obtain the shares of the key $\alpha$
- Use preprocessing to obtain random values $[r]$ and Beaver triples $[a], [b], [c]$
SPDZ Execution

- Run secure setup to obtain the shares of the key $\alpha$
- Use preprocessing to obtain random values $\[ r \]$ and Beaver triples $\[ a \], \[ b \], \[ c \]$
- Run sharing protocol to obtain the inputs $\[ x \]$
SPDZ Execution

- Run secure setup to obtain the shares of the key $\alpha$
- Use preprocessing to obtain random values $[r]$ and Beaver triples $[a], [b], [c]$
- Run sharing protocol to obtain the inputs $[x]$
- Run the desired computations (arithmetic operations) with partial openings (open without verification)
SPDZ Execution

- Run secure setup to obtain the shares of the key $\alpha$
- Use preprocessing to obtain random values $[r]$ and Beaver triples $[a], [b], [c]$
- Run sharing protocol to obtain the inputs $[x]$
- Run the desired computations (arithmetic operations) with partial openings (open without verification)
- Verify the partial openings of the computation phase all at once
  - All multiplications can be done without verifying the openings
    - Because we open random values
  - Abort if the check on multiplication publishing fails
SPDZ Execution

- Run secure setup to obtain the shares of the key $\alpha$
- Use preprocessing to obtain random values $[r]$ and Beaver triples $[a], [b], [c]$
- Run sharing protocol to obtain the inputs $[x]$
- Run the desired computations (arithmetic operations) with partial openings (open without verification)
- Verify the partial openings of the computation phase all at once
  - All multiplications can be done without verifying the openings
    - Because we open random values
  - Abort if the check on multiplication publishing fails
- Run the publishing phase, verify the correctness of opening
  - Abort if the check fails

For more details on SPDZ see [Dam+12] and [Dam+13]
SPDZ Execution

- Run secure setup to obtain the shares of the key \( \alpha \)
- Use preprocessing to obtain random values \([r]\) and Beaver triples \([a], [b], [c]\)
- Run sharing protocol to obtain the inputs \([x]\)
- Run the desired computations (arithmetic operations) with partial openings (open without verification)
- Verify the partial openings of the computation phase all at once
  - All multiplications can be done without verifying the openings
    - Because we open random values
    - Abort if the check on multiplication publishing fails
- Run the publishing phase, verify the correctness of opening
  - Abort if the check fails
- For more details on SPDZ see [Dam+12] and [Dam+13]
Homomorphic Encryption (HE)

- Public key encryption
  - Encryption $\text{Enc}_{pk}(a)$
  - Decryption $\text{Dec}_{sk}(\text{Enc}_{pk}(a)) = a$

Examples
- Multiplicatively homomorphic: RSA (textbook version), ElGamal
- Additively homomorphic: Paillier scheme
Homomorphic Encryption (HE)

- Public key encryption
  - Encryption $\text{Enc}_{pk}(a)$
  - Decryption $\text{Dec}_{sk}(\text{Enc}_{pk}(a)) = a$
- Additively homomorphic $\text{Enc}_{pk}(a + b) = \text{Enc}_{pk}(a) + \text{Enc}_{pk}(b)$
Homomorphic Encryption (HE)

- Public key encryption
  - Encryption $\text{Enc}_{pk}(a)$
  - Decryption $\text{Dec}_{sk}(\text{Enc}_{pk}(a)) = a$
- Additively homomorphic $\text{Enc}_{pk}(a + b) = \text{Enc}_{pk}(a) + \text{Enc}_{pk}(b)$
- Multiplicatively homomorphic $\text{Enc}_{pk}(a \cdot b) = \text{Enc}_{pk}(a) \cdot \text{Enc}_{pk}(b)$
- Examples
  - Multiplicatively homomorphic: RSA (textbook version), ElGamal
  - Additively homomorphic: Paillier scheme
Multiplication with Additive HE

- One party knows \( a \) the other party knows \( b \), *Discussion* How to compute \( \text{Enc}_{pk}(ab) \)?

- Party one encrypts \( \text{Enc}_{pk}(a) \) sends it to the second party
- The second party computes \( b \cdot \text{Enc}_{pk}(a) = \text{Enc}_{pk}(ab) \) and sends it back

*Discussion*

Uses?
- If the first party knows the secret key then it gets \( ab \) (can send it to the second party)
- If some other party needs the multiplication result then their key can be used
- If we need additive shares of the result
  Second party computes \( b \cdot \text{Enc}_{pk}(a) - \text{Enc}_{pk}(r) \) for some random \( r \)
  The first party decrypts \( d = ab - r, r + d = ab \)
Multiplication with Additive HE

- One party knows $a$ the other party knows $b$, *Discussion* How to compute $\text{Enc}_{pk}(ab)$?
- Party one encrypts $\text{Enc}_{pk}(a)$ sends it to the second party
- The second party computes $b \cdot \text{Enc}_{pk}(a) = \text{Enc}_{pk}(ab)$ and sends it back
Multiplication with Additive HE

- One party knows $a$, the other party knows $b$, \textit{Discussion} How to compute $\text{Enc}_{pk}(ab)$?
- Party one encrypts $\text{Enc}_{pk}(a)$ sends it to the second party
- The second party computes $b \cdot \text{Enc}_{pk}(a) = \text{Enc}_{pk}(ab)$ and sends it back
- \textit{Discussion} Uses?

April 8, 2020
Multiplication with Additive HE

- One party knows $a$ the other party knows $b$, Discussion How to compute $\text{Enc}_{pk}(ab)$?
- Party one encrypts $\text{Enc}_{pk}(a)$ sends it to the second party
- The second party computes $b \cdot \text{Enc}_{pk}(a) = \text{Enc}_{pk}(ab)$ and sends it back
- Discussion Uses?
  - If the first party knows the secret key then it gets $ab$ (can send it to the second party)
Multiplication with Additive HE

- One party knows $a$ the other party knows $b$, Discussion How to compute $\text{Enc}_{pk}(ab)$?
- Party one encrypts $\text{Enc}_{pk}(a)$ sends it to the second party
- The second party computes $b \cdot \text{Enc}_{pk}(a) = \text{Enc}_{pk}(ab)$ and sends it back
- Discussion Uses?
  - If the first party knows the secret key then it gets $ab$ (can send it to the second party)
  - If some other party needs the multiplication result then their key can be used
Multiplication with Additive HE

- One party knows $a$ the other party knows $b$, Discussion How to compute $\text{Enc}_{pk}(ab)$?
- Party one encrypts $\text{Enc}_{pk}(a)$ sends it to the second party
- The second party computes $b \cdot \text{Enc}_{pk}(a) = \text{Enc}_{pk}(ab)$ and sends it back
- Discussion Uses?
  - If the first party knows the secret key then it gets $ab$ (can send it to the second party)
  - If some other party needs the multiplication result then their key can be used
  - If we need additive shares of the result
    - Second party computes $b \cdot \text{Enc}_{pk}(a) - \text{Enc}_{pk}(r)$ for some random $r$
    - The first party decrypts $d = ab - r$
    - $r + d = ab$
HE in MPC

- Can be used to compute any functionality
  - Straightforward for two parties
  - Computationally heavy

Generating Beaver triples - random

\[
\begin{bmatrix}
a \\
b \\
ab
\end{bmatrix}
\]

Generating the random values (adding the MAC to the shared value)
HE in MPC

- Can be used to compute any functionality
  - Straightforward for two parties
  - Computationally heavy
- Often used for precomputation
  - Sometimes with threshold decryption
    - No one party has the decryption key
    - Some parties have shares of the key
    - A collection of parties can decrypt together
- Generating Beaver triples - random $[a]$ and $[b]$ and $[ab]$
- Generating the random values (adding the MAC to the shared value)
Somewhat Homomorphic Cryptosystem (SHE)

- Public key encryption with homomorphic properties for addition and multiplication
- $\text{Enc}_{pk}(a + b) = \text{Enc}_{pk}(a) + \text{Enc}_{pk}(b)$
- $\text{Enc}_{pk}(a \cdot b) = \text{Enc}_{pk}(a) \cdot \text{Enc}_{pk}(b)$

Each operation adds some noise.

Decryption:
- $\text{Dec}_{sk}(\text{Enc}_{pk}(a)) = a$ if there is not too much noise

For example, any number of additions, but one multiplication is allowed; hence, somewhat homomorphic.
Somewhat Homomorphic Cryptosystem (SHE)

- Public key encryption with homomorphic properties for addition and multiplication
  \[ \text{Enc}_{pk}(a + b) = \text{Enc}_{pk}(a) + \text{Enc}_{pk}(b) \]
  \[ \text{Enc}_{pk}(a \cdot b) = \text{Enc}_{pk}(a) \cdot \text{Enc}_{pk}(b) \]
- Each operation adds some noise
- \( \text{Dec}_{sk}(\text{Enc}_{pk}(a)) = a \) if there is not too much noise
- For example any number of additions, but one multiplication is allowed
  - hence, somewhat homomorphic
Additive Triple Generation with SHE

- Each party $i$ chooses $a_i, b_i, r_i$ at random
- Broadcast $\text{Enc}_{pk}(a_i), \text{Enc}_{pk}(b_i), \text{Enc}_{pk}(r_i)$
Additive Triple Generation with SHE

- Each party $i$ chooses $a_i$, $b_i$, $r_i$ at random
- Broadcast $\text{Enc}_{pk}(a_i)$, $\text{Enc}_{pk}(b_i)$, $\text{Enc}_{pk}(r_i)$
- $\text{Enc}_{pk}(a) = \sum \text{Enc}_{pk}(a_i)$
- $\text{Enc}_{pk}(b) = \sum \text{Enc}_{pk}(b_i)$
- $\text{Enc}_{pk}(r) = \sum \text{Enc}_{pk}(r_i)$

$D = \text{Enc}_{pk}(a) \cdot \text{Enc}_{pk}(b)$

Compute and (threshold) decrypt $d = \text{Dec}_{sk}(D - \text{Enc}_{pk}(r)) = ab - r$,

Set $c_1 = r_1 + d$, $c_i = r_i$ to define $[c]$
Additive Triple Generation with SHE

- Each party $i$ chooses $a_i$, $b_i$, $r_i$ at random
- Broadcast $\text{Enc}_{pk}(a_i), \text{Enc}_{pk}(b_i), \text{Enc}_{pk}(r_i)$
- $\text{Enc}_{pk}(a) = \sum \text{Enc}_{pk}(a_i)$
- $\text{Enc}_{pk}(b) = \sum \text{Enc}_{pk}(b_i)$
- $\text{Enc}_{pk}(r) = \sum \text{Enc}_{pk}(r_i)$
- $D = \text{Enc}_{pk}(a) \cdot \text{Enc}_{pk}(b)$
- Compute and (threshold) decrypt $d = \text{Dec}_{sk}(D - \text{Enc}_{pk}(r)) = ab - r$
- Set $c_1 = r_1 + d$, $c_i = r_i$ to define $\llbracket c \rrbracket$
  - $c = r + d = ab$
Fully Homomorphic Encryption (FHE)

- Infinite number of operations is allowed
- *Discussion* Can SHE be extended to FHE?

FHE = SHE + Bootstrapping

- Do the noise reduction (e.g. decryption computation) under encryption
- The trick is to encode the ciphertext into the circuit
- Encrypted decryption key is the input (noise of the output only depends on noise here)
- We need SHE to allow at least operations for the noise removal + 1 real operation that we want to make

With FHE any operation can be securely computed on encrypted inputs

However, current schemes require a lot of computation power
Fully Homomorphic Encryption (FHE)

- Infinite number of operations is allowed
- Discussion Can SHE be extended to FHE?
- $\text{FHE} = \text{SHE} + \text{Bootstrapping}$
  - Do the noise reduction (e.g. decryption computation) under encryption
    - The trick is to encode the ciphertext into the circuit
    - Encrypted decryption key is the input (noise of the output only depends on noise here)
  - We need SHE to allow at least operations for the noise removal + 1 real operation that we want to make
Fully Homomorphic Encryption (FHE)

- Infinite number of operations is allowed

**Discussion** Can SHE be extended to FHE?

FHE = SHE + Bootstrapping

- Do the noise reduction (e.g. decryption computation) under encryption
  - The trick is to encode the ciphertext into the circuit
  - Encrypted decryption key is the input (noise of the output only depends on noise here)

- We need SHE to allow at least operations for the noise removal + 1 real operation that we want to make

- With FHE any operation can be securely computed on encrypted inputs
  - However, current schemes require a lot of computation power
Garbled Circuits (GC)

- This approach actually started the MPC field
- Two parties, one has input $a$ and the other $b$
  - We call them the garbler and the evaluator
Garbled Circuits (GC)

- This approach actually started the MPC field
- Two parties, one has input a and the other b
  - We call them the garbler and the evaluator
- Circuit - the Boolean circuit of the desired functionality
  - Commonly XOR and AND gates
Garbled Circuits (GC)

- This approach actually started the MPC field
- Two parties, one has input $a$ and the other $b$
  - We call them the garbler and the evaluator
- Circuit - the Boolean circuit of the desired functionality
  - Commonly XOR and AND gates
- Garbling - encrypting the circuit
GC Protocol

- The garbler garbles the circuit
- The garbler sends the garbled circuit to the evaluator
- The garbler and evaluator interact to give some more information to the evaluator
- The evaluator decrypts the circuit to learn the computation outcome
Garbling

Discussion How to approach encrypting a circuit so that it can still be evaluated?
Discussion How to approach encrypting a circuit so that it can still be evaluated?
Gate-by-gate using the truth tables
### Garbling

**Discussion** How to approach encrypting a circuit so that it can still be evaluated?
- Gate-by-gate using the truth tables

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$a \land b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$a \oplus b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Encrypting the Truth Table

- Choose keys $w_a^0$, $w_a^1$, $w_b^0$, $w_b^1$
- Choose keys $w^1$ and $w^0$ for the output of the gate
Encrypting the Truth Table

- Choose keys $w_a^0$, $w_a^1$, $w_b^0$, $w_b^1$
- Choose keys $w_1$ and $w_0$ for the output of the gate

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$a \land b$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

AND gate becomes

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a \land b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Enc_{w_a^0}(Enc_{w_b^0}(w_0))</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Enc_{w_a^0}(Enc_{w_b^1}(w_0))</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Enc_{w_a^1}(Enc_{w_b^0}(w_0))</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Enc_{w_a^1}(Enc_{w_b^1}(w_1))</td>
</tr>
</tbody>
</table>

April 8, 2020
Encrypting the Truth Table

- Choose keys $w^0_a$, $w^1_a$, $w^0_b$, $w^1_b$
- Choose keys $w^1$ and $w^0$ for the output of the gate

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a \land b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **AND gate** becomes

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a \land b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$\text{Enc}<em>{w^0_a}(\text{Enc}</em>{w^0_b}(w^0))$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$\text{Enc}<em>{w^0_a}(\text{Enc}</em>{w^1_b}(w^0))$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$\text{Enc}<em>{w^1_a}(\text{Enc}</em>{w^0_b}(w^0))$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$\text{Enc}<em>{w^1_a}(\text{Enc}</em>{w^1_b}(w^1))$</td>
</tr>
</tbody>
</table>

- So when you have the keys $w^a_a$ and $w^b_b$ then you can decrypt the ciphertext that gives $w_{a \land b}$
Encrypting the Truth Table

- Choose keys $w_a^0$, $w_a^1$, $w_b^0$, $w_b^1$
- Choose keys $w^1$ and $w^0$ for the output of the gate

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$a \land b$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- AND gate becomes

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$a \land b$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>$\text{Enc}<em>{w_a^0}(\text{Enc}</em>{w_b^0}(w^0))$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$\text{Enc}<em>{w_a^0}(\text{Enc}</em>{w_b^0}(w^0))$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$\text{Enc}<em>{w_a^1}(\text{Enc}</em>{w_b^0}(w^0))$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$\text{Enc}<em>{w_a^1}(\text{Enc}</em>{w_b^1}(w^1))$</td>
<td></td>
</tr>
</tbody>
</table>

- So when you have the keys $w_a^a$ and $w_b^b$ then you can decrypt the ciphertext that gives $w_{a \land b}$
- We consider the randomly permuted \textit{(why?)} order of $\text{Enc}_{w_a^0}(\text{Enc}_{w_b^b}(w^0))$, $\text{Enc}_{w_a^0}(\text{Enc}_{w_b^b}(w^0))$, $\text{Enc}_{w_a^1}(\text{Enc}_{w_b^0}(w^0))$, $\text{Enc}_{w_a^1}(\text{Enc}_{w_b^1}(w^1))$ to be the encryption of this table
Encrypting the Truth Table

- Choose keys $w_a^0$, $w_a^1$, $w_b^0$, $w_b^1$
- Choose keys $w^1$ and $w^0$ for the output of the gate

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$a \land b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- AND gate becomes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$a \land b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$\text{Enc}<em>{w_a^0}(\text{Enc}</em>{w_b^0}(w^0))$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$\text{Enc}<em>{w_a^0}(\text{Enc}</em>{w_b^0}(w^0))$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$\text{Enc}<em>{w_a^1}(\text{Enc}</em>{w_b^0}(w^0))$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$\text{Enc}<em>{w_a^1}(\text{Enc}</em>{w_b^1}(w^1))$</td>
</tr>
</tbody>
</table>

- So when you have the keys $w_a^a$ and $w_b^b$ then you can decrypt the ciphertext that gives $w_{a \land b}$
- We consider the randomly permuted (*why?*) order of $\text{Enc}_{w_a^0}(\text{Enc}_{w_b^0}(w^0))$, $\text{Enc}_{w_a^0}(\text{Enc}_{w_b^1}(w^0))$, $\text{Enc}_{w_a^1}(\text{Enc}_{w_b^0}(w^0))$, $\text{Enc}_{w_a^1}(\text{Enc}_{w_b^1}(w^1))$ to be the encryption of this table
- Same approach applies to all other gates
Evaluating the Gate

- The evaluator receives the keys $w^a$ and $w^b$
- It decrypts the ciphertexts
- Note that there must be some way for the evaluator to know which decryption succeeds
  - some extra marker in the plaintext to mark success
  - or some way to see which ciphertext to approach with these keys
- Hence, it recognizes when it decrypts $\text{Enc}_{w^a}(\text{Enc}_{w^b}(w^c))$
- $w^c$ can be either the output bit or the next key
Garbling and Evaluating the Circuit [Rom17]

Circuit: \((W_1 \land W_2) \oplus (W_3 \lor W_4)\)

1) Choose keys \(w_i^0, w_i^1\) for each wire \(W_i\)

We use colours to distinguish the keys when decrypting.

\[
\begin{align*}
W_1 & \quad W_2 \\
\bullet w_1^0 / \bullet w_1^1 \\
\bullet w_2^0 / \bullet w_2^1 \\
\end{align*}
\]

\[
\begin{align*}
W_3 & \quad W_4 \\
\bullet w_3^0 / \bullet w_3^1 \\
\bullet w_4^0 / \bullet w_4^1 \\
\end{align*}
\]
Garbling and Evaluating the Circuit [Rom17]

Circuit: \((W_1 \land W_2) \oplus (W_3 \lor W_4)\)

2) Encrypt the tables with the respective keys

Permute according to the colours

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td>••</td>
<td>Enc_{w_1^1, w_2^0}(w_5^0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_1^1, w_2^1}(w_5^1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_1^0, w_2^0}(w_5^0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_1^0, w_2^1}(w_5^1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td>••</td>
<td>Enc_{w_3^1, w_4^0}(w_6^0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_3^1, w_4^1}(w_6^1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_3^0, w_4^0}(w_6^0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_3^0, w_4^1}(w_6^1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td>••</td>
<td>Enc_{w_5^1, w_6^0}(1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_5^1, w_6^1}(0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_5^0, w_6^0}(0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_5^0, w_6^1}(1)</td>
</tr>
</tbody>
</table>
Garbling and Evaluating the Circuit [Rom17]

Circuit: \((W_1 \land W_2) \oplus (W_3 \lor W_4)\)

3) We start the evaluation with the keys corresponding to the real input bits (in bold)

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td>••</td>
<td>Enc(<em>{w_1}^{1},w_2^{0})(</em>{w_6^{0}})</td>
</tr>
<tr>
<td>••</td>
<td>Enc(<em>{w_1}^{1},w_2^{1})(</em>{w_5^{1}})</td>
</tr>
<tr>
<td>••</td>
<td>Enc(<em>{w_1}^{0},w_2^{0})(</em>{w_5^{0}})</td>
</tr>
<tr>
<td>••</td>
<td>Enc(<em>{w_1}^{0},w_2^{1})(</em>{w_5^{1}})</td>
</tr>
<tr>
<td>••</td>
<td>Enc(<em>{w_3}^{1},w_4^{1})(</em>{w_6^{1}})</td>
</tr>
<tr>
<td>••</td>
<td>Enc(<em>{w_3}^{1},w_4^{0})(</em>{w_6^{0}})</td>
</tr>
<tr>
<td>••</td>
<td>Enc(<em>{w_3}^{0},w_4^{1})(</em>{w_6^{1}})</td>
</tr>
<tr>
<td>••</td>
<td>Enc(<em>{w_3}^{0},w_4^{0})(</em>{w_6^{0}})</td>
</tr>
</tbody>
</table>

Order | Garbled value |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>••</td>
<td>Enc(<em>{w_1}^{1},w_6^{0})(</em>{1})</td>
</tr>
<tr>
<td>••</td>
<td>Enc(<em>{w_1}^{1},w_6^{1})(</em>{0})</td>
</tr>
<tr>
<td>••</td>
<td>Enc(<em>{w_5}^{1},w_6^{1})(</em>{0})</td>
</tr>
<tr>
<td>••</td>
<td>Enc(<em>{w_5}^{0},w_6^{1})(</em>{1})</td>
</tr>
</tbody>
</table>
Garbling and Evaluating the Circuit [Rom17]

Circuit: \((W_1 \land W_2) \oplus (W_3 \lor W_4)\)

4) Evaluator decrypts the respective rows (in bold) in the first layer of truth tables to obtain the keys for the next layer.

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td>••</td>
<td>Enc_{w_1^1, w_2^0}(\cdot w_2^0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_1^1, w_2^1}(\cdot w_1^1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_0^0, w_1^1}(\cdot w_0^0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_1^1, w_2^1}(\cdot w_0^0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td>••</td>
<td>Enc_{w_3^1, w_4^0}(\cdot w_6^1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_3^1, w_4^0}(\cdot w_6^1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_3^1, w_4^1}(\cdot w_6^1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_3^1, w_4^0}(\cdot w_6^0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td>••</td>
<td>Enc_{w_5^1, w_6^0}(1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_5^1, w_6^0}(0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_5^0, w_6^0}(0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_5^0, w_6^1}(1)</td>
</tr>
</tbody>
</table>
Garbling and Evaluating the Circuit [Rom17]

Circuit: \((W_1 \land W_2) \oplus (W_3 \lor W_4)\)

5) Evaluator decrypts the last layer to obtain the real output bits

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td>••</td>
<td>Enc_{w_1^1,w_2^0}(w_5^0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_1^1,w_2^1}(w_5^1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_0^1,w_2^0}(w_5^0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_0^1,w_2^1}(w_5^1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td>••</td>
<td>Enc_{w_3^1,w_4^1}(w_6^1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_3^1,w_4^0}(w_6^1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_3^0,w_4^1}(w_6^0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_3^0,w_4^0}(w_6^0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td>••</td>
<td>Enc_{w_5^1,w_6^0}(1)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_5^1,w_6^1}(0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_5^0,w_6^0}(0)</td>
</tr>
<tr>
<td>••</td>
<td>Enc_{w_5^0,w_6^1}(1)</td>
</tr>
</tbody>
</table>
Garbling and Evaluating the Circuit [Rom17]

Circuit: \((W_1 \land W_2) \oplus (W_3 \lor W_4)\)
Computation: \((0 \land 0) \oplus (1 \lor 0) = 0 \oplus 1 = 1\)

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enc_{w_1^1,w_2^0}(w_3^0)</td>
</tr>
<tr>
<td></td>
<td>Enc_{w_1^1,w_2^1}(w_3^1)</td>
</tr>
<tr>
<td></td>
<td>Enc_{w_1^0,w_2^0}(w_3^0)</td>
</tr>
<tr>
<td></td>
<td>Enc_{w_1^0,w_2^1}(w_3^1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enc_{w_3^1,w_4^0}(w_5^0)</td>
</tr>
<tr>
<td></td>
<td>Enc_{w_3^1,w_4^1}(w_5^1)</td>
</tr>
<tr>
<td></td>
<td>Enc_{w_3^0,w_4^0}(w_5^0)</td>
</tr>
<tr>
<td></td>
<td>Enc_{w_3^0,w_4^1}(w_5^1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th>Garbled value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enc_{w_5^1,w_6^0}(1)</td>
</tr>
<tr>
<td></td>
<td>Enc_{w_5^1,w_6^1}(0)</td>
</tr>
<tr>
<td></td>
<td>Enc_{w_5^0,w_6^0}(0)</td>
</tr>
<tr>
<td></td>
<td>Enc_{w_5^0,w_6^1}(1)</td>
</tr>
</tbody>
</table>
Communicating the Circuit and the Keys

- The garbler sends the garbled circuit to the evaluator
- For each of its input wire $W_i$ that has value $a_i$ it can send $w_i^{a_i}$
- Discussion What should we do with the inputs of the evaluator?
Communicating the Circuit and the Keys

- The garbler sends the garbled circuit to the evaluator
- For each of its input wire $W_i$ that has value $a_i$ it can send $w_i^{a_i}$
- **Discussion** What should we do with the inputs of the evaluator?
  - Sending both keys allows the evaluator to decrypt too much
  - Asking for the right key from the garbler leaks the input of the evaluator
  - Solution: Oblivious transfer (later in this lecture)
GC Protocol

- The garbler garbles the circuit gate-by-gate
- The garbler sends the garbled circuit to the evaluator
- The garbler sends the input keys of their input to the evaluator
- The evaluator obtains the keys for their input with oblivious transfer
- The evaluator decrypts the circuit gate-by-gate to learn the computation outcome
Trusted Execution Environment (TEE)

- Secure hardware (and software surrounding it), e.g. Intel SGX, ARM TrustZone
Trusted Execution Environment (TEE)

- Secure hardware (and software surrounding it), e.g. Intel SGX, ARM TrustZone
- Secure an area of the processor
  - Code is executed correctly
  - Check that the expected code runs in genuine TEE (attestation)
  - Only authorized code can run
  - Only expected communication between an app in TEE and outside world
- Data is only accessible to the code in this area

This could become the killer solution for secure computation

BUT: so far it has been error-prone
BUT: probably always need to trust the manufacturer
Trusted Execution Environment (TEE)

- Secure hardware (and software surrounding it), e.g. Intel SGX, ARM TrustZone
- Secure an area of the processor
  - Code is executed correctly
  - Check that the expected code runs in genuine TEE (attestation)
  - Only authorized code can run
  - Only expected communication between an app in TEE and outside world
- Data is only accessible to the code in this area
- This could become the killer solution for secure computation
Trusted Execution Environment (TEE)

- Secure hardware (and software surrounding it), e.g. Intel SGX, ARM TrustZone
- Secure an area of the processor
  - Code is executed correctly
  - Check that the expected code runs in genuine TEE (attestation)
  - Only authorized code can run
  - Only expected communication between an app in TEE and outside world
- Data is only accessible to the code in this area
- This could become the killer solution for secure computation
  - BUT: so far it has been error-prone
  - BUT: probably always need to trust the manufacturer
## Summary of Generic MPC

<table>
<thead>
<tr>
<th></th>
<th>Secret sharing</th>
<th>(F)HE</th>
<th>GC</th>
<th>TEE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Communication</strong></td>
<td>Each operation</td>
<td>Sending inputs and results</td>
<td>Sending the circuit, OT for inputs</td>
<td>Sending inputs and results</td>
</tr>
<tr>
<td><strong>Computations</strong></td>
<td>Modular arithmetic</td>
<td>Public key encryption</td>
<td>Symmetric crypto for garbling, public key for OT</td>
<td>Public key crypto, hardware specifications</td>
</tr>
<tr>
<td><strong>Assumptions</strong></td>
<td>Information theoretic security (often still computational)</td>
<td>Computational security</td>
<td>Computational security</td>
<td>Computational security, verification of the manufacturing</td>
</tr>
</tbody>
</table>
Part II: Special Purpose Protocols
Private Information Retrieval (PIR)

- PIR problem:
  - One party has a set of data $x_1, \ldots, x_n$
  - Other party wants to retrieve some $x_i$ without disclosing $i$
Private Information Retrieval (PIR)

- PIR problem:
  - One party has a set of data $x_1, \ldots, x_n$
  - Other party wants to retrieve some $x_i$ without disclosing $i$
- Discussion Solution?
Private Information Retrieval (PIR)

- PIR problem:
  - One party has a set of data $x_1, \ldots, x_n$
  - Other party wants to retrieve some $x_i$ without disclosing $i$

- Discussion Solution?
  - First party sends $x_1, \ldots, x_n$ to the second party
Private Information Retrieval (PIR)

- PIR problem:
  - One party has a set of data $x_1, \ldots, x_n$
  - Other party wants to retrieve some $x_i$ without disclosing $i$
- Discussion Solution?
  - First party sends $x_1, \ldots, x_n$ to the second party
- Non-trivial PIR
  - Send less information than the whole database
  - Can not be done with perfect privacy if sender is one party [Cho+95]
    - Either use cryptography
    - or use multiple senders
PIR with Crypto

- Computational security
- *Discussion* Any ideas how to get the protocol?
PIR with Crypto

- Computational security
- Discussion Any ideas how to get the protocol?
- Use homomorphic encryption
PIR with Crypto

- Computational security
- **Discussion** Any ideas how to get the protocol?
- Use homomorphic encryption
- Receiver encrypts $\text{Enc}(i)$ and sends to the server
- The server computes $\sum_{j=1}^{n} \delta_{ij} \cdot x_j = x_i$ under encryption, sends $\text{Enc}(x_i)$ to the receiver
  - Kronecker delta $\delta_{ij} = \begin{cases} 
1, & \text{if } i = j \\
0, & \text{otherwise}
\end{cases}$
  - for our case $\delta_{ij} = \prod_{\ell=1, \ell \neq j}^{n} \frac{i-\ell}{j-\ell} = \begin{cases} 
1, & i = j \\
0, & i \neq j, \ i \leq n
\end{cases}$
- The receiver decrypts $\text{Enc}(x_i)$ to get $x_i$
PIR with Multiple Senders

- Many senders, all have the same data $x_1, \ldots, x_n$
- Discussion How to get information theoretic security?

\[ \hat{x} = (x_1, \ldots, x_n) \]

For the receivers choice define
\[ \hat{i} = (\delta_{i1}, \ldots, \delta_{in}) \]

Vector of zeros, except for 1 in \( i \)'th position

Dot product
\[ \hat{i} \cdot \hat{x} = \sum_j \delta_{ij} \cdot x_j = x_i \]

Use secret sharing

Receiver secret shares \( \hat{i}_1, \hat{i}_2 \) (each component separately)

Send \( \hat{i}_j \) to \( j \)'th sender

Sender computes \( \hat{i}_j \cdot \hat{x} \) and sends it back

Receiver sums \( \hat{i}_1 \cdot \hat{x} + \hat{i}_2 \cdot \hat{x} = (\hat{i}_1 + \hat{i}_2) \cdot \hat{x} = \hat{i} \cdot \hat{x} = x_i \)

This still has communication linear in \( n \)

Can be extended to arrange the data as a matrix and receiver sends two indices - communication linear in $\sqrt{n}$.
PIR with Multiple Senders

- Many senders, all have the same data $x_1, \ldots, x_n$
- **Discussion** How to get information theoretic security?
- Let’s say the database is $\hat{x} = (x_1, \ldots, x_n)$
- For the receivers choice define $\hat{i} = (\delta_{i1}, \ldots, \delta_{in})$
  - Vector of zeros, except for 1 in $i$’th position
PIR with Multiple Senders

- Many senders, all have the same data \( x_1, \ldots, x_n \)
- **Discussion** How to get information theoretic security?
- Let’s say the database is \( \hat{x} = (x_1, \ldots, x_n) \)
- For the receivers choice define \( \hat{i} = (\delta_{i1}, \ldots, \delta_{in}) \)
  - Vector of zeros, except for 1 in \( i \)'th position
- Dot product \( \hat{i} \cdot \hat{x} = \sum_j \delta_{ij} \cdot x_j = x_i \)
PIR with Multiple Senders

- Many senders, all have the same data $x_1, \ldots, x_n$
- *Discussion* How to get information theoretic security?
- Let’s say the database is $\hat{x} = (x_1, \ldots, x_n)$
- For the receivers choice define $\hat{i} = (\delta_{i1}, \ldots, \delta_{in})$
  - Vector of zeros, except for 1 in $i$’th position
- Dot product $\hat{i} \cdot \hat{x} = \sum_j \delta_{ij} \cdot x_j = x_i$
- Use secret sharing
PIR with Multiple Senders

- Many senders, all have the same data $x_1, \ldots, x_n$
- **Discussion** How to get information theoretic security?
- Let’s say the database is $\hat{x} = (x_1, \ldots, x_n)$
- For the receivers choice define $\hat{i} = (\delta_{i1}, \ldots, \delta_{in})$
  - Vector of zeros, except for 1 in $i$’th position
- Dot product $\hat{i} \cdot \hat{x} = \sum_j \delta_{ij} \cdot x_j = x_i$
- Use secret sharing
  - Receiver secret shares $\hat{i} = \hat{i}_1 + \hat{i}_2$ (each component separately)
  - Send $\hat{i}_j$ to $j$’th sender
  - Sender computes $\hat{i}_j \cdot \hat{x}$ and sends it back
  - Receiver sums $\hat{i}_1 \cdot \hat{x} + \hat{i}_2 \cdot \hat{x} = (\hat{i}_1 + \hat{i}_2) \cdot \hat{x} = \hat{i} \cdot \hat{x} = x_i$
- This still has communication linear in $n$
  - Can be extended to arrange the data as a matrix and receiver sends two indices - communication linear in $\sqrt{n}$
Oblivious Transfer (OT)

- **OT problem:**
  - The first party has two inputs \( x_0, x_1 \)
  - The second party wants to retrieve some \( x_i \)
  - The second party must not learn \( x_j \) for \( j \neq i \)
  - The first party must not learn \( i \)

First proposed in 1981 in [Rab81]

There the problem was that some data was sent but the sender did not know if the receiver got the information

There are variations:
- Random OT
- Correlated OT
- Can also have 1-out-of-\( n \) OT or \( k \)-out-of-\( n \) OT

The variations are equivalent and can be built from one another
Oblivious Transfer (OT)

- OT problem:
  - The first party has two inputs $x_0, x_1$
  - The second party wants to retrieve some $x_i$
  - The second party must not learn $x_j$ for $j \neq i$
  - The first party must not learn $i$

- First proposed in 1981 in [Rab81]
  - There the problem was that some data was sent but the sender did not know if the receiver got the information
Oblivious Transfer (OT)

- **OT problem:**
  - The first party has two inputs $x_0, x_1$
  - The second party wants to retrieve some $x_i$
  - The second party must not learn $x_j$ for $j \neq i$
  - The first party must not learn $i$

- First proposed in 1981 in [Rab81]
  - There the problem was that some data was sent but the sender did not know if the receiver got the information

- There are variations:
  - Random OT
  - Correlated OT
  - Can also have 1-out-of-$n$ OT or $k$-out-of-$n$ OT

- The variations are equivalent and can be built from one another
OT and Public Key Encryption [Ger+00]

- There is no trivial OT
OT and Public Key Encryption [Ger+00]

- There is no trivial OT
- There is no OT from black-box public key encryption
  - You can get OT from public key encryption with some more specific assumptions
    - e.g. homomorphism
OT and Public Key Encryption [Ger+00]

- There is no trivial OT
- There is no OT from black-box public key encryption
  - You can get OT from public key encryption with some more specific assumptions
    - e.g. homomorphism
- One round OT implies existence of public key encryption
  - General black box OT does not
OT and Public Key Encryption [Ger+00]

- There is no trivial OT
- There is no OT from black-box public key encryption
  - You can get OT from public key encryption with some more specific assumptions
    - e.g. homomorphism
- One round OT implies existence of public key encryption
  - General black box OT does not
- OT is an interesting cryptographic primitive on its own
  - both in theory and practice
Aiello-Ishai-Reingold OT [AIR01]

Discussion OT using additively homomorphic encryption
Aiello-Ishai-Reingold OT [AIR01]

- **Discussion** OT using additively homomorphic encryption
- The receiver encrypts $\text{Enc}(i)$ and sends to the sender
- The sender chooses two random values $r_0$ and $r_1$
- The sender computes $\text{Enc}(x_0 + i \cdot r_0)$ and $\text{Enc}(x_1 + (i - 1) \cdot r_1)$ and sends back

if $i = 0$ then
  $x_0 + i \cdot r_0 = x_0$
if $i = 1$ then
  $x_1 + (i - 1) \cdot r_1 = x_1$
if the message space has prime order then
  $i \cdot r_0$ has uniform distribution for $i \neq 0$ and $(i - 1) \cdot r_1$ is uniform if $i \neq 1$

This gives privacy for the sender
Aiello-Ishai-Reingold OT [AIR01]

- Discussion OT using additively homomorphic encryption
- The receiver encrypts Enc(i) and sends to the sender
- The sender chooses two random values $r_0$ and $r_1$
- The sender computes Enc($x_0 + i \cdot r_0$) and Enc($x_1 + (i - 1) \cdot r_1$) and sends back
- The receiver decrypts to get $x_0 + i \cdot r_0$ and $x_1 + (i - 1) \cdot r_1$
Aiello-Ishai-Reingold OT [AIR01]

- **Discussion** OT using additively homomorphic encryption
- The receiver encrypts $\text{Enc}(i)$ and sends to the sender
- The sender chooses two random values $r_0$ and $r_1$
- The sender computes $\text{Enc}(x_0 + i \cdot r_0)$ and $\text{Enc}(x_1 + (i - 1) \cdot r_1)$ and sends back
- The receiver decrypts to get $x_0 + i \cdot r_0$ and $x_1 + (i - 1) \cdot r_1$
  - if $i = 0$ then $x_0 + i \cdot r_0 = x_0$
  - if $i = 1$ then $x_1 + (i - 1) \cdot r_1 = x_1$

if the message space has prime order then $i \cdot r_0$ has uniform distribution for $i \neq 0$ and $(i - 1) \cdot r_1$ is uniform if $i \neq 1$ This gives privacy for the sender
Aiello-Ishai-Reingold OT [AIR01]

- Discussion OT using additively homomorphic encryption
- The receiver encrypts Enc($i$) and sends to the sender
- The sender chooses two random values $r_0$ and $r_1$
- The sender computes Enc($x_0 + i \cdot r_0$) and Enc($x_1 + (i - 1) \cdot r_1$) and sends back
- The receiver decrypts to get $x_0 + i \cdot r_0$ and $x_1 + (i - 1) \cdot r_1$
  - if $i = 0$ then $x_0 + i \cdot r_0 = x_0$
  - if $i = 1$ then $x_1 + (i - 1) \cdot r_1 = x_1$
  - if the message space has prime order then $i \cdot r_0$ has uniform distribution for $i \neq 0$ and $(i - 1) \cdot r_1$ is uniform if $i \neq 1$
  - This gives privacy for the sender
Two-Party Multiplication with Correlated OT

- First party knows $a$ and the second party knows $b = \sum 2^i B_i$
Two-Party Multiplication with Correlated OT

- First party knows $a$ and the second party knows $b = \sum 2^i B_i$
- The parties run correlated OT for each bit $B_i$:
  - First (sender) party always inputs $a$ (this is the correlation between messages)
  - Second (receiver) party inputs $B_i$
  - First party learns random $x_i$
  - The second party learns $y_i = x_i + B_i \cdot a$

OT is also used to do preprocessing for secret sharing based MPC
Two-Party Multiplication with Correlated OT

- First party knows $a$ and the second party knows $b = \sum 2^i B_i$
- The parties run correlated OT for each bit $B_i$:
  - First (sender) party always inputs $a$ (this is the correlation between messages)
  - Second (receiver) party inputs $B_i$
  - First party learns random $x_i$
  - The second party learns $y_i = x_i + B_i \cdot a$
- First party computes $x = -\sum 2^i x_i$
- Second party computes
  $$y = \sum 2^i y_i = \sum 2^i (x_i + B_i \cdot a) = -x + a \cdot \sum 2^i B_i = -x + a \cdot b$$
- $x + y = a \cdot b$
Two-Party Multiplication with Correlated OT

- First party knows $a$ and the second party knows $b = \sum 2^i B_i$
- The parties run correlated OT for each bit $B_i$:
  - First (sender) party always inputs $a$ (this is the correlation between messages)
  - Second (receiver) party inputs $B_i$
  - First party learns random $x_i$
  - The second party learns $y_i = x_i + B_i \cdot a$
- First party computes $x = -\sum 2^i x_i$
- Second party computes
  $y = \sum 2^i y_i = \sum 2^i (x_i + B_i \cdot a) = -x + a \cdot \sum 2^i B_i = -x + a \cdot b$
- $x + y = a \cdot b$
- OT is also used to do preprocessing for secret sharing based MPC
Private Set Intersection (PSI)

- PSI problem:
  - There are two parties having sets of elements $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ respectively.
  - They should learn which elements both of them have.
  - They should not learn other elements of the other party.

Discussion

- Any use-cases?
- Protocol ideas?

Public key encryption based protocols

April 8, 2020
Private Set Intersection (PSI)

- **PSI problem:**
  - There are two parties having sets of elements $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ respectively
  - They should learn which elements both of them have
  - They should not learn other elements of the other party

- **Discussion** Any use-cases?
Private Set Intersection (PSI)

- PSI problem:
  - There are two parties having sets of elements $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ respectively
  - They should learn which elements both of them have
  - They should not learn other elements of the other party

- Discussion Any use-cases?
- Discussion Protocol ideas?
**Private Set Intersection (PSI)**

- **PSI problem:**
  - There are two parties having sets of elements $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ respectively
  - They should learn which elements both of them have
  - They should not learn other elements of the other party

- **Discussion** Any use-cases?
- **Discussion** Protocol ideas?
  - Any generic two-party computation
  - Public key encryption based protocols
  - OT based protocols
PSI with Hashing

- Both parties hash their values as $H(x_1), \ldots, H(x_n)$ and $H(y_1), \ldots, H(y_m)$
- First party sends their hashes to the second party
- The second party checks if any received hashes are equal to the ones it computed
  - if $H(y_i) = H(x_j)$ then it outputs $y_i$
PSI with Hashing

- Both parties hash their values as $H(x_1), \ldots, H(x_n)$ and $H(y_1), \ldots, H(y_m)$
- First party sends their hashes to the second party
- The second party checks if any received hashes are equal to the ones it computed
  - if $H(y_i) = H(x_j)$ then it outputs $y_i$
- **Discussion** Which properties we need from the hash function?
PSI with Hashing

- Both parties hash their values as $H(x_1), \ldots, H(x_n)$ and $H(y_1), \ldots, H(y_m)$
- First party sends their hashes to the second party
- The second party checks if any received hashes are equal to the ones it computed
  - if $H(y_i) = H(x_j)$ then it outputs $y_i$

Discussion Which properties we need from the hash function?
- Communication efficient if hashes are shorter than actual data
- Need collision resistant hash function
PSI with Hashing

- Both parties hash their values as $H(x_1), \ldots, H(x_n)$ and $H(y_1), \ldots, H(y_m)$
- First party sends their hashes to the second party
- The second party checks if any received hashes are equal to the ones it computed
  - if $H(y_i) = H(x_j)$ then it outputs $y_i$
- **Discussion** Which properties we need from the hash function?
  - Communication efficient if hashes are shorter than actual data
  - Need collision resistant hash function
- **Discussion** Problems with this protocol?
PSI with Hashing

- Both parties hash their values as $H(x_1), \ldots, H(x_n)$ and $H(y_1), \ldots, H(y_m)$
- First party sends their hashes to the second party
- The second party checks if any received hashes are equal to the ones it computed
  - if $H(y_i) = H(x_j)$ then it outputs $y_i$

Discussion Which properties we need from the hash function?
- Communication efficient if hashes are shorter than actual data
- Need collision resistant hash function

Discussion Problems with this protocol?
- Can leak the first parties inputs (guessing or rainbow table attack)
PSI with Hashing

- Both parties hash their values as $H(x_1), \ldots, H(x_n)$ and $H(y_1), \ldots, H(y_m)$
- First party sends their hashes to the second party
- The second party checks if any received hashes are equal to the ones it computed
  - if $H(y_i) = H(x_j)$ then it outputs $y_i$
- **Discussion** Which properties we need from the hash function?
  - Communication efficient if hashes are shorter than actual data
  - Need collision resistant hash function
- **Discussion** Problems with this protocol?
  - Can leak the first parties inputs (guessing or rainbow table attack)
- This is what is probably used in practice
PSI based on Diffie-Hellman [Mea86]

- Both parties hash their values as $H(x_1), \ldots, H(x_n)$ and $H(y_1), \ldots, H(y_m)$
- Some hash function that transforms the items to cyclic group elements
PSI based on Diffie-Hellman [Mea86]

- Both parties hash their values as $H(x_1), \ldots, H(x_n)$ and $H(y_1), \ldots, H(y_m)$
  - Some hash function that transforms the items to cyclic group elements
- The parties choose random elements $a$ and $b$ respectively
- Compute $H(x_1)^a, \ldots, H(x_n)^a$ and $H(y_1)^b, \ldots, H(y_m)^b$
PSI based on Diffie-Hellman [Mea86]

- Both parties hash their values as $H(x_1), \ldots, H(x_n)$ and $H(y_1), \ldots, H(y_m)$
  - Some hash function that transforms the items to cyclic group elements
- The parties choose random elements $a$ and $b$ respectively
- Compute $H(x_1)^a, \ldots, H(x_n)^a$ and $H(y_1)^b, \ldots, H(y_m)^b$
- Parties exchange the computed values and compute
  - First party: $(H(y_1)^b)^a, \ldots, (H(y_m)^b)^a = (H(y_1)^{ab}, \ldots, H(y_m)^{ab})$
  - Second party: $(H(x_1)^a)^b, \ldots, (H(x_n)^a)^b = (H(x_1)^{ab}, \ldots, H(x_n)^{ab})$
PSI based on Diffie-Hellman [Mea86]

- Both parties hash their values as $H(x_1), \ldots, H(x_n)$ and $H(y_1), \ldots, H(y_m)$
  - Some hash function that transforms the items to cyclic group elements
- The parties choose random elements $a$ and $b$ respectively
- Compute $H(x_1)^a, \ldots, H(x_n)^a$ and $H(y_1)^b, \ldots, H(y_m)^b$
- Parties exchange the computed values and compute
  - First party: $(H(y_1)^b)^a, \ldots, (H(y_m)^b)^a = (H(y_1)^{ab}, \ldots, H(y_m)^{ab})$
  - Second party: $(H(x_1)^a)^b, \ldots, (H(x_n)^a)^b = (H(x_1)^{ab}, \ldots, H(x_n)^{ab})$
- Second party sends $(H(x_1)^{ab}, \ldots, H(x_n)^{ab})$ to the first party
- The first party compares $(H(x_1)^{ab}, \ldots, H(x_n)^{ab})$ and $(H(y_1)^{ab}, \ldots, H(y_m)^{ab})$ to find matches
- Randomizers $a$ and $b$ provide privacy for the parties
PSI based on OT

- First OT is used to build private equality check
PSI based on OT

- First OT is used to build private equality check
- When we have private equality then we can check if some element $x$ is in set $Y$
  - Private equality checks in parallel
  - a.k.a private set inclusion
PSI based on OT

- First OT is used to build private equality check
- When we have private equality then we can check if some element \(x\) is in set \(Y\)
  - Private equality checks in parallel
  - a.k.a private set inclusion
- Then we extend it to check a set of elements \(X\) and if they are in set \(Y\)
  - Parallel executions of private set inclusion (a lot of private equality checks)
  - More tricks are used to make it more efficient, see [PSZ18]
PSI based on OT

- First OT is used to build private equality check
- When we have private equality then we can check if some element $x$ is in set $Y$
  - Private equality checks in parallel
  - a.k.a private set inclusion
- Then we extend it to check a set of elements $X$ and if they are in set $Y$
  - Parallel executions of private set inclusion (a lot of private equality checks)
  - More tricks are used to make it more efficient, see [PSZ18]
- Parallelism can be done more efficiently by aligning and reusing part of the protocols
  - and by OT extension protocols
Private Equality Test from OT

- Inputs are two \( n \)-bit strings \( x_1 \ldots x_n \) and \( y_1 \ldots y_n \)
- Run OT with random sender messages (\( n \) times)
  - Sender learns two (sufficiently long) random values \( r_{i,0}, r_{i,1} \) for \( i \)'th execution
  - Receiver inputs \( x_i \) and learns \( r_{i,x_i} \) for each execution

Receiver compares \( r_s \) and \( r_r \)
If they are equal then so were the initial values
This is correct if the random values are long enough
Privacy is ensured because one unknown value \( r_{i,y_i} \) completely randomizes \( r_s \)
Private Equality Test from OT

- Inputs are two $n$-bit strings $x_1 \ldots x_n$ and $y_1 \ldots y_n$
- Run OT with random sender messages ($n$ times)
  - Sender learns two (sufficiently long) random values $r_{i,0}, r_{i,1}$ for $i$’th execution
  - Receiver inputs $x_i$ and learns $r_{i,x_i}$ for each execution
- Sender computes $r_s = r_{1,y_1} \oplus \ldots \oplus r_{n,y_n}$ and sends to the receiver
- Receiver computes $r_r = r_{1,x_1} \oplus \ldots \oplus r_{n,x_n}$
- Receiver compares $r_s$ and $r_r$
  - If they are equal then so were the initial values
  - This is correct if the random values are long enough
  - Privacy is ensured because one unknown value $r_{i,y_i}$ completely randomizes $r_s$
OT from PSI

- Discussion Ideas? One bit messages?

- Receiver has input $b$, for PSI encode it as set $(b_0, b_1)$
- Sender has one bit inputs $x_0, x_1$ encoded as 0 $x_0$ and 1 $x_1$
- If $b = 1$ then we do the intersection $(10, 11)$ with $(0x_1, 1x_1)$ and get 1 $x_1$ as the intersection
- If $b = 0$ then we similarly get 0 $x_0$
OT from PSI

- Discussion Ideas? One bit messages?
- Need a version of PSI where only one party gets the outputs
OT from PSI

- *Discussion* Ideas? One bit messages?
- Need a version of PSI where only one party gets the outputs
- Receiver has input $b$, for PSI encode it as set $(b_0, b_1)$
- Sender has one bit inputs $x_0, x_1$ encoded as $0x_0$ and $1x_1$
OT from PSI

- Discussion Ideas? One bit messages?
- Need a version of PSI where only one party gets the outputs
- Receiver has input $b$, for PSI encode it as set $(b0, b1)$
- Sender has one bit inputs $x_0, x_1$ encoded as $0x_0$ and $1x_1$
- If $b = 1$ then we do the intersection $(10, 11)$ with $(0x_1, 1x_1)$ and get $1x_1$ as the intersection
- If $b = 0$ then we similarly get $0x_0$
Connections Between the Topics

- Non-trivial single server PIR implies OT [DCMO00]
- Communication efficient 1-out-of-$n$ OT is sometimes also called symmetric PIR (SPIR)
Connections Between the Topics

- Non-trivial single server PIR implies OT [DCMO00]
  - Communication efficient 1-out-of-\(n\) OT is sometimes also called symmetric PIR (SPIR)
- PSI implies OT
  - We had the construction
Connections Between the Topics

- Non-trivial single server PIR implies OT [DCMO00]
  - Communication efficient 1-out-of-\(n\) OT is sometimes also called symmetric PIR (SPIR)
- PSI implies OT
  - We had the construction
- PSI and non-trivial single server PIR require public key primitives
  - Because OT does [Ger+00]
Connections Between the Topics

- Non-trivial single server PIR implies OT [DCMO00]
  - Communication efficient 1-out-of-n OT is sometimes also called symmetric PIR (SPIR)
- PSI implies OT
  - We had the construction
- PSI and non-trivial single server PIR require public key primitives
  - Because OT does [Ger+00]
- OT is complete for secure multiparty computation [Kil88]
  - Can use OT to construct any other protocol (including PSI and PIR)
  - We have seen the multiplication and equality test as examples
Oblivious RAM (ORAM)

- ORAM problem
  - Memory accesses (reads and writes) leak information about the data or the execution/program
  - This also applies when the memory itself is encrypted
- Discussion Where might it be a problem?
Oblivious RAM (ORAM)

- ORAM problem
  - Memory accesses (reads and writes) leak information about the data or the execution/program
  - This also applies when the memory itself is encrypted
- Discussion Where might it be a problem?
- Discussion Solution?
Oblivious RAM (ORAM)

- ORAM problem
  - Memory accesses (reads and writes) leak information about the data or the execution/program
  - This also applies when the memory itself is encrypted
- Discussion Where might it be a problem?
- Discussion Solution?
  - Always read and write the whole memory
  - For reading PIR or OT are sufficient
Oblivious RAM (ORAM)

- ORAM problem
  - Memory accesses (reads and writes) leak information about the data or the execution/program
  - This also applies when the memory itself is encrypted

- Discussion Where might it be a problem?
- Discussion Solution?
  - Always read and write the whole memory
  - For reading PIR or OT are sufficient
  - More efficient solutions focus on the memory layout
    - Mostly tree based solutions
Requirements for ORAM

- Client needs randomness
  - e.g. randomize the location where the value is written back
- Data needs to be encrypted
- Each time you read data you must also write it (to hide the type of access)
  - Re-encrypting the value to hide if we changed it or not
- Accessing the same item many times should go to different physical locations
Tree based ORAM Idea

- Arrange memory as a binary tree

Each item is stored in the path to this leaf. Each node in the tree can contain $\sqrt{n}$ items. Reading: Look for the node in the path, remove when found. Writing/Updating: Add to the root, update the position map with a new leaf. Preventing overflow in the root: Randomly move objects towards the leaf (from two nodes in each level). Writing to both children of the chosen node.
Tree based ORAM Idea

- Arrange memory as a binary tree
  - The Client stores a leaf marker for each item
  - Each item is stored in the path to this leaf
  - Each node in the tree can contain $\sqrt{n}$ items
Tree based ORAM Idea

- Arrange memory as a binary tree
  - The Client stores a leaf marker for each item
  - Each item is stored in the path to this leaf
  - Each node in the tree can contain $\sqrt{n}$ items
- Reading: Look for the node in the path
  - Remove when found

Diagram:

- Root
- Node 1
- Node 2
- Node 3
- Node 4
- Node 5
- Node 6
- Node 7
- Node 8

April 8, 2020
Tree based ORAM Idea

- Arrange memory as a binary tree
  - The Client stores a leaf marker for each item
  - Each item is stored in the path to this leaf
  - Each node in the tree can contain $\sqrt{n}$ items
- Reading: Look for the node in the path
  - Remove when found
- Writing/Updating: Add to the root
  - Update the position map with a new leaf
Tree based ORAM Idea

- Arrange memory as a binary tree
  - The Client stores a leaf marker for each item
  - Each item is stored in the path to this leaf
  - Each node in the tree can contain $\sqrt{n}$ items
- Reading: Look for the node in the path
  - Remove when found
- Writing/Updating: Add to the root
  - Update the position map with a new leaf
- Preventing overflow in the root
  - Randomly move objects towards the leaf (from two nodes in each level)
  - Writing to both children of the chosen node
References and Extra Materials I


References and Extra Materials II


References and Extra Materials III


References and Extra Materials IV

[Rom17] Yolan Romailler. *YAO’S GARBLED CIRCUITS AND HOW TO CONSTRUCT THOSE*. 2017. URL:
https://romaille.ch/2017/06/09/garbling_circuits/.

