Computing on Encrypted Data

N.P. Smart

COSIC,
KU Leuven, ESAT,
Kasteelpark Arenberg 10, bus 2452,
B-3001 Leuven-Heverlee,
Belgium.

August 31, 2018

Outline

What Questions Will We Answer In These Lectures?

How can we compute without revealing data?

What is Multi-Party Computation?

What is Fully Homomorphic Encryption?

What is Yao’s Garbled Circuit method for two parties?

What are Linear Secret Sharing Schemes and how are they used?

Multi-Party Computation

Suppose I have n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$

Each party has some input x_i

They want to compute a function

$$F(x_1, \ldots, x_n)$$

without revealing the input values

- Bar what can be learned from the output.

We let $A \subset \mathcal{P}$ denote the set of bad people

- We do not know what this set is

Examples

Danish Sugar Beet Auction: Partisia

Sharemind Database: Cybernetica

Jana Database: Galois/KU Leuven

vHSM: Unbound Tech

KMaaS: Sepior

Multi-Party Computation

We can have various properties:

Correctness: If the parties in A do not deviate from the protocol then the parties get the correct output.

Passive Security: If the parties in A do not deviate from the protocol then they can learn nothing about the parties input.

Robust Active Security: If the parties in A deviate from the protocol then the honest parties will always still determine the correct output.

Active Security with Abort: If the parties in A deviate from the protocol then the honest parties will abort the computation.

Multi-Party Computation

The set of all possible sets A is called the Adversary structure $\mathcal{A}$.

If for all $A \in \mathcal{A}$ we have $|A| \leq t$ then we are said to have a threshold structure.

- We will only look at threshold adversaries in this talk.
- We will also assume synchronous networks

There are other properties one could have:

Static Security: The set A is determined at the start of the protocol.

Adaptive Security: The set A is determined by the adversary during the protocol.

Adaptive security is hard to obtain, so mainly protocols focus on static security.

Multi-Party Computation

Information Theoretically Secure MPC:

- Passive security can be achieved if and only if $t < n/2$
- Robust active security can be achieved if and only if $t < n/3$ (Byzantine generals)
- Active security with abort can be achieved if and only if $t < n/2$

Computationally Secure MPC:

- Passive security can be achieved if $t < n$
- Robust active security can be achieved if and only if $t < n/2$
- Active security with abort can be achieved if $t < n$

Fully Homomorphic Encryption

Consider a public key encryption scheme defined by three algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with message space a finite field $\mathbb{F}_p$

$$\mathsf{Dec}(\mathsf{Enc}(m, \mathsf{pk}; r), \mathsf{sk}) = m.$$

Now suppose we have two additional algorithms which satisfy

$$\mathsf{Dec}(\mathsf{Add}(\mathsf{Enc}(m_1, \mathsf{pk}; r_1), \mathsf{Enc}(m_2, \mathsf{pk}; r_2)), \mathsf{sk}) = m_1 + m_2,$$
$$\mathsf{Dec}(\mathsf{Mult}(\mathsf{Enc}(m_1, \mathsf{pk}; r_1), \mathsf{Enc}(m_2, \mathsf{pk}; r_2)), \mathsf{sk}) = m_1 \cdot m_2$$

Then we are said to have a Fully Homomorphic Encryption scheme.

Note, an FHE scheme cannot be IND-CCA (it is homomorphic), at most can hope for IND-CPA

Fully Homomorphic Encryption

Text-book RSA is an OW-CPA secure multiplicatively homomorphic scheme

Text-book Paillier is an IND-CPA secure additively homomorphic scheme

The trick is to come up with a scheme which is both additively and multiplicatively homomorphic.

Once we have addition and multiplication we can evaluate *any* function

- Since $+$ and $\times$ are universal over $\mathbb{F}_p$.

Fully Homomorphic Encryption

We can define a procedure Eval which takes

- Any function $f(x_1, \ldots, x_n)$
- A set of ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ encrypting $m_1, \ldots, m_n$

and satisfies

$$\mathsf{Dec}(\mathsf{Eval}(f, \{\mathsf{ct}_1, \ldots, \mathsf{ct}_n\}), \mathsf{sk}) = f(m_1, \ldots, m_n).$$

We can then *out-source* any computation to a server.

- Server gets the ciphertexts.
- Server evaluates the function.
- Server returns the encrypted result to the user for decryption.

Somewhat Homomorphic Encryption

In practice SHE schemes are easier to construct

These are schemes for which the function Eval has restrictions

- It cannot be applied recursively
- The input function f comes from a set of admissable functions $\mathcal{A}$.

In practice we take $\mathcal{A}$ to be the set of functions with low-multiplicative depth

- This measure of complexity of a function comes up again and again in this area.

SHE+FHE

If the set $\mathcal{A}$ is big enough then we can transform an SHE scheme into an FHE one.

Suppose $\mathcal{A}$ contains all functions of the form

$$f_{\mathsf{ct}_1,\mathsf{ct}_2}(\mathsf{sk}) = \mathsf{Dec}(\mathsf{ct}_1,\mathsf{sk}) + \mathsf{Dec}(\mathsf{ct}_2,\mathsf{sk}),$$
$$g_{\mathsf{ct}_1,\mathsf{ct}_2}(\mathsf{sk}) = \mathsf{Dec}(\mathsf{ct}_1,\mathsf{sk}) \cdot \mathsf{Dec}(\mathsf{ct}_2,\mathsf{sk}).$$

for *any* valid ciphertexts ct_1 and ct_2.
Suppose we also obtain an encryption of the secret key

$$\mathsf{ct}_{\mathsf{sk}} = \mathsf{Enc}(\mathsf{sk},\mathsf{pk}; r)$$

SHE+FHE

Then given *fresh* ciphertexts (newly encrypted) ct_1 and ct_2 encryting m_1 and m_2 we can compute

$$\begin{aligned}\mathsf{ct}_a &= \mathsf{Eval}(f_{\mathsf{ct}_1,\mathsf{ct}_2}, \mathsf{ct}_{\mathsf{sk}}),\\ \mathsf{ct}_m &= \mathsf{Eval}(g_{\mathsf{ct}_1,\mathsf{ct}_2}, \mathsf{ct}_{\mathsf{sk}})\end{aligned}$$

- ct_a will be an encryption of $m_1 + m_2$.
- ct_m will be an encryption of $m_1 \cdot m_2$.

We *can* apply this trick recursively

- Only ever apply Eval to encryptions of the secret key
- The output ciphertexts define different functions from $\mathcal{A}$.

Thus if $\mathcal{A}$ is big enough we have an FHE scheme.

FHE+MPC

Now suppose we have an ability to perform a distributed decryption in our FHE scheme

For n parties we split sk into n keys and have a new decryption algorithm which runs in two steps

- $\mathrm{pm}_i = \mathrm{PartialDec}(\mathrm{ct}, \mathrm{sk}_i)$.
- $m = \mathrm{Combine}(\mathrm{pm}_1, \ldots, \mathrm{pm}_n)$.

such that m is the same value as executing $\mathrm{Dec}(\mathrm{ct}, \mathrm{sk})$.

FHE+MPC

We can then define a passive computationally secure two round MPC protocol...

- Party P_i encrypts its input $\mathsf{ct}_i = \mathsf{Enc}(m_i, \mathsf{pk}; r_i)$.
- The parties exchange the ciphertexts ct_i.
- All parties compute $\mathsf{ct} = \mathsf{Eval}(f, \{\mathsf{ct}_1, \ldots, \mathsf{ct}_n\})$.
- Party P_i computes $\mathsf{pm}_i = \mathsf{PartialDec}(\mathsf{ct}, \mathsf{sk}_i)$.
- The parties exchange the partial decryptions pm_i.
- The parties compute $f(m_1, \ldots, m_n) = \mathsf{Combine}(\mathsf{pm}_1, \ldots, \mathsf{pm}_n)$.

This would be a very slow protocol as existing FHE schemes are very slow indeed.

Summary of What's Coming Up

The FHE based approach to MPC gives us low round complexity, but

- It is very slow
- It is only passively secure

We would like a more efficient solution

We will look at two approaches to MPC

- Yao's Garbled Circuit approach
- Linear Secret Sharing Schemes based approach

The first is best suited to two parties, whereas the second works for any number of parties.

Yao's Garbled Circuits

We first consider the case of two party passively secure computation

We assume two parties who want to compute a function $y = f(x_1, x_2)$

- Party P_1 holds x_1
- Party P_2 holds x_2

Both parties want to learn y

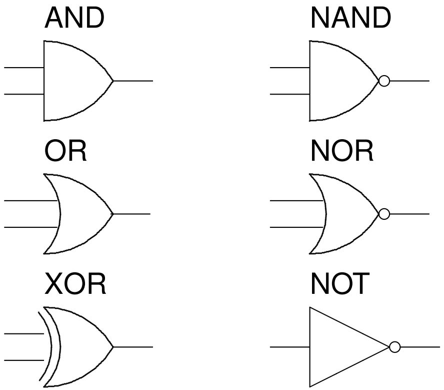
Party P_1 does not want P_2 to learn x_1, and vice versa.

The oldest and simplest way of achieving this is via Yao's Garbled Circuits

- Which are surprisingly fast these days

Garbled Circuits: Simple Version

We take the function *f* are write it as a boolean circuit



Our aim is to “encrypt” each gate.

Wire Values

Each wire w_i in the circuit can have two values on it 0 or 1

We assign two (symmetric) keys k_i^0 and k_i^1 to each wire value on each wire.

Every gate G can be represented by a function with two input wires and one output wire

$$w_k = G(w_i, w_j)$$

Note: “NOT” gates can be “folded” into the following output gate.

AND Gate Encryption

We go through an example of how to encrypt an AND gate

w_i	w_j	w_k
0	0	0
0	1	0
1	0	0
1	1	1

AND Gate Encryption

When someone evaluates the gate we want them to learn the wire key

w_i	w_j	w_k	m
0	0	0	k_k^0
0	1	0	k_k^0
1	0	0	k_k^0
1	1	1	k_k^1

AND Gate Encryption

Now we encrypt this message with the wire keys associated to w_i and w_j.

- We assume an IND-CCA two key symmetric encryption function $E_{k,k'}(m)$.

w_i	w_j	w_k	c
0	0	0	$E_{k_i^0,k_j^0}(k_k^0)$
0	1	0	$E_{k_i^0,k_j^1}(k_k^0)$
1	0	0	$E_{k_i^1,k_j^0}(k_k^0)$
1	1	1	$E_{k_i^1,k_j^1}(k_k^1)$

AND Gate Encryption

We now create a random permutation of the table

w_i	w_j	w_k	c
1	1	1	$E_{k_i^1,k_j^1}(k_k^1)$
0	1	0	$E_{k_i^0,k_j^1}(k_k^0)$
0	0	0	$E_{k_i^0,k_j^0}(k_k^0)$
1	0	0	$E_{k_i^1,k_j^0}(k_k^0)$

AND Gate Encryption

We then just keep the ciphertext columns

- This table is called a Garbled Gate.

c
$E_{k_i^1,k_j^1}(k_k^1)$
$E_{k_i^0,k_j^1}(k_k^0)$
$E_{k_i^0,k_j^0}(k_k^0)$
$E_{k_i^1,k_j^0}(k_k^0)$

So each gate in the circuit has four ciphertexts associated to it.

Gate Evaluation

Gate evaluation occurs as follows:

Suppose the player learns the wire label value for the zero value on wire *i* and the one value on wire *j*.

- They learn k_i^0 and k_j^1.
- Note they do not know wire *i* is zero and wire *j* is one.

Using these values they can decrypt only one row of the table

- They try all rows, but only one actually decrypts
- This is why we needed an IND-CCA scheme, as it rejects invalid ciphertexts.

Gate Evaluation

c
$E_{k_i^1,k_j^1}(k_k^1)$
$E_{k_i^0,k_j^1}(k_k^0)$
$E_{k_i^0,k_j^0}(k_k^0)$
$E_{k_i^1,k_j^0}(k_k^0)$

We can only decrypt the second row.

Hence, we learn k_k^0, but we have no idea it corresponds to the zero value on the output wire.

Garbled Circuit

Given a function

$$y = F(x)$$

expressed as a boolean circuit for F the entire garbled circuit is the following values

- The garbled table for every gate in F.
- The “wire label table” for every possible input bit
- The “wire label table” for every possible output bit

Suppose the input wires are wire numbers $0, \ldots, t$.

The input wire label table is then the values

$$(i, k_i^0, k_i^1).$$

Same for the output wire label table.

Garbled Circuits: Complex Version

The above construction was very wasteful in resources:

Needed an IND-CCA symmetric encryption scheme

Evaluator had to decrypt four ciphertexts, when he was only interested in one.

Would like something simpler, and faster...

Wire Values

As before each wire w_i in the circuit can have two values on it 0 or 1

As before we assign two (symmetric) keys k_i^0 and k_i^1 to each wire value on each wire.

But Now: We pick a random bit $\rho_i \in \{0,1\}$ for each wire.

If when the circuit is evaluated the actual values if $v_i \in \{0,1\}$, then

- $e_i = \rho_i \oplus v_i$ is called the *external value*
- $k_i^{v_i}$ is called the *wire key*.

Every gate G can be represented by a function with two input wires and one output wire

$$w_k = G(w_i, w_j)$$

Note: "NOT" gates can be "folded" into the following output gate.

AND Gate Encryption

We go through an example of how to encrypt an AND gate

w_i	w_j	w_k
0	0	0
0	1	0
1	0	0
1	1	1

AND Gate Encryption

We first add in the random values ρ_i, ρ_j, ρ_k...

- We pick $\rho_i = 1$, $\rho_j = 0$ and $\rho_k = 1$.

w_i	w_j	w_k	e_i	e_j	e_k
0	0	0	1	0	1
0	1	0	1	1	1
1	0	0	0	0	1
1	1	1	0	1	0

AND Gate Encryption

When someone evaluates the gate we want them to learn the wire key and the external value.

- So we encode this as a message to be encrypted

w_i	w_j	w_k	e_i	e_j	e_k	m
0	0	0	1	0	1	$k_k^0 \| 1$
0	1	0	1	1	1	$k_k^0 \| 1$
1	0	0	0	0	1	$k_k^0 \| 1$
1	1	1	0	1	0	$k_k^1 \| 0$

AND Gate Encryption

Now we encrypt this message with the wire keys associated to w_i and w_j.

- We assume an IND-CPA two key symmetric encryption function $E_{k,k'}(m)$.

w_i	w_j	w_k	e_i	e_j	e_k	c
0	0	0	1	0	1	$E_{k_i^0,k_j^0}(k_k^0 \| 1)$
0	1	0	1	1	1	$E_{k_i^0,k_j^1}(k_k^0 \| 1)$
1	0	0	0	0	1	$E_{k_i^1,k_j^0}(k_k^0 \| 1)$
1	1	1	0	1	0	$E_{k_i^1,k_j^1}(k_k^1 \| 0)$

AND Gate Encryption

We now re-order the table according to a standard ordering of e_i and e_j.

w_i	w_j	w_k	e_i	e_j	e_k	c
1	0	0	0	0	1	$E_{k_i^1, k_j^0}(k_k^0 \| 1)$
1	1	1	0	1	0	$E_{k_i^1, k_j^1}(k_k^1 \| 0)$
0	0	0	1	0	1	$E_{k_i^0, k_j^0}(k_k^0 \| 1)$
0	1	0	1	1	1	$E_{k_i^0, k_j^1}(k_k^0 \| 1)$

AND Gate Encryption

We then just keep the e_i, e_j and ciphertext columns

- This table is called a Garbled Gate.

e_i	e_j	c
0	0	$E_{k_i^1,k_j^0}(k_k^0\|1)$
0	1	$E_{k_i^1,k_j^1}(k_k^1\|0)$
1	0	$E_{k_i^0,k_j^0}(k_k^0\|1)$
1	1	$E_{k_i^0,k_j^1}(k_k^0\|1)$

We label the ciphertexts $c_{a,b}^k$ where $a = e_i$, $b = e_j$ and k is the output wire number.

So each gate in the circuit has four ciphertexts associated to it.

Gate Evaluation

Gate evaluation occurs as follows:

Suppose the player learns the wire label/external value for the zero value on wire *i* and the one value on wire *j*.

- They learn that $e_i = 1$ (as recall $\rho_i = 1$)
- They learn that $e_j = 1$ (as recall $\rho_j = 0$)
- They learn k_i^0 and k_j^1.
- Note they do not know wire *i* is zero and wire *j* is one.

Using these values they can decrypt only one row of the table

- And they know which one it is, since the external values tell them

Gate Evaluation

e_i	e_j	c
0	0	$E_{k_i^1,k_j^0}(k_k^0 \| 1)$
0	1	$E_{k_i^1,k_j^1}(k_k^1 \| 0)$
1	0	$E_{k_i^0,k_j^0}(k_k^0 \| 1)$
1	1	$E_{k_i^0,k_j^1}(k_k^0 \| 1)$

Note, only the last row can be decrypted.

The evaluating party knows that wire w_k has external value $e_k = 1$ and wire key k_k^0.

But they have no idea what these values really represent.

Garbled Circuit

Given a function

$$y = F(x)$$

expressed as a boolean circuit for F the entire garbled circuit is the following values

- The garbled table for every gate in F.
- The “wire label table” for every possible input bit
- The “wire label table” for every possible output bit

Suppose the input wires are wire numbers $0, \ldots, t$.

The input wire label table is then the values

$$(i, k_i^0, k_i^1, e_i^0, e_i^1)$$

where $e_i^b = \rho_i \oplus b$.

Same for the output wire label table.

Encryption Scheme

Only question remaining here is what encryption function to use...

Take a block cipher B_k such as AES encrypting 128-bit blocks.

Take k_i^0 and k_i^1 to be 127-bit values

We want to encrypt the 128 bit value $k_k^0 \| e_k$ using two keys k_i^u, k_j^v.

Use, for a fixed key k,

$$\begin{aligned} E_{k_i^u, k_j^v}(k_k^0 \| e_k) &= B_k(k_i^u \oplus k_j^v \oplus G) \oplus (k_i^u \oplus k_j^v \oplus G) \oplus (k_k^0 \| e_k), \\ &= H_{k,G}(k_i^u \oplus k_j^v) \oplus (k_k^0 \| e_k). \end{aligned}$$

where G is some gate identifier

- Keeping k fixed ensures no need to do expensive rekey.
- Gate identifier enables re-use of wires, saving costs.

Garbled Circuits: Complex Version - II

We want to cut down the size of a Garbled Circuit.

Turns out XOR gates can be done for free...

The constructor fixes a global secret value Δ

- $|\Delta| = |k_i^0| = |k_i^1|$.

We then set, for all wire labels,

$$k_i^1 = k_i^0 \oplus \Delta.$$

This allows XOR gates to come for “free”...

Garbled Circuits: Complex Version - II

If

- a and b are the actual inputs to a XOR gate, with output $c = a \oplus b$.
- Input wire numbers i and j, output wire number k

Then the associated data obtained by the evaluator is

$$(k_i^0 \oplus a \cdot \Delta) \| a \oplus \rho_i \quad \text{and} \quad (k_j^0 \oplus b \cdot \Delta) \| b \oplus \rho_j$$

From which the evaluator can compute the new output data as

$$\begin{aligned}(k_i^0 \oplus a \cdot \Delta \| a \oplus \rho_i) \oplus (k_j^0 \oplus b \cdot \Delta \| b \oplus \rho_j) \\ &= ((k_i^0 \oplus k_j^0) \oplus (a \oplus b) \cdot \Delta \| (a \oplus b) \oplus (\rho_i \oplus \rho_j)) \\ &= (k_k^0 \oplus c \cdot \Delta \| c \oplus \rho_k)\end{aligned}$$

where we set $k_k^0 = k_i^0 \oplus k_j^0$ and $\rho_k = \rho_i \oplus \rho_j$.

Garbled Circuits: Complex Version - II

We can simplify notation a little bit...

Make k_i^b one bit longer, and Δ one bit longer.

Let the last bit of Δ be one.

Then

- ρ_i is the last bit of k_i^0.
- e_i is the last bit of $k_i^0 \oplus \Delta$ (which the evaluator receives)

This allows us to explain our next optimization more clearly...

Half-Gate Evaluation of AND

We can now half the size of the AND gates...

Garbler knows for an AND gate

- k_a^0, k_b^0, ρ_a, ρ_b and Δ.

Evaluator knows for an AND gate

- k_a^a, k_b^b, e_a and e_b.

We give a gate which only requires two ciphertexts

Recall from earlier $H_{k,G}(X)$ is an encryption under a fixed key k with a gate label G.

- To simplify notation we will write $H(X) = H_{k,G}(X)$ with the key and gate label being implicit.
- We use $H(X)$ in the following optimization in a different way than before

Half-Gate Evaluation of AND: Garbler

Define the output zero wire as...

$$k_c^0 = H(k_a^0 \oplus \rho_a \cdot \Delta) \oplus H(k_b^0 \oplus \rho_b \cdot \Delta) \oplus \rho_a \cdot \rho_b \cdot \Delta$$

The garbled AND gate is then the two values

$$\begin{aligned} E_1 &= H(k_b^0) \oplus H(k_b^1) \oplus \rho_a \cdot \Delta, \\ E_2 &= H(k_a^0) \oplus H(k_a^1) \oplus k_b^0. \end{aligned}$$

Recall the garbler knows k_a^0, k_b^0, ρ_a, ρ_b and Δ and

$$\begin{aligned} k_a^1 &= k_a^0 \oplus \Delta, \\ k_b^1 &= k_b^0 \oplus \Delta. \end{aligned}$$

Half-Gate Evaluation of AND: Evaluator

The evaluator now knows k_a^a, k_b^b, e_a, e_b, E_1 and E_2.

They then compute

$$H(k_a^a) \oplus e_a \cdot (E_2 \oplus k_b^b) \oplus H(k_b^b) \oplus e_b \cdot E_1$$

When $e_a = 0$ and $e_b = 0$ this is equal to

$$\begin{aligned} H(k_a^a) \oplus H(k_b^b) &= H(k_a^0 \oplus a \cdot \Delta) \oplus H(k_b^0 \oplus b \cdot \Delta), \\ &= H(k_a^0 \oplus \rho_a \cdot \Delta) \oplus H(k_b^0 \oplus \rho_b \cdot \Delta), \\ &= k_c^0 \oplus \rho_a \cdot \rho_b \cdot \Delta, \\ &= k_c^0 \oplus a \cdot b \cdot \Delta, \end{aligned}$$

since $a \oplus \rho_a = e_a = 0$ and $b \oplus \rho_b = e_b = 0$.

Which is exactly what an AND gate should compute

Half-Gate Evaluation of AND: Evaluator

As another example take $e_a = 0$ and $e_b = 1$.

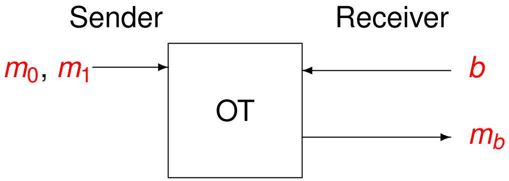
$$\begin{aligned}
& H(k_a^a) \oplus H(k_b^b) \oplus E_1 \\
& \quad = H(k_a^0 \oplus a \cdot \Delta) \oplus H(k_b^b) \oplus H(k_b^0) \oplus H(k_b^1) \oplus \rho_a \cdot \Delta, \\
& \quad = H(k_a^0 \oplus \rho_a \cdot \Delta) \oplus H(k_b^{1 \oplus b}) \oplus a \cdot \Delta \\
& \quad = H(k_a^0 \oplus \rho_a \cdot \Delta) \oplus H(k_0 \oplus (1 \oplus b) \cdot \Delta) \oplus a \cdot \Delta \\
& \quad = H(k_a^0 \oplus \rho_a \cdot \Delta) \oplus H(k_0 \oplus \rho_b \cdot \Delta) \oplus a \cdot \Delta \\
& \quad = k_c^0 \oplus \rho_a \cdot \rho_b \cdot \Delta \oplus a \cdot \Delta \\
& \quad = k_c^0 \oplus a \cdot (1 \oplus b) \cdot \Delta \oplus a \cdot \Delta \\
& \quad = k_c^0 \oplus a \cdot b \cdot \Delta.
\end{aligned}$$

since $a \oplus \rho_a = e_a = 0$ and $b \oplus \rho_b = e_b = 1$.

Checking the other two possibilities for e_a and e_b results in the same equation

Oblivious Transfer

Before giving Yao's MPC protocol we need another cryptographic tool



Simple OT Protocol

A simple passively secure OT protocol can be given by

$$\begin{array}{rcl}
\underline{\text{Sender}} & & \underline{\text{Receiver}} \\
C \leftarrow E(\mathbb{F}_p) & \xrightarrow{C} & x \leftarrow (\mathbb{Z}/q\mathbb{Z}) \\
 & & Q_b \leftarrow [x] \cdot P \\
 & \xleftarrow{Q_0} & Q_{1-b} \leftarrow C - Q_b \\
Q_1 \leftarrow C - Q_0 & & \\
k \leftarrow (\mathbb{Z}/q\mathbb{Z}) & & \\
C_1 \leftarrow [k] \cdot P & & \\
E_0 \leftarrow M_0 + [k] \cdot Q_0 & & \\
E_1 \leftarrow M_1 + [k] \cdot Q_1 & & \\
 & \xrightarrow{C_1, E_0, E_1} & \\
 & & M_b \leftarrow E_b - [x] \cdot C_1
\end{array}$$

Simple OT Protocol

Above protocol is basically two lots of ElGamal encryption

- Sender has choice of two public keys Q_0 and Q_1 to use.
- Sender does not know which key is “real”
- Receiver does not the private key of the Q_{1-b} public key

Use M_0 as the key to one message, and M_1 as the key to another.

The actual message can then be sent using symmetric crypto if needs be.

It is only passively secure.

- But what does secure mean?

Security of OT

An OT protocol is secure if:

Sender Security: The receiver cannot learn anything about M_{1-b}

- In above protocol this holds due to DDH hardness
- A receiver who breaks this can be turned into an adversary to solve DDH.

Receiver Security: The sender cannot learn anything about the bit b

- The above protocol is perfectly secure in this respect.
- There is a perfect simulation of the receiver, where there is no hidden bit b.

Yao's Two Party Protocol

So we now have the building blocks for Yao's two party protocol.

We first assume that the function is of the form

$$(y_1, y_2) = F(x_1, x_2)$$

where

- x_1 (resp. y_1) is party one's input (resp. output)
- x_2 (resp. y_2) is party two's input (resp. output)

We now give a passively secure protocol (so having only a passively secure OT is OK).

Yao's Two Party Protocol

Step 1:

Party one (the circuit garbler) creates a garbled circuit for F

$$(G, (I_1, I_2), (O_1, O_2))$$

where

- G is the set of garbled gates
- I_1 is the input wire label table for party one.
- I_2 is the input wire label table for party two
- O_1 is the output wire label table for party one.
- O_2 is the output wire label table for party two.

Yao's Two Party Protocol

Step 2:
The circuit garbler sends G to party two.

The circuit garbler also sends the values in I_1 corresponding to its input to the function

- So if the garbler wants to input bit b on wire w then it sends to party one the value (k_w^b, e_w^b).
- This reveals nothing about the actual input as k_w^b is a random key, and ρ_w remains hidden.

The circuit garbler also sends the table O_2 over to party two.

Yao's Two Party Protocol

Step 3:
The players now execute at OT protocol.

- One for each input wire w for player two.

Player two acts as the receiver with input bit the input he wants for the function.

Player one acts as the sender with the two "messages"

$$m_0 = (k_w^0, e_w^0) \quad \text{and} \quad m_1 = (k_w^1, e_w^1).$$

So if player two had input bit 0 he would learn (k_w^0, e_w^0) but not (k_w^1, e_w^1).

Yao's Two Party Protocol

Step 4:
The receiver (the circuit evaluator) can now evaluate the garbled circuit to get the garbled output wire labels.

Using O_2 the receiver can now decode his output to the value y_2.

The receiver then sends the rest of the output wire labels back to Player one.

Step 5:
Player one can decode his output value y_1 using this data and the table O_1.

String Comparison

Sometimes we can use OT to do simple function evaluation without using Garbled Circuits at all

Suppose P_1 has a bit string $x_1, \ldots, x_n$.

Suppose P_2 has a bit string $y_1, \ldots, y_n$.

They want to test if the two strings are equal

We present a protocol that gives P_2 this information

- For P_1 to get the info just run it in reverse

String Equality Check

The protocol goes as follows...

- Party P_1 generates random 128-bit inputs strings (r_i^0, r_i^1) for $i = 1, \ldots, n$.
- The parties run n OT protocols
  - Party P_1's input to the i-th OT is (r_i^0, r_i^1).
  - Party P_2's input to the i-th OT is y_i.
  - Party P_2 learns $r_i^{y_i}$.
- Party P_1 computes $X = \bigoplus_{i=1}^n r_i^{x_i}$.
- Party P_2 computes $Y = \bigoplus_{i=1}^n r_i^{y_i}$.
- Party P_1 sends P_2 the value X.
- If $X = Y$ then P_2 accepts.

Why is Yao not actively secure?

In basic Yao we have a circuit creator and a circuit evaluator.

The evaluator really cannot cheat at all

- They just get input and then do stuff
- This assumes an actively secure OT protocol is used
    - The earlier OT protocol is not actively secure.
    - But for our purposes lets just assume this can be fixed.

Why is Yao not actively secure?

The circuit creator P_1 could however create a bogus circuit

Instead of creating a circuit for the function

$$(y_1, y_2) = f(x_1, x_2)$$

then could create a circuit for the function

$$(x_2, y_2) = f(x_1, x_2)$$

and hence learn P_2's input.

An obvious solution is to remove this inbalance in the players

Dual Execution

So now let *both* parties act as circuit creator and circuit evaluator.

Instead of computing the function

$$(y_1, y_2) = f(x_1, x_2)$$

we first transform it to the function

$$(y_1 \oplus r_1, y_2 \oplus r_2) = g(\{x_1, r_1\}, \{x_2, r_2\})$$

Note if we computed the function g via Yao's method then we do not need P_2 to send the garbled outputs back to P_1.

- P_2 simply obtains output $(y_1 \oplus r_1, y_2 \oplus r_2)$
- P_2 obtains their output from $y_2 \oplus r_2$ and r_2.
- P_2 sends P_1 the value $y_1 \oplus r_1$
- P_1 obtains their output from $y_1 \oplus r_1$ and r_1.

Dual Execution

Now compute g two ways

- P_2 acts as evaluator for P_1's circuit
- P_1 acts as evaluator for P_2's circuit

Player P_2 will obtain $y_2 \oplus r_2$ in two ways

- Once on their own
- Once from P_1's evaluation of P_2's circuit

If they differ then P_1's circuit creation was invalid

In the second evaluation P_1 will obtain $y_1 \oplus r_1$ in two ways

- Once on their own
- Once from P_2's evaluation of P_1's circuit

If they differ then P_2's circuit creation was invalid

Attack 1 on Dual Execution

Suppose P_1 is an adversary

Player P_1 instead of sending a garbling of the function

$$(y_1 \oplus r_1, y_2 \oplus r_2) = g(\{x_1, r_1\}, \{x_2, r_2\})$$

could send a garbling of the function

$$(x_2 \oplus r_1, y_2 \oplus r_2) = g'(\{x_1, r_1\}, \{x_2, r_2\})$$

From the returned value by P_2 the attacker can learn x_2

Dual Execution

To avoid this we do an equality check without sending the values back

- The two parties execute Yao twice in each direction
- Party P_1 obtains $A_1 = (y_1 \oplus r_1, y_2 \oplus r_2)$.
- Party P_2 obtains $A_2 = (y_1 \oplus r_1, y_2 \oplus r_2)$.
- The parties run the equality check on A_1 and A_2.
- If OK P_1 outputs y_1 and P_2 outputs y_2.

This uses the string equality testing protocol from earlier

Attack 2 on Dual Execution

Again we assume P_1 is the bad party.

Suppose P_1 wants to learn the first bit of x_2, call this bit z.

Instead of creating a circuit for the function g above, party P_1 now creates a circuit for

$$(y_1 \oplus r_1 \oplus (z \| 0 \ldots 0), y_2 \oplus r_2) = g(\{x_1, r_1\}, \{x_2, r_2\})$$

When we run the equality check

- The equality check passes if $z = 0$
- The equality check fails if $z = 1$

This party P_1 learns one bit (indeed any bit he chooses) of P_2's input

Cut-and-Choose

The previous attack was because P_1 could send P_2 a bogus circuit.

So we stop this by now making each party send each other two circuits.

So party P_2 receives two garbled circuits from P_1, G_0 and G_1.

Party P_2 now flips a bit $b \in \{0, 1\}$ and asks P_1 to reveal the randomness used to generate G_{1-b}.

- P_2 can then check that G_{1-b} is the correct function.

Then P_2 evaluates G_1 as usual.

This means P_1 only has a 50 percent change of providing a fake circuit.

Attack 1 on the OT

But still P_1 can learn a bit of P_2's input with 100 percent probability.

Now when executing the OT protocol for the un-opened circuit for P_2's first input wire, party P_1 provides the input to the OT of (r^0, f^1)

- r^0 is the correct input wire value for the bit 0.
- f^1 is a fake input wire value for the bit 1.

If P_2's input bit was zero then the protocol will succeed

If P_2's input bit was one, then the protocol will not succeed

- As the equality test will fail, as P_2 will evaluate garbage.

Fixing the OT

The problem is that each OT input corresponds to a single wire input.

We now replace the circuit

$$(y_1 \oplus r_1, y_2 \oplus r_2) = g(\{x_1, r_1\}, \{x_2, r_2\})$$

by the circuit

$$(y_1 \oplus r_1, y_2 \oplus r_2) = h(\{\mathbf{z}_1, r_1\}, \{\mathbf{z}_2, r_2\})$$

where $\mathbf{z}_i \in \{0,1\}^{s_2 \cdot |x_i|}$ and we define (say)

$$x_1 = (\oplus_{i=1}^{s_2} \mathbf{z}_1^{(i)}, \oplus_{i=1}^{s_2} \mathbf{z}_1^{(i+s_2)}, \ldots, \oplus_{i=1}^{s_2} \mathbf{z}_1^{(i+|x_1|-s_2)}).$$

Thus each bit of x_1 is a sum of s_2 bits in $\mathbf{z}_1$

Now if a bit in the OT is provided wrong the protocol will always abort (with high probability).

Majority Voting

We want the cheating probability to be smaller than $1/2$.

So we produce more circuits, say s_1 of them

We open half of them to check all is OK with their construction

But now we have $s_1/2$ circuits to evaluate

To obtain output we take the majority verdict

Majority Voting

To win this game the adversary has to

- Get all opened circuits to be good.
- Get majority of the evaluated circuits to be bad.

Probability that an adversary gets an invalid circuit passed this check is about $2^{-s_1/4}$.

For intuition about this, suppose half are bad and half are good

- A single bad circuit is not opened with probability $1/2$
- We need the bad circuits to agree with the majority output
- Thus we need more than $s_1/4$ bad circuits to be not opened.

A more careful analysis gives a better value for s_1.

Why Take Majority?

The problem is

- If you abort, since you know you are being cheated, you leak information
- The Adv can make the minority wrong circuits be ones which compute garbage if an input bit is one.
- Thus we need to take the majority as the answer and not abort.

Attack 2 on the OT

Recall we now take s_1 circuits and evaluate them.

Suppose P_2's input size is ℓ bits then we need to evaluate $\ell \cdot s_2$ OT's.

The attacker could send *different* inputs in the different executions of the OT on the *same* input bit.

To fix this we need to execute the $\ell \cdot s_2$ OT's as only ℓ OTs.

Each OT transfers the input wire labels for a specific wire for *all* circuits in one go.

Summary

This game of attack-and-fix does actually terminate...

And further optimizations can be provided.

We have just given the flavour of the methodology above.

In the end we can produce two party actively secure computation using Yao

It is actually both practical and efficient

Can evaluate circuits of many billions of gates quite easily

Any Questions?