Secure computing, fifth lecture

Secure multi-party computation: garbled circuits and oblivious transfer

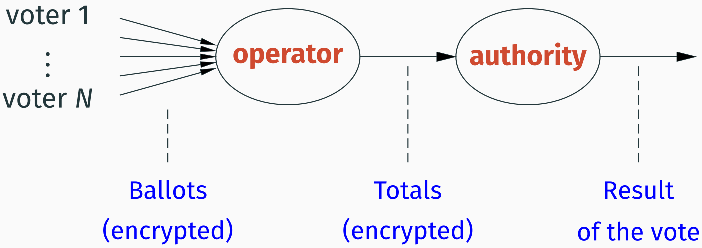
Xavier Leroy

2025-12-04

Collège de France, chair of Software sciences
xavier.leroy@college-de-france.fr

Key sharing and threshold decryption

An example of electronic vote (reminder)



A key pair (pk, sk) for a weakly-homomorphic cipher.

Ballots are encrypted with the public key pk.

Ballots are counted by homomorphic addition.

The total is to be decrypted with the private key sk.

We want to share the key sk between n trustees, so that $k < n$ trustees can decrypt the total.

A naive multi-party algorithm

We share the key *sk* between the n trustees using Shamir sharing (polynomials of degree $t = k - 1$).

Once the ballots are counted, k trustees reveal their shares, recover *sk*, and decrypt the total.

Problems:

- Any trustee can decrypt any ballot, not just the total.
- The key cannot be reused for another vote.

The ElGamal cipher (reminder)

A finite group $(G, \cdot)$ of order q generated by g.

Private key: $s \in \{1, \ldots, q-1\}$.

Public key: $h \stackrel{def}{=} g^s$.

Randomized encryption:

$$\mathcal{E}_h(m) = (g^r, h^r \cdot m) \quad \text{with } r \in \{1, \ldots, q-1\} \text{ random}$$

Decryption:

$$\mathcal{D}_s((a, b)) = b / a^s$$

Multi-party decryption

Consider a full additive sharing of the private key s between n participants:

$$s = s_1 + \cdots + s_n \pmod{q}$$

To jointly decrypt (a, b)

- each participant i computes $y_i = a^{s_i}$ and sends it to the others;
- one or several participants compute $y = y_1 \cdots y_n$, then b/y.

This correctly decrypts the message, since

$$y = a^{s_1} \cdots a^{s_n} = a^{s_1 + \cdots + s_n} = a^s$$

The private key s is not revealed, only a multiplicative sharing of $y = a^s$.

Threshold decryption

If we use Shamir sharing, or some other LSSS linear sharing, the private key is a linear combination of the shares:

$$s = \lambda_1 s_1 + \cdots + \lambda_n s_n \pmod{q}$$

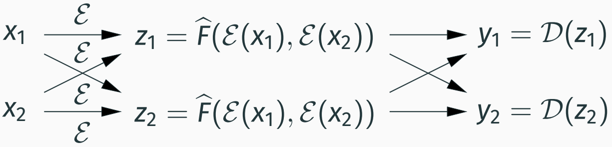
where some of the λ_i can be 0 if we do not need the corresponding shares. ($\Rightarrow$ lecture #4)

In this case, each participant i computes $y_i = a^{\lambda_i s_i}$, and we have

$$y = y_1 \cdots y_n = a^{\lambda_1 s_1} \cdots a^{\lambda_n s_n} = a^{\lambda_1 s_1 + \cdots + \lambda_n s_n} = a^s$$

Multi-party computation and homomorphic encryption

We can implement a multi-party computation $y = F(x_1, \ldots, x_n)$ by homomorphic evaluation of the circuit F.

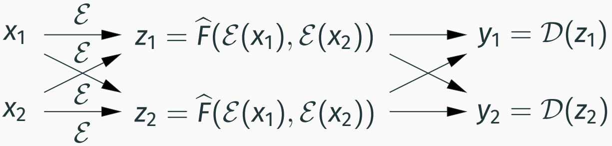


We assume the private key sk is shared between the participants, and the public key pk is known to all.

Each participant i sends its encrypted secret $\mathcal{E}_{pk}(x_i)$ to the other participants.

Multi-party computation and homomorphic encryption

We can implement a multi-party computation $y = F(x_1, \dots, x_n)$ by homomorphic evaluation of the circuit F.



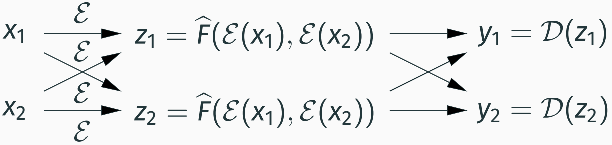
Each participant i computes

$$z_i = \widehat{F}(\mathcal{E}_{pk}(x_1), \dots, \mathcal{E}_{pk}(x_n))$$

where $\widehat{F}$ is the homomorphic evaluation of F.

Multi-party computation and homomorphic encryption

We can implement a multi-party computation $y = F(x_1, \ldots, x_n)$ by homomorphic evaluation of the circuit F.

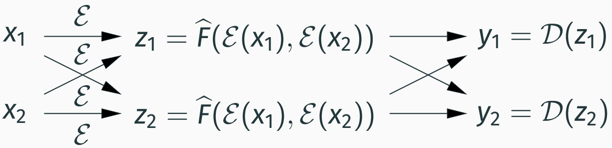


All the participants cooperate to decrypt the z_i:

$$y_i = \mathcal{D}_{sk}(z_i) \qquad \text{(without revealing } sk\text{)}$$

and check that $y_1 = \ldots = y_n$.

We can implement a multi-party computation $y = F(x_1, \ldots, x_n)$ by homomorphic evaluation of the circuit F.



Point in favor: the number of communication rounds is independent of the multiplicative depth of F.

Point against: homomorphic evaluation is costly in CPU time.

Yao’s garbled circuits

Yao's millionaire problem

(Andrew C. Yao, *Protocols for Secure Computation*, SFCS 1982.)

Alice and Bob wish to know who is the wealthiest, without revealing their exact wealth to the other.

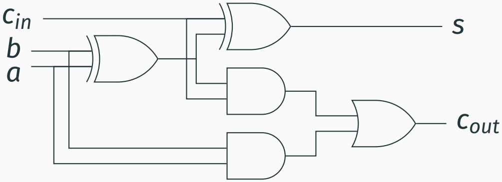
A two-party variant of the call for tenders problem.

Formally: compute the Boolean value of $a \geq b$
while keeping a and b secret.

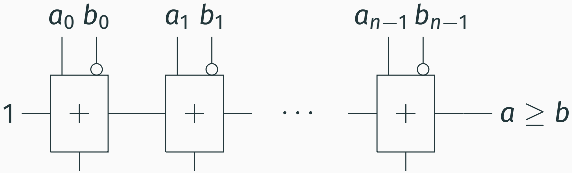
(Variant: the socialist millionaire problem, where the result is the Boolean value of $a = b$.)

A Boolean circuit for comparison

Full adder:



n-bit comparator:



Secure two-party evaluation of the circuit

Using one of the secret sharing protocols from lecture #4, for example the GMW protocol.

- Alice writes her wealth in binary $A = \sum a_i 2^i$ and shares the secret bits $a_0, \ldots, a_{39}$ with Bob.
- Bob writes his wealth in binary $B = \sum b_i 2^i$ and shares the secret bits $b_0, \ldots, b_{39}$ with Alice.
- Alice and Bob jointly evaluate the comparison circuit.
- Once the sharing $[c]$ of the result is computed, Alice and Bob reveal it, obtaining c, which is $a \geq b$.

Potential problem: the amount of communication
(3 multiplications par bit $\rightarrow$ at least 120 communications).

Yao's garbled circuits

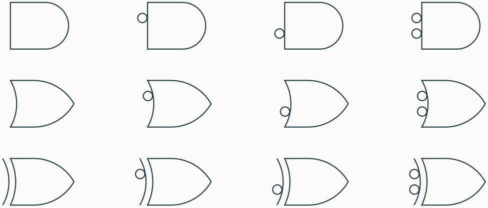
$$c = F(a, b) \qquad \begin{aligned} a &: \text{Alice's private data} \\ b &: \text{Bob's private data} \\ c &: \text{shared results} \end{aligned}$$

An asymmetric alternative to secret sharing.

1. Alice prepares a “garbled” variant of the circuit F and sends it to Bob, along with her secrets a after garbling.
2. Bob garbles his secrets b using oblivious transfer with Alice.
3. Bob evaluates the garbled circuit, obtaining $c = F(a, b)$ garbled. (Purely local evaluation; no communication.)
4. Bob sends this result to Alice, who un-garbles it and announces c.

The logical gates used

We consider AND, OR, XOR gates, possibly with a negation on one or both inputs:



No need for NOT gates: negation is performed on the input of the next gate.

Representing gates by truth tables

Each gate F can be represented by its truth table:

0	0	value of $F(0,0)$
0	1	value of $F(0,1)$
1	0	value of $F(1,0)$
1	1	value of $F(1,1)$

Examples:

=

0	0	0
0	1	1
1	0	1
1	1	0

=

0	0	0
0	1	1
1	0	0
1	1	0

=

0	0	1
0	1	1
1	0	1
1	1	0

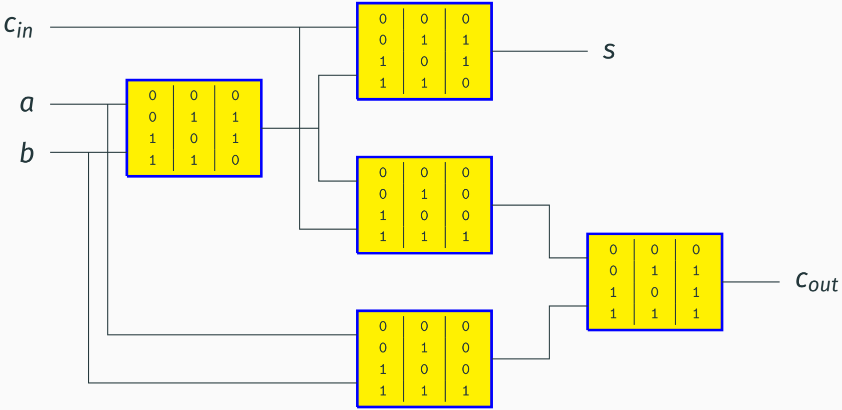
Garbling the wires

For each wire w of the circuit, Alice chooses two bit-vectors w_0 representing bit 0 and w_1 representing bit 1.

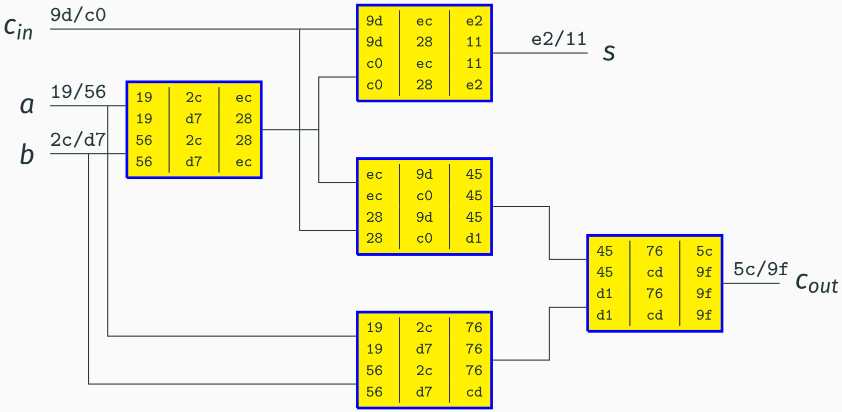
(Each w_b is one half of a symmetric encryption key, i.e. a 128-bit vector for AES-256.)

She rewrites the truth tables accordingly.

Example: garbling the wires of the full adder



Example: garbling the wires of the full adder



Encrypting the logical gates

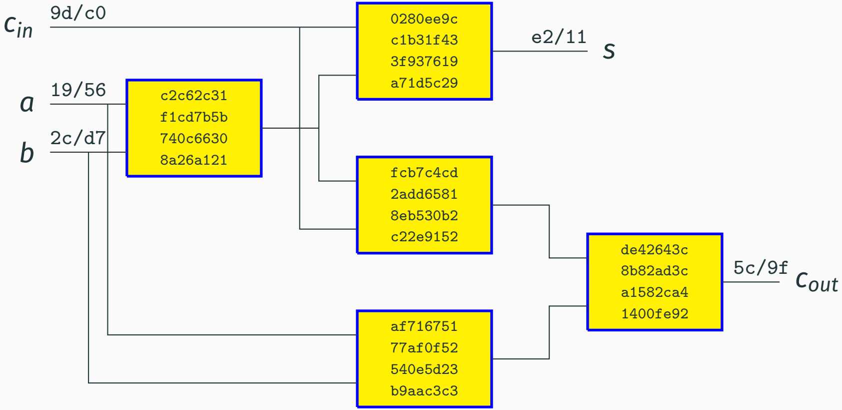
For the gate number g, with inputs a, b and output c:

$$\begin{array}{c|c|l} a_0 & b_0 & c_{F(0,0)} \\ a_0 & b_1 & c_{F(0,1)} \\ a_1 & b_0 & c_{F(1,0)} \\ a_1 & b_1 & c_{F(1,1)} \end{array} \quad \Longrightarrow \quad \begin{array}{l} \{ \quad \mathcal{E}_{a_0 \| b_0}(g \parallel c_{F(0,0)}), \\ \quad\;\; \mathcal{E}_{a_0 \| b_1}(g \parallel c_{F(0,1)}), \\ \quad\;\; \mathcal{E}_{a_1 \| b_0}(g \parallel c_{F(1,0)}), \\ \quad\;\; \mathcal{E}_{a_1 \| b_1}(g \parallel c_{F(1,0)}) \quad \} \end{array}$$

Each possible value of the output (c_0 or c_1 depending on $F(i,j)$) is encrypted with the secret key $a_i \parallel b_j$
(the concatenation of the two input values).

The 4 resulting ciphertexts are permuted randomly.

Example: encryption of the gates of the full adder



Evaluating an encrypted gate

$$
\begin{array}{c|c|c}
a_0 & b_0 & c_{F(0,0)} \\
a_0 & b_1 & c_{F(0,1)} \\
a_1 & b_0 & c_{F(1,0)} \\
a_1 & b_1 & c_{F(1,1)}
\end{array}
\quad \Longrightarrow \quad
\begin{array}{ll}
\{ & \mathcal{E}_{a_0 \| b_0}(g \parallel c_{F(0,0)}), \\
 & \mathcal{E}_{a_0 \| b_1}(g \parallel c_{F(0,1)}), \\
 & \mathcal{E}_{a_1 \| b_0}(g \parallel c_{F(1,0)}), \\
 & \mathcal{E}_{a_1 \| b_1}(g \parallel c_{F(1,0)}) \quad \}
\end{array}
$$

Bob only knows the gate identifier g, its 4 encrypted lines, and the garbled inputs a, b.

He decrypts the 4 lines with the key $a \parallel b$.

With very high probability, only one decryption is of the form $g \parallel c$ for some code value c. (The other decryptions are noise.)

This c is the garbled output of the gate.

Evaluating an encrypted gate

$$\begin{array}{c|c|c} a_0 & b_0 & c_{F(0,0)} \\ a_0 & b_1 & c_{F(0,1)} \\ a_1 & b_0 & c_{F(1,0)} \\ a_1 & b_1 & c_{F(1,1)} \end{array} \quad \Longrightarrow \quad \left\{ \begin{array}{l} \mathcal{E}_{a_0 \| b_0}(g \parallel c_{F(0,0)}), \\ \mathcal{E}_{a_0 \| b_1}(g \parallel c_{F(0,1)}), \\ \mathcal{E}_{a_1 \| b_0}(g \parallel c_{F(1,0)}), \\ \mathcal{E}_{a_1 \| b_1}(g \parallel c_{F(1,0)}) \end{array} \right\}$$

Bob only knows the gate identifier g, its 4 encrypted lines, and the garbled inputs a, b.

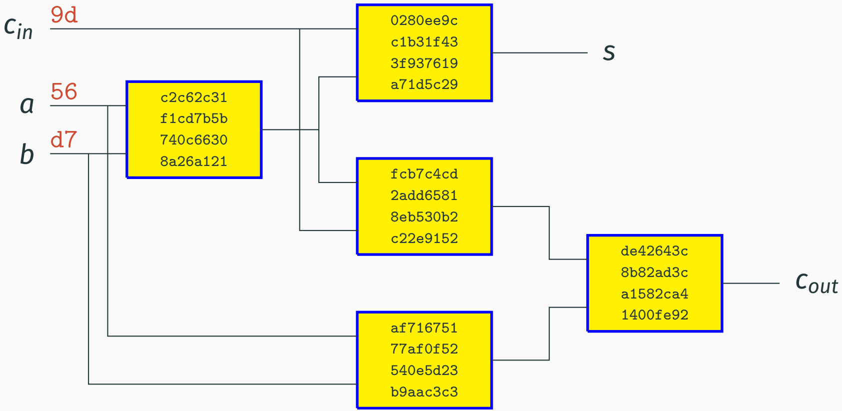
He decrypts the 4 lines with the key $a \parallel b$.

With very high probability, only one decryption is of the form $g \parallel c$ for some code value c. (The other decryptions are noise.)

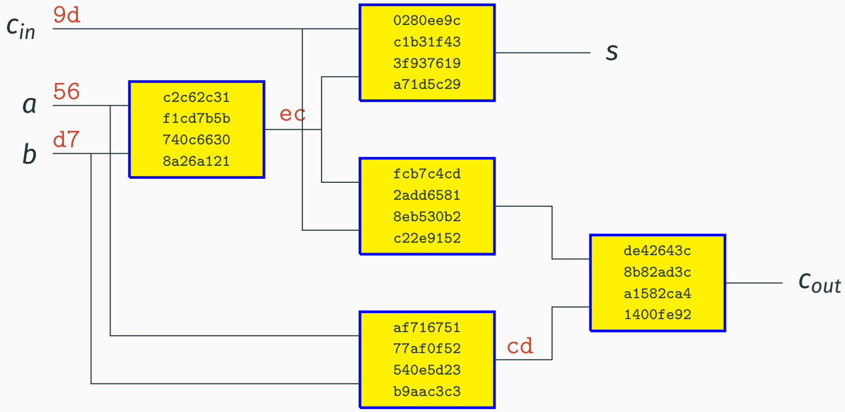
This c is the garbled output of the gate.

Bob is able to evaluate the logic gate, but does not know which bits a, b, c stand for, and does not know the other 3 lines.

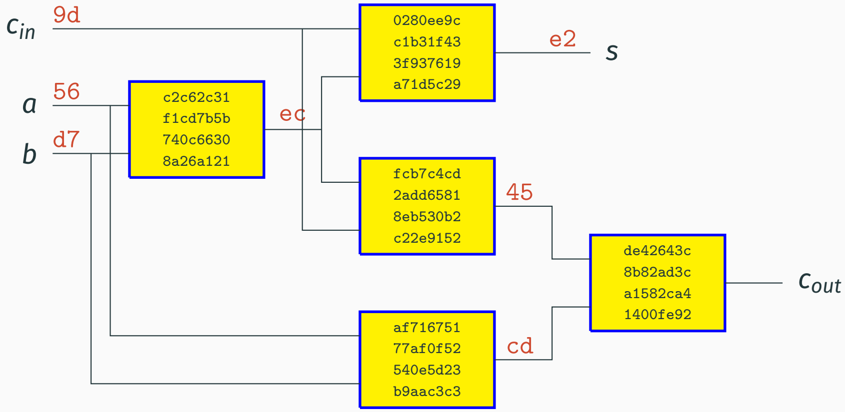
Example: evaluating the full adder



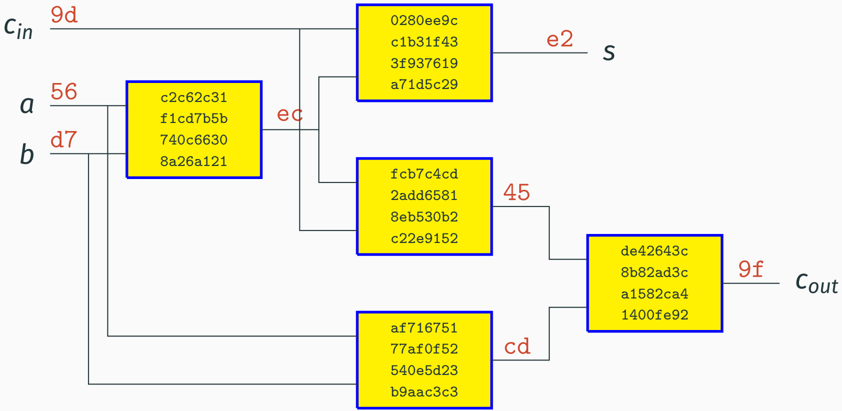
Example: evaluating the full adder



Example: evaluating the full adder



Example: evaluating the full adder



The full protocol for garbled circuits

1. Alice garbles the wires: $w \mapsto w_0, w_1$ random.
   Alice garbles the circuit and sends it to Bob.
   For each of her inputs a with value x, she sends the garbled input a_x to Bob.

2. Oblivious transfer: for each input b of Bob's with value y, Alice offers b_0 and b_1, Bob chooses y, Bob receives b_y.

3. Bob evaluates the garbled circuit (locally).

4. For each circuit output c, Bob sends Alice its garbled value c_0 or c_1, Alice recovers the corresponding 0/1 bit, and announces it.

The full protocol for garbled circuits

1. Alice garbles the wires: $w \mapsto w_0, w_1$ random.
   Alice garbles the circuit and sends it to Bob.
   For each of her inputs a with value x, she sends the garbled input a_x to Bob.

2. Oblivious transfer: for each input b of Bob's with value y, Alice offers b_0 and b_1, Bob chooses y, Bob receives b_y.

3. Bob evaluates the garbled circuit (locally).

4. For each circuit output c, Bob sends Alice its garbled value c_0 or c_1, Alice recovers the corresponding 0/1 bit, and announces it.

(Variant: use a trivial garbling $c \mapsto 0, 1$ for the outputs c.
Then, Bob knows the result in the clear and can announce it himself.)

Security of garbled circuits

Passive security:

- Bob learns nothing about Alice's secrets a (they are masked by the garbling $0, 1 \mapsto a_0, a_1$).
- Alice learns nothing about Bob's secrets b (assuming that the oblivious transfer is secure).

Active security:

- If Bob does not follow the protocol, with high probability he'll get impossible values (neither c_0 nor c_1) for the output wires c. Alice will spot this.
- Alice can cheat in many ways. For example she can send a garbled circuit that outputs Bob's secret: $F(a, b) = b$.

Speeding up the evaluation of a garbled gate

To evaluate a garbled gate $\{z_1, \ldots, z_4\}$ on the inputs a, b, we need to decrypt 2.5 lines z_i on an average, 4 in the worst case.

We can use the least significant bits of a and b to know in advance which z_i to decrypt.

Speeding up the evaluation of a garbled gate

We choose the wire garblings $k \mapsto k_0, k_1$ so that $LSB(k_0) \neq LSB(k_1)$.

We sort the 4 ciphertexts $\mathcal{E}_{a_x \| b_y}(g \parallel c_{F(x,y)})$ by $2 \times LSB(a_x) + LSB(b_y)$.

The evaluator knows which line to decrypt: the line number $2 \times LSB(a) + LSB(b)$.

Example: initial table / table sorted by LSB.

19	2c	af716751
19	d7	77af0f52
56	2c	540e5d23
56	d7	b9aac3c3

$\Longrightarrow$

56	2c	540e5d23
56	d7	b9aac3c3
19	2c	af716751
19	d7	77af0f52

$\Longrightarrow$

540e5d23
b9aac3c3
af716751
77af0f52

Speeding up the decryption

We can use a hash function $\mathcal{H}$ to encrypt the lines.

The four z_i lines are computed as

$$z_{2\times LSB(a_x)+LSB(b_y)} = \mathcal{H}(g \parallel a_x \parallel b_y) \oplus c_{F(x,y)}$$

The decryption performed during the execution of the gate is

$$c = z_{2\times LSB(a)+LSB(b)} \oplus \mathcal{H}(g \parallel a \parallel b)$$

Speeding up the decryption

We can use a hash function $\mathcal{H}$ to encrypt the lines.

The four z_i lines are computed as

$$z_{2\times LSB(a_x)+LSB(b_y)} = \mathcal{H}(g \parallel a_x \parallel b_y) \oplus c_{F(x,y)}$$

The decryption performed during the execution of the gate is

$$c = z_{2\times LSB(a)+LSB(b)} \oplus \mathcal{H}(g \parallel a \parallel b)$$

(The hash function can be implemented efficiently using a block cipher such as AES and a key known to both participants.)

Free XOR gates

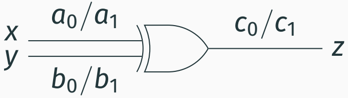
For a wire k, instead of picking random k_0 and k_1, we can pick k_0 randomly and take $k_1 = k_0 \oplus \Delta$ where Δ is a secret chosen by Alice. (Δ must be odd.)

The garbling of bit x over wire k is, then, $k_0 \oplus x \cdot \Delta$.

Free XOR gates

For a wire k, instead of picking random k_0 and k_1,
we can pick k_0 randomly and take $k_1 = k_0 \oplus \Delta$
where Δ is a secret chosen by Alice. (Δ must be odd.)

The garbling of bit x over wire k is, then, $k_0 \oplus x \cdot \Delta$.



Consider the XOR gate above, If we choose $c_0 = a_0 \oplus b_0$, this gate evaluates without decryption, simply as the XOR of its inputs:

$$\begin{aligned} a_x \oplus b_y &= (a_0 \oplus x \cdot \Delta) \oplus (b_0 \oplus y \cdot \Delta) \\ &= (a_0 \oplus b_0) \oplus (x \oplus y) \cdot \Delta = c_0 \oplus z \cdot \Delta = c_z \end{aligned}$$

Active security: the cut-and-choose technique

Can we make sure that the garbled circuit constructed by Alice does compute the function F and not the function $F'(a, b) = b$ for example?

The cut-and-choose technique:

- Alice constructs n garbled circuits $C_1, \ldots, C_n$ using different randomness, and sends them all to Bob.
- Bob chooses $i \in \{1, \ldots, n\}$ and ask Alice the randomness used to construct C_j for all $j \neq i$.
- Using the randomness, Bob can check that the circuits $C_j, j \neq i$ are correct garblings of F.
- Bob and Alice use the circuit C_i to continue the protocol.

Active security: expanding the secret inputs

Another possible attack by Alice:

2. Oblivious transfer: for each input b of Bob's with value y, Alice offers b_0 and b_1, Bob chooses y, Bob receives b_y.

Instead of offering b_0 and b_1, Alice could offer b_0 and 0.

If Bob produces a well-formed result nonetheless, it means that he did not use the value 0. Alice learns that $y = 0$.

Counter-measure: expand the input b into n inputs $b_1, \ldots, b_n$, with a little circuit that computes $b = b_1 \oplus \cdots \oplus b_n$.
Combine this with the cut-and-choose technique.

(Y. Lindell, B. Pinkas: *An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries.* J. Cryptol, 2015).

Oblivious transfer

Oblivious Transfer (OT)

A protocol between two participants:

- Alice (the sender) knows n values $m_1, \ldots, m_n$.
- Bob (the receiver) chooses $i \in \{1, \ldots, n\}$.

At the end of the protocol,

- Bob knows the value m_i.
- Alice does not know Bob's choice i.
- Bob learnt nothing about the other values m_j for $j \neq i$.

The EGL protocol for 1-out-of-2 oblivious transfer

(S. Even, O. Goldreich, A. Lempel, *A Randomized Protocol for Signing Contracts*, CRYPTO 1982.)

Uses a public-key cipher (*Keygen*) for which we can randomly draw “fake” public keys (*PubKeySamp*) indistinguishable from the “genuine” public keys.

1. Bob the receiver draws a key pair $(pk, sk) \leftarrow Keygen$ and a fake public key $pk' \leftarrow PubKeySamp$.

If he chooses $i = 0$, he sends (pk, pk') to Alice.
If he chooses $i = 1$, he sends (pk', pk) to Alice.

The EGL protocol for 1-out-of-2 oblivious transfer

2. Alice the sender receives two public keys pk_0, pk_1 and encrypts her messages with these keys:

$$c_0 = \mathcal{E}_{pk_0}(m_0) \qquad c_1 = \mathcal{E}_{pk_1}(m_1)$$

She sends the ciphertexts c_0 and c_1 to Bob.

3. Bob receives c_0, c_1 and decrypts c_i using his private key:

$$m_i = \mathcal{D}_{sk}(c_i)$$

Correctness: pk_i is the public key associated with sk, hence we have $\mathcal{D}_{sk}(\mathcal{E}_{pk_i}(m_i)) = m_i$.

Security of the EGL protocol

Passive security:

- Alice receives two public keys but cannot distinguish the genuine one from the fake one.
  $\rightarrow$ Alice learns nothing about Bob's choice i.
- Bob receives two ciphertexts c_0, c_1 and can decrypt c_i but not c_{1-i} (he doesn't have a private key that matches pk').
  $\rightarrow$ Bob learns nothing about m_{1-i} .

Active security: Bob can easily cheat.

Instead of $pk' \leftarrow PubKeySamp$, he draws $(pk', sk') \leftarrow Keygen$.

Then, he can decrypt both messages sent by Alice, and learn both m_0 and m_1.

Variant: 1-out-of-4 oblivious transfer

(Easily extended to 1 out of 2^n.)

1. Bob the receiver draws two key pairs and two fake keys

$$\begin{aligned} (pk_0, sk_0) &\leftarrow Keygen & \qquad pk_0' &\leftarrow PubKeySamp \\ (pk_1, sk_1) &\leftarrow Keygen & \qquad pk_1' &\leftarrow PubKeySamp \end{aligned}$$

He writes his choice i in binary: $i = i_0 + 2i_1$.

He sends (pk_0, pk_0') if $i_0 = 0$ or (pk_0', pk_0) if $i_0 = 1$.

He sends (pk_1, pk_1') if $i_1 = 0$ or (pk_1', pk_1) if $i_1 = 1$.

Variant: 1-out-of-4 oblivious transfer

2. Alice the sender receives two pairs of public keys (u_0, u_1) and (v_0, v_1). She uses them to perform double encryption of her four messages:

$$
\begin{aligned}
c_0 &= \mathcal{E}_{u_0}(\mathcal{E}_{v_0}(m_0)) \\
c_1 &= \mathcal{E}_{u_1}(\mathcal{E}_{v_0}(m_1)) \\
c_2 &= \mathcal{E}_{u_0}(\mathcal{E}_{v_1}(m_2)) \\
c_3 &= \mathcal{E}_{u_1}(\mathcal{E}_{v_1}(m_3))
\end{aligned}
$$

3. Bob receives $c_0, \ldots, c_3$ and decrypts c_i with his private keys:

$$m_i = \mathcal{D}_{sk_0}(\mathcal{D}_{sk_1}(c_i))$$

The NP protocol: oblivious transfer with active security

(M. Naor, B. Pinkas, *Efficient oblivious transfer protocols*, SODA 2001.)

Idea: give Alice a way to check that only one of the keys pk_1, pk_2 is genuine, in the sense that Bob knows the corresponding private key.

We use the following property of the ElGamal cipher:

> if $pk = g^s$ is a genuine public key.
> and C an arbitrary group element, fixed in advance,
> then C/pk is a fake public key
>
> > (it is computationally hard to find t such that $C/pk = g^t$).

The NP protocol: oblivious transfer with active security

0. Beforehand: Alice draws C randomly and sends it to Bob.

1. Bob draws a key pair $(pk, sk) = (g^s, s)$ with $s \in \{1, \ldots, q-1\}$ random.

If he chooses $i = 0$, he sends $(pk, C/pk)$ to Alice.
If he chooses $i = 1$, he sends $(C/pk, pk)$ to Alice.

The NP protocol: oblivious transfer with active security

2. Alice receives two public keys pk_0, pk_1.

She checks that $pk_0 \cdot pk_1 = C$ and fails otherwise.

She encrypts her two messages with pk_0, pk_1:

$$c_0 = \mathcal{E}_{pk_0}(m_0) \qquad c_1 = \mathcal{E}_{pk_1}(m_1)$$

She sends the ciphertexts c_0 and c_1 to Bob.

3. Bob receives c_0, c_1 and decrypts c_i with his private key:

$$m_i = \mathcal{D}_{sk}(c_i)$$

Sécurité du protocole NP

Passive security:

- Alice cannot distinguish the fake key C/pk from the genuine key pk, because C/pk is as random as pk is.
- Bob cannot easily find a secret key y matching C/pk:
  if he could find y, he would know $z = s + y$ such that $C = g^z$, and he would have computed the discrete logarithm of C.

Active security: Bob has zero degree of freedom in choosing the fake key; it must be C/pk for Alice to accept it.

Variant: random oblivious transfer (ROT)

A variant of OT where Alice's messages and Bob's choice are randomly chosen by the protocol.

At the beginning of the protocol: no information.

At the end of the protocol:

- Alice knows two random messages r_0 and r_1.
- Bob knows one random bit $b \in \{0, 1\}$ and the message r_b .
- Alice doesn't know the bit b .
- Bob knows nothing about r_{1-b} .

Building OT from ROT

Initially:

Alice has two m_0 and m_1; Bob has a choice $b \in \{0, 1\}$.

Execution of the ROT protocol:

Alice receives random r_0, r_1; Bob receives $s \in \{0, 1\}$ and r_s.

Bob computes $t = b \oplus s$ and sends it to Alice (Masking.)

If $t = 0$, Alice sends $c_0 = m_0 \oplus r_0$ and $c_1 = m_1 \oplus r_1$ to Bob.
If $t = 1$, Alice sends $c_0 = m_0 \oplus r_1$ and $c_1 = m_1 \oplus r_0$ to Bob.
(Masking.)

Bob recovers $m_b = c_b \oplus r_s$.

(Correctness: if $t = 0$, we have $s = b$ and $c_b \oplus r_s = m_b \oplus r_b \oplus r_b = m_b$.
If $t = 1$, we have $s = 1 - b$ and $c_b \oplus r_s = m_b \oplus r_{1-b} \oplus r_{1-b} = m_b$.)

Extending an oblivious transfer protocol

All OT protocols rely on public-key encryption, which is expensive.

The OT extension problem: after performing n oblivious transfers using public-key encryption, can we perform $N \gg n$ oblivious transfers without any public-key encryption?

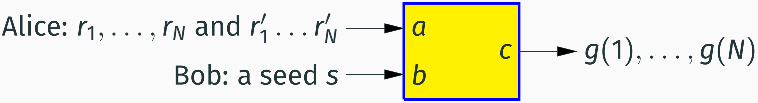
Random oblivious transfer using a garbled circuit

(D. Beaver, *Correlated pseudorandomness and the complexity of private computations*, STOC 1996.)

Assume given a pseudo-random generator *PRNG*:

$$PRNG : \text{seed} \times \mathbb{N} \to \text{bit}$$

Alice prepares a garbling of the following circuit:



Outputs: $g(i) = \begin{cases} (0, r_i) & \text{if } PRNG(s, i) = 0 \\ (1, r'_i) & \text{if } PRNG(s, i) = 1 \end{cases}$

Random oblivious transfer using a garbled circuit

Alice garbles the circuit and sends it to Bob.

Alice draws (pseudo-)randomly $2N$ numbers $r_1, \ldots, r_N$ and $r'_1, \ldots, r'_N$, and sends them to Bob after garbling.

Bob randomly chooses a seed s and has it garbled by Alice using standard OT (n transfers if n is the bit size of the seed.)

Bob runs the circuit, obtaining $g(1), \ldots, g(N)$.

Result: the pairs $((r_i, r'_i), g(i))$ for $i = 1, \ldots, N$ are N random oblivious transfers; they can be used later to perform N cryptography-free oblivious transfers.

Extending an oblivious transfer protocol

Beaver's construction shows that we can obtain N OTs without public-key cryptography from $n \ll N$ standard OTs.

Main limitation: the size of the garbled circuit.

For better OT extension techniques, see
section 7.3 of the book *A pragmatic introduction to MPC*
and Geoffroy Couteau's seminar.

Summary

Summary on Yao's garbled circuits

One of the first realizations of secure multi-party computation.

One of the most efficient, still today !
(Few communication rounds + symmetric cryptography.)

Non-obvious extension to $n > 2$ participants.

Passive security is easily achieved
(but: never evaluate twice the same garbled circuit!).

Active security can be achieved but is expensive
(cut-and-choose techniques that sacrifice many circuits).

Summary on oblivious transfer

A primitive used in many protocols.

Requires some amount of public-key cryptography.

Extension techniques are able to amortize the cost of public-key crypto on a large number of transfers.

References

References

For more details:

- *A pragmatic introduction to secure multi-party computation*, David Evans, Vladimir Kolsnikov, Mike Rosulek, NOW Publishers, 2018.
  Section 3.1: Yao's garbled circuits.
  Section 3.7: oblivious transfer.

Advanced reading:

- *Foundations of garbled circuits*, Mihir Bellare, Viet Tung Hoang, Phillip Rogaway, CCS 2012, https://doi.org/10.1145/2382196.2382279
- *Oblivious Transfer Is in MiniQCrypt*, Alex B. Grilo, Huijia Lin, Fang Song, Vinod Vaikuntanathan, Eurocrypt 2021, https://doi.org/10.1007/978-3-030-77886-6_18