Secure multiparty computation

30.09.2021
Principle

- There is a (randomized) function $f : (\{0, 1\}^\ell)^n \rightarrow (\{0, 1\}^\ell)^n$.
- There are $n$ parties, $P_1, \ldots, P_n$.
  - Some of them may be adversarial.
  - Two forms of adversarial behaviour:
    - Semi-honest — will follow the protocol as prescribed, but tries to deduce extra information from what it sees.
    - Malicious — will not necessarily follow the protocol.
- Party $P_i$ has the bit-string $x_i \in \{0, 1\}^\ell$.
- Party $P_i$ wants to learn $y_i$, where

  $$(y_1, \ldots, y_n) = f(x_1, \ldots, x_n) .$$

- No party $P_i$ may learn anything beyond $x_i$ and $y_i$. 

30.09.2021
Examples

- Two millionaires want to determine who is richer.
  - Revealing one’s net worth to the other party means losing one’s face if the other party turns out to be much richer.
- Alice and Bob are considering to start dating each other.
  - Revealing that one is interesting in dating means losing one’s face if the other party is not interested.
- Three cryptographers are dining in a restaurant. The waiter informs them that the bill has already been paid. The cryptographers want to know whether the payer was one of them (who wants to remain anonymous) or the NSA.
- Everything else...
Pieces

- The number of parties $n$.
  - $n = 2$
  - $n \geq 3$

- A function $f$, represented as a Boolean circuit.
  - Deterministic or randomized.
  - Common output or separate outputs.

- A $n$-party protocol $\Pi$. Two main techniques:
  - “Garbled circuits”
  - Secret-sharing the values on wires.

- The adversary (a coalition of parties)
  - maximum size
  - semi-honest or malicious
    - Different levels of tolerable malice
  - non-adaptive or adaptive
Consider the deterministic two-party, semi-honest case.

A protocol $\Pi$ securely evaluates the function $f(x_1, x_2)$ if there exist PPT simulators $S_1$ and $S_2$, such that for all $x_1$ and $x_2$:

- The distribution of $S_1(x_1, f_1(x_1, x_2))$ is indistinguishable from the view of the first party in the execution of $\Pi(x_1, x_2)$.
- The distribution of $S_2(x_2, f_2(x_1, x_2))$ is indistinguishable from the view of the second party in the execution of $\Pi(x_1, x_2)$. 
Example: Yao’s millionaires’ problem

- Alice has $a$ million dollars, Bob has $b$ million dollars
- Alice and Bob want to privately compute $[a \ ? \ b]$?
- What follows, is (similar to) Yao’s original protocol
- Assume $1 \leq a, b \leq \ell$, where $\ell$ is rather small.
  - The complexity of Yao’s protocol is about $O(\ell)$
Set-up

- Let \((\mathcal{E}, \mathcal{D})\) be a one-way trapdoor permutation on some set \(\mathbb{Z}_N\)
  - \(\mathbb{Z}_N = \{0, 1, 2, \ldots, N - 1\}\), operations *modulo* \(N\)
  - Something like public-key encryption scheme, but deterministic
  - \(\mathcal{E}\) is *permutation* on some set \(\mathbb{Z}_N\)
  - \(\mathcal{D} = \mathcal{E}^{-1}\)
  - Think “simple RSA”
  - Someone with access only to \(\mathcal{E}\) cannot distinguish it from a random permutation of \(\mathbb{Z}_N\)

- \(\mathcal{D}\) is available only to Alice
  - i.e. Alice has the private key

- There is also a hash function \(H\) that we model as a random oracle
  - Let its type be \(H : \mathbb{Z}_N \rightarrow \{0, 1\}^n\)
Random oracle

- $H : X \to Y$, where $Y$ is finite
- A box. Has input, output, internal state
- Internal state $S$: finite set of pairs in $X \times Y$
  - Initially empty
- On input $x_{\text{input}} \in X$:
  - Find the unique pair $(x, y) \in S$, where $x = x_{\text{input}}$
    - If no such pair: randomly pick $y \leftarrow Y$ and add $(x_{\text{input}}, y)$ to $S$
  - Output $y$
- Alternative specification:
  - All algorithms have access to an oracle $H : X \to Y$
  - This oracle is instantiated with a uniformly randomly chosen element of $X \to Y$
- (In simulation, $H$ is “run” by the simulator)
Protocol

1. Bob randomly picks $x \leftarrow \mathbb{Z}_N$

2. Bob sends $y := \mathcal{E}(x) - b$ to Alice

3. Alice computes $z_i := \mathcal{D}(y + i)$ for $i \in \{1, \ldots, \ell\}$
   
   ◦ Note that $z_b = x$

4. Alice picks random values $r_1, \ldots, r_a \leftarrow \{0, 1\}^n$

5. Alice sends to Bob

   $$(s_1, \ldots, s_\ell) := (r_1, \ldots, r_a, H(z_{a+1}), \ldots, H(z_\ell))$$

6. Bob checks if $s_b = H(x)$. If yes, then $a < b$

Exercise. Construct the simulators
Solution: privacy against Alice

- Input and output: $a$, the result bit $c = [a < b]$
  - Ability to invoke $D$ is also an input
- Alice also sees $y$. Simulator needs to construct it
Solution: privacy against Bob

- Input and output: $b$, the result bit $c = [a < b]$
- Bob also sees $x$, $s_1, \ldots, s_\ell$. Simulator needs to construct them
Private set intersection (PSI)

- There is a universal set $U$
- Alice has $S \subseteq U$. Bob has $T \subseteq U$
- Alice and Bob want to privately compute $S \cap T$
  - i.e. they both receive the elements of $S \cap T$
- Let the cardinalities $|S|$ and $|T|$ be public
Homomorphic encryption (HE)

- Public-key encryption, IND-CPA secure
- The plaintext space is a ring $R$ (think $\mathbb{Z}_N$)
- There are operations $\oplus$, $\odot$ on ciphertexts, such that

\[
\forall x, y \in R : \\
\mathcal{D}(\mathcal{E}(x) \oplus \mathcal{E}(y)) = x + y \\
\mathcal{D}(x \odot \mathcal{E}(y)) = x \cdot y
\]
Homomorphic encryption schemes

**Paillier cryptosystem**
- Plaintext space: $\mathbb{Z}_N$, where $N$ is a RSA modulus
- Ciphertexts are in $\mathbb{Z}_{N^2}$
- $\boxplus$ is multiplication
- $\Box$ is exponentiation
- I do not want to present the construction right now

Generalization:
Damgård-Jurik cryptosystem
- Plaintext space: $\mathbb{Z}_{N^s}$
- Ciphertext space: $\mathbb{Z}_{N^{s+1}}$

**ElGamal in exponents**
- $M$ encrypted as $(g^r, g^M \cdot h^r)$
- $\boxplus$ — componentwise multiplication
- $\Box$ — raise both sides to given power
- Decryption requires the computation of a discrete logarithm
- Works, if there are few possible plaintexts
A protocol for PSI

- \( S = \{s_1, \ldots, s_n\} \). \( T = \{t_1, \ldots, t_m\} \). \( S, T \subseteq \mathbb{Z}_N \)
- Alice can invoke \( D \)
- Alice finds the coefficients \( a_0, \ldots, a_n \in \mathbb{Z}_N \) of the polynomial

\[
P(x) = \sum_{i=0}^{n} a_i x^i = \prod_{j=1}^{n} (x - s_i)
\]

and sends \( E(a_0), \ldots, E(a_n) \) to Bob
- Bob generates random \( r_1, \ldots, r_m \leftarrow \mathbb{Z}_N \)
- Bob computes \( E(r_1 \cdot P(t_1)), \ldots, E(r_m \cdot P(t_m)) \) and sends them to Alice
- Alice tells Bob, which values are 0. Bob tells Alice the corresponding elements of \( T \)

**Exercise.** Construct the simulators
Another protocol for PSI

- $S = \{s_1, \ldots, s_n\}$. $T = \{t_1, \ldots, t_m\}$
- Let $\mathbb{G}$ be a group of size $p$ with hard DDH problem
- Let $H : U \to \mathbb{G}$ be a random oracle
- Alice generates $\alpha \leftarrow \mathbb{Z}_p$. Bob generates $\beta \leftarrow \mathbb{Z}_p$
- Alice sends $H(s_1)\alpha, \ldots, H(s_n)\alpha$ to Bob. Bob sends $H(t_1)\beta, \ldots, H(t_m)\beta$ to Alice
- Alice sends $(H(t_1)\beta)^\alpha, \ldots, (H(t_m)\beta)^\alpha$ to Bob. Bob sends $(H(s_1)\alpha)^\beta, \ldots, (H(s_n)\alpha)^\beta$ to Alice
- Both compare

Exercises. Why is Diffie-Hellman necessary? Why is $H$ necessary? Construct the simulators
PSI-CA

- Task: Alice has $S$. Bob has $T$. Securely compute $|S \cap T|$.
- Cannot just take any PSI protocol and take the cardinality of the computed intersection.
- The HE-based PSI protocol is easily convertible to a PSI-CA protocol.

**Exercise.** Do it. (including the simulators)
Millionaires from PSI-CA

Let \( a = a_n a_{n-1} \cdots a_1 \) and \( b = b_n b_{n-1} \cdots b_1 \) (binary representation of \( a \) and \( b \)).

For \( s = s_n s_{n-1} \cdots s_1 \) define sets of bit-strings of length \(< n\):

\[
S_s^0 = \{s_n s_{n-1} \cdots s_i + 1 | 1 \leq i \leq n, s_i = 0\}
\]

\[
S_s^1 = \{s_n s_{n-1} \cdots s_i + 1 | 1 \leq i \leq n, s_i = 1\}
\]

Claim. For any \( n \)-bit numbers \( a \) and \( b \), \( |S_s^0 \cap S_s^1| \in \{0, 1\}\)

Claim. If \( a < b \), then \( S_a^0 \cap S_b^1 \neq \emptyset\)

Exercises.

Prove the two previous claims

Construct a protocol for millionaires’ problem from any PSI-CA protocol (including the simulators)
Security definition (randomized $f$)

- Consider the randomized two-party, semi-honest case.
- A protocol $\Pi$ securely evaluates the randomized function $f(x_1, x_2)$ if there exist PPT simulators $S_1$ and $S_2$, such that for all $x_1$ and $x_2$: The following two distributions are indistinguishable (for $i \in \{1, 2\}$):
  1. First distribution:
     - Sample $(y_1, y_2) \leftarrow f(x_1, x_2)$;
     - Run $z \leftarrow S_i(x_i, y_i)$; consider the pair $(z, y_{3-i})$.
  2. Second distribution:
     - Sample $tr \leftarrow \Pi(x_1, x_2)$;
     - Take $(\text{VIEW}_i(tr), \text{RESULT}_{3-i}(tr))$.

Exercise. Why is the simulation output combined with the output of the other party? Compared to deterministic case, what else is there to protect?
Exercises

- Show that the secure evaluation of a randomized $f$ is reducible to the secure evaluation of some deterministic $f'$.
- Show that the secure evaluation of some $f$ with separate outputs for all parties is reducible to the secure evaluation of some $f$ with common output.
Example functionality

\[ f(x, y) = (x \equiv y, x < y) \] (common output).

For one-bit \( x \) and \( y \):
Example functionality

For 2k-bit $x$ and $y$:

\[
x_1 \ldots k \quad y_1 \ldots k \quad k-bit \quad x_{k+1} \ldots 2k \quad y_{k+1} \ldots 2k
\]
Example functionality

For 2k-bit x and y:

\[
\begin{align*}
& \oplus \\
& \& \\
\end{align*}
\]
Example functionality (4-bit $x \times \text{and} \ y$)
Evaluating a circuit

- Each internal gate $g$ is determined by the function it computes:

<table>
<thead>
<tr>
<th>Input</th>
<th>$g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00$</td>
<td>$g(00)$</td>
</tr>
<tr>
<td>$01$</td>
<td>$g(01)$</td>
</tr>
<tr>
<td>$10$</td>
<td>$g(10)$</td>
</tr>
<tr>
<td>$11$</td>
<td>$g(11)$</td>
</tr>
</tbody>
</table>

- The values of input gates are the bits of $x$ and $y$.
- The value of an internal gate is found from its inputs.
  - Computed from top to bottom.
- The value of an output gate is its input.
Garbling a circuit

- For each input and internal gate $g$, generate two (symmetric) encryption keys $k^0_g$ and $k^1_g$.
- Let $g$ be an internal gate. Let $g_1$ and $g_2$ provide the two inputs to $g$. Compute

$$\left\{ \{k^0_g(00)\} \right\} r^0_g \left\{ k^0_{g_2} \right\} r^1_g \left\{ k^0_{g_1} \right\}$$

$$\left\{ \{k^0_g(01)\} \right\} r^4_g \left\{ k^1_{g_2} \right\} r^3_g \left\{ k^0_{g_1} \right\}$$

$$\left\{ \{k^0_g(10)\} \right\} r^6_g \left\{ k^0_{g_2} \right\} r^5_g \left\{ k^1_{g_1} \right\}$$

$$\left\{ \{k^0_g(11)\} \right\} r^8_g \left\{ k^1_{g_2} \right\} r^7_g \left\{ k^1_{g_1} \right\}$$

- The encoding of an internal gate $g$ is a random permutation of these four values.
- Let $g$ provide the input to some output gate. The encoding of this output gate is $(\{0\}_{k^0_g}, \{1\}_{k^1_g})$ or $(\{1\}_{k^1_g}, \{0\}_{k^0_g})$.
- The encoding of a circuit maps each internal and output gate to its encoding.
Evaluating a garbled circuit

- Let the input gates be $g_1, \ldots, g_\ell$. Let the input be $b_1 \cdots b_\ell$.
- Somehow obtain $k_{g_1}^{b_1}, \ldots, k_{g_\ell}^{b_\ell}$. Do not obtain $k_{g_1}^{\neg b_1}, \ldots, k_{g_\ell}^{\neg b_\ell}$.
- Let $g$ be an internal gate.
  - Let $(m_1, m_2, m_3, m_4)$ be its encoding.
  - Let $g'$ and $g''$ provide the inputs to $g$.
  - Let us know a key $k'$ corresponding to $g'$.
  - Let us know a key $k''$ corresponding to $g''$.
  - Try to decrypt: compute $D_{k''}(D_{k'}(m_i))$ for $1 \leq i \leq 4$.
  - Let $k$ be the result of successful decryption. This key corresponds to $g$.
- At an output gate we can decrypt one of the ciphertexts, giving us either the bit 0 or the bit 1.
The protocol $\Pi$

- Both parties have agreed on the circuit that computes $f$.
- Party $P_1$ prepares the garbled circuit and sends it to $P_2$.
- $P_1$ sends to $P_2$ the keys $k_1^{x_1}, \ldots, k_\ell^{x_\ell}$ corresponding to its input $x$ going into the input gates $g_1, \ldots, g_\ell$.
- Party $P_1$ and $P_2$ run a protocol, resulting in
  - $P_2$ learning the keys $k_{\ell+1}^{y_1}, \ldots, k_{2\ell}^{y_\ell}$ corresponding to its input $y$ going into the input gates $g_{\ell+1}, \ldots, g_{2\ell}$;
  - $P_1$ not learning anything new at all.
- $P_2$ evaluates the garbled circuit, eventually learning $f(x, y)$.
- $P_2$ sends $f(x, y)$ back to $P_1$. 
Oblivious transfer

- A special case of two-party computation.
- $P_1$ (sender) has $\ell$-bit strings $m_1, \ldots, m_n$.
- $P_2$ (receiver) has an integer $i \in \{1, \ldots, n\}$.
- $P_2$ should learn $m_i$ and nothing else. $P_1$ should learn nothing.
Oblivious transfer

- A special case of two-party computation.
- $P_1$ (sender) has $\ell$-bit strings $m_1, \ldots, m_n$.
- $P_2$ (receiver) has an integer $i \in \{1, \ldots, n\}$.
- $P_2$ should learn $m_i$ and nothing else. $P_1$ should learn nothing.
- Let $E$ be a trapdoor permutation of some set $X \supseteq \{0, 1\}^\ell$ for example RSA.
- Let $(k_e, k_d)$ be the public and secret key of $P_1$.
- $P_2$ randomly chooses $r_1, \ldots, r_n \in X$. He defines
  \[
  z_j := \begin{cases} 
  r_j, & \text{if } j \neq i \\
  E_{k_e}(r_j), & \text{if } j = i 
  \end{cases}
  \]
  and sends $(z_1, \ldots, z_n)$ to $P_1$.
- $P_1$ computes $w_j := m_j \boxplus E_{k_d}^{-1}(z_j)$ and sends $(w_1, \ldots, w_n)$ to $P_2$.
  \(\boxplus\) — a group operation on $X$
- $P_2$ finds $m_i$ as $w_i \boxplus r_i^{-1}$. 

30.09.2021
Exercise

Show that the preceding protocol securely performs oblivious transfer in the presence of semi-honest adversaries. I.e. construct the simulators.
Exercise

Show that the preceeding protocol securely performs oblivious transfer in the presence of semi-honest adversaries. I.e. construct the simulators.

- This simulation is not quite correct, though.
- $E_{k_e}(r)$ reveals some information about $r$.
- But each trapdoor permutation has a hardcore bit $B$.
- If $m_1, \ldots, m_n$ are just 1 bit long, then the sender may define $w_j := m_j \oplus B(E_{k_d}^{-1}(z_j))$. The receiver recovers $m_i$ as $w_i \oplus B(r_i)$. 

30.09.2021
Exercise

Construct a 1-bit oblivious transfer protocol from private set intersection

Consider that only one of the parties gets the result of PSI. The other one gets nothing
Correctness of the protocol \( \Pi \)

- **Trace of the first party:**
  - Random coins for all the keys and encryptions.
  - Traces of the OT-protocol as the sender.
  - The final result.

- **Trace of the second party:**
  - A garbled circuit and the keys corresponding to the input bits.
  - Traces of the OT-protocol as the receiver.

**Exercise.** Construct a simulator for the first party.
Simulator for $P_2$

- For each input or internal gate $g$, generate two keys $k_g$, $\tilde{k}_g$.
- The simulated encoding of the gate $g \leftarrow g_1, g_2$ is
  \[
  \{\{k_g\}_{k_{g_2}}\}_{k_{g_1}}, \{\{k_g\}_{\tilde{k}_{g_2}}\}_{k_{g_1}}, \{\{k_g\}_{k_{g_2}}\}_{\tilde{k}_{g_1}}, \{\{k_g\}_{\tilde{k}_{g_2}}\}_{\tilde{k}_{g_1}}.
  \]
- The simulated encoding of an output gate $\leftarrow g$ is
  \[
  (\{z\}_{k_g}, \{z\}_{\tilde{k}_g}),
  \] where $z$ is the output bit corresponding to this gate.
- The simulated trace consists of
  - simulated garbled circuit;
  - the keys $k_1, \ldots, k_\ell$;
  - Simulated traces of the OT-protocol as the receiver, resulting in the keys $k_{\ell+1}, \ldots, k_{2\ell}$.
Example — comparing one bit

Let \( x_0 = 0, y_0 = 1 \). Then the output bits are \((0, 1)\). The real view is

\[
\{ k_1^0, k_1^1, \{ \{ k_3^1 \} k_2^0 \} r_2^1, \{ \{ k_3^0 \} k_1^0 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_1^0 \} k_1^1 \}, \{ \{ k_3^1 \} k_2^1 \} k_1^1 \},
\{ \{ k_0^0 \} k_2^0 \} r_9^1, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}
\]

The simulated view is

\[
k_1, k_2, \{ \{ k_3^1 \} k_2^0 \} r_2^1, \{ \{ k_3^0 \} k_2^1 \} r_1^1, \{ \{ k_3^0 \} k_1^0 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}, \{ \{ k_3^0 \} k_1^1 \}
\]

\[
\{ \{ k_0^0 \} k_2^0 \} r_9^1, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}, \{ \{ k_0^0 \} k_1^1 \}, \{ \{ k_1^1 \} k_1^1 \}
\]

\[
\{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}, \{ \{ 0 \} k_3^0 \}, \{ \{ 1 \} k_1^1 \}
\]
Decomposing a formal expression

Formal expressions $e ::= k \mid b \mid (e_1, e_2) \mid \{e\}'_k$

$e_1 ⊢ e_2$

The value of $e_1$ tells us the value of $e_2$
Decomposing a formal expression

Formal expressions \( e ::= k \mid b \mid (e_1, e_2) \mid \{e\}' \)

\[ e_1 \vdash e_2 \]

The value of \( e_1 \) tells us the value of \( e_2 \)

\[ e \vdash e \]

\[ e \vdash (e_1, e_2) \Rightarrow e \vdash e_1 \land e \vdash e_2 \]

\[ e \vdash \{e\}' \land e \vdash k \Rightarrow e \vdash e' \]
Decomposing a formal expression

Formal expressions $e ::= k \mid b \mid (e_1, e_2) \mid \{e\}'_k$

$$
e_1 \vdash e_2$$

The value of $e_1$ tells us the value of $e_2$

$$e \vdash e$$

$$e \vdash (e_1, e_2) \Rightarrow e \vdash e_1 \land e \vdash e_2$$

$$e \vdash \{e\}'_k \land e \vdash k \Rightarrow e \vdash e'$$

Examples:

$$
\begin{align*}
\{1011\}'_k, \{k_1\}'_k, k_2 \vdash 1011 \\
\{1011\}'_k, \{k_1\}'_k, \{k_2\}'_k, k_3 \nmid 1011 \\
\{1011\}'_k, \{k_1\}'_k, \{k_2\}'_k, k_3 \nmid 1011 \\
\end{align*}
$$
Decomposing a formal expression

Formal expressions $e ::= k \mid b \mid (e_1, e_2) \mid \{e'\}^r_k$

$$e_1 \vdash e_2$$

The value of $e_1$ tells us the value of $e_2$

$$e \vdash e$$

$$e \vdash (e_1, e_2) \Rightarrow e \vdash e_1 \land e \vdash e_2$$

$$e \vdash \{e'\}^r_k \land e \vdash k \Rightarrow e \vdash e'$$

Examples:

$$\{1011\}^r_{k_1}, \{k_1\}^r_{k_2}, k_2) \vdash 1011$$

$$\{1011\}^r_{k_1}, \{k_1\}^r_{k_2}, \{k_2\}^r_{k_3} \not\vdash 1011$$

$$\{1011\}^r_{k_1}, \{k_1\}^r_{k_2}, \{k_2\}^r_{k_3} \not\vdash 1011$$

Let $\text{openkeys}(e) = \{ k \in \text{Keys} \mid e \vdash k \}$. 
The **pattern** of a formal expression

- $e ::= \ldots | \square r$
- For a set $K \subseteq \text{Keys}$ define

  
  \[
  \begin{align*}
  \text{pat}(k, K) &= k \\
  \text{pat}(b, K) &= b \\
  \text{pat}((e_1, e_2), K) &= (\text{pat}(e_1, K), \text{pat}(e_2, K)) \\
  \text{pat}\{e\}_k^r, K) &= \begin{cases} 
  \{\text{pat}(e, K)\}_k^r, & \text{if } k \in K \\
  \square r, & \text{if } k \notin K
  \end{cases}
  \end{align*}
  \]

- Let $\text{pattern}(e) = \text{pat}(e, \text{openkeys}(e))$. 

The **pattern** of a formal expression

- \( e ::= \ldots | \Box^r \)
- For a set \( K \subseteq \textbf{Keys} \) define

\[
\begin{align*}
\text{pat}(k, K) &= k \\
\text{pat}(b, K) &= b \\
\text{pat}((e_1, e_2), K) &= (\text{pat}(e_1, K), \text{pat}(e_2, K)) \\
\text{pat}({e}^r_k, K) &= \begin{cases} 
\{\text{pat}(e, K)\}^r_k, & \text{if } k \in K \\
\Box^r, & \text{if } k \notin K
\end{cases}
\end{align*}
\]

- Let \( \text{pattern}(e) = \text{pat}(e, \text{openkeys}(e)) \).
- Define \( e_1 \simeq e_2 \) if \( \text{pattern}(e_1) = \text{pattern}(e_2)\sigma_K\sigma_R \) for some
  - \( \sigma_K \) — a permutation of the keys \textbf{Keys};
  - \( \sigma_R \) — a permutation of the random coins \textbf{Coins}. 

30.09.2021 35
Examples

\[
\text{pattern}\left( (\{1011\}_{k_1}^{r}, \{k_1\}_{k_2}^{r'}, k_2) \right) = (\{1011\}_{k_1}^{r}, \{k_1\}_{k_2}^{r'}, k_2) \\
\text{pattern}\left( (\{1011\}_{k_1}^{r}, \{k_1\}_{k_2}^{r'}, \{k_2\}_{k_3}^{r''}) \right) = (\square^r, \square^{r'}, \square^{r''}) \\
\text{pattern}\left( (\{1011\}_{k_1}^{r}, \{k_1\}_{k_2}^{r'}, \{k_2\}_{k_3}^{r''}) \right) = (\square^r, \square^{r'}, \square^{r''}) \\
\text{pattern}\left( (\{1\}_{k_2}^{r_1}, \{k_2\}_{k_3}^{r_2}, \{0\}_{k_2}^{r_4}, \{k_1\}_{k_1}^{r_3}) \right) = (\square^{r_1}, \square^{r_2}, \square^{r_4}, k_1) \\
\text{pattern}\left( (\{k_4, 0\}_{k_3}^{r_1}, \{k_3\}_{k_2}^{r_2}, \{11\}_{k_4}^{r_4}, \{k_1\}_{k_1}^{r_3}) \right) = (\square^{r_1}, \square^{r_2}, \square^{r_4}, k_1)
\]
The real pattern

The real view is

\[ k_1^0, k_2^1, \left\{ \left\{ k_3^1 \right\}_{k_2^0} r_2^1 \right\}_{k_1^0}, \left\{ \left\{ k_3^0 \right\}_{k_2^1} r_4^1 \right\}_{k_1^0}, \left\{ \left\{ k_3^0 \right\}_{k_2^0} r_6^1 \right\}_{k_1^1}, \left\{ \left\{ k_3^1 \right\}_{k_2^1} r_8^1 \right\}_{k_1^1}, \left\{ \left\{ k_4^0 \right\}_{k_2^0} r_{10}^1 \right\}_{k_1^0}, \left\{ \left\{ k_4^1 \right\}_{k_2^1} r_{12}^1 \right\}_{k_1^0}, \left\{ \left\{ k_4^0 \right\}_{k_2^0} r_{14}^1 \right\}_{k_1^1}, \left\{ \left\{ k_4^1 \right\}_{k_2^1} r_{16}^1 \right\}_{k_1^1}, \left\{ 0 \right\}_{k_3^0}, \left\{ 1 \right\}_{k_3^1}, \left\{ 0 \right\}_{k_4^0}, \left\{ 1 \right\}_{k_4^1} \right\}, \left\{ \left\{ 0 \right\}_{k_3^0}, \left\{ 1 \right\}_{k_3^1} \right\}, \left\{ \left\{ 0 \right\}_{k_4^0}, \left\{ 1 \right\}_{k_4^1} \right\} \]

Its pattern is

\[ k_1^0, k_2^1, \left\{ \left\{ \square \right\}_{k_1^0} r_2^1 \right\}_{k_1^0}, \left\{ \left\{ k_3^0 \right\}_{k_2^1} r_4^1 \right\}_{k_1^0}, \left\{ \left\{ k_3^0 \right\}_{k_2^0} r_6^1 \right\}_{k_1^1}, \left\{ \left\{ 0 \right\}_{k_3^0}, \left\{ \square \right\}_{k_3^1} \right\}, \left\{ \left\{ \square \right\}_{k_1^0} r_{10}^1 \right\}_{k_1^0}, \left\{ \left\{ k_4^1 \right\}_{k_2^1} r_{12}^1 \right\}_{k_1^0}, \left\{ \left\{ k_4^0 \right\}_{k_2^0} r_{14}^1 \right\}_{k_1^1}, \left\{ \left\{ 0 \right\}_{k_3^0}, \left\{ \square \right\}_{k_3^1} \right\}, \left\{ \left\{ 0 \right\}_{k_4^0}, \left\{ \square \right\}_{k_4^1} \right\}, \left\{ \left\{ 0 \right\}_{k_3^0}, \left\{ \square \right\}_{k_3^1} \right\}, \left\{ \left\{ k_4^1 \right\}_{k_3^0} r_{17}^1 \right\}_{k_3^0}, \left\{ 0 \right\}_{k_3^0}, \left\{ \square \right\}_{k_3^1} \right\}, \left\{ \left\{ 0 \right\}_{k_4^0}, \left\{ \square \right\}_{k_4^1} \right\}, \left\{ \left\{ 0 \right\}_{k_4^0}, \left\{ \square \right\}_{k_4^1} \right\} \]
The simulated pattern

The simulated view is

\[ k_1, k_2, \{ \{ \{ k_3 \}_2 \}_2 \}_2, \{ \{ k_3 \}_4 \}_2, \{ \{ k_3 \}_6 \}_2, \{ \{ k_3 \}_8 \}_2 \}, \]
\[ \{ \{ \{ k_4 \}_2 \}_2 \}_2, \{ \{ k_4 \}_4 \}_2, \{ \{ k_4 \}_6 \}_2, \{ \{ k_4 \}_8 \}_2, \{ \{ k_4 \}_10 \}_2 \}, \]
\[ \{ 0 \}_2, \{ 0 \}_2, \{ 1 \}_2, \{ 1 \}_2 \]

Its pattern is

\[ k_1, k_2, \{ \{ \{ k_3 \}_2 \}_2 \}_2, \{ \{ k_3 \}_4 \}_2, \{ \{ k_3 \}_6 \}_2, \{ \{ k_3 \}_8 \}_2 \}, \]
\[ \{ \{ \{ k_4 \}_2 \}_2 \}_2, \{ \{ k_4 \}_4 \}_2, \{ \{ k_4 \}_6 \}_2, \{ \{ k_4 \}_8 \}_2, \{ \{ k_4 \}_10 \}_2 \}, \]
\[ \{ 0 \}_2, \{ 0 \}_2, \{ 1 \}_2, \{ 1 \}_2 \]
Compare the patterns

The real pattern

\[ k_0^1, k_2^1, \{\{\square r_2\}^r_1 k_0^1, \{\{k_3^0\}^r_4 k_2^1 \}^r_3 \}, \square r_5, \square r_7 \}, \]
\[ \{\{\square r_{10}\}^r_9 k_0^1, \{\{k_4^1\}^r_{12} k_2^1 \}^r_{11}, \square r_{13}, \square r_{15} \}, \{\{0\}^r_{17}, \square r_{18} \}, \{\square r_{19}, \{1\}^r_{20} k_4^1 \} \]

The simulated pattern

\[ k_1, k_2, \{\{k_3^2 \}^r_{k_2} k_1, \{\square r_{4} \}^r_3 k_1, \square r_5, \square r_7 \}, \]
\[ \{\{k_4 \}^r_{10} k_1, \{\square r_{12} \}^r_{11} k_1, \square r_{13}, \square r_{15} \}, \{\{0\}^r_{17}, \square r_{18} \}, \{\{1\}^r_{19} k_4, \square r_{20} \} \]

They are equal up to renaming.

In general, the real and simulated garbled circuits will be indistinguishable.

30.09.2021 39
A different kind of OT

- Sender has a message $m$. Receiver gets it with probability 50%. Receiver knows whether he got it, sender will not know.

- Rabin’s construction:
  - Sender generates RSA modulus $n$ and picks $e \in \mathbb{Z}_{\varphi(n)}^*$. Sends $(n, e)$ to receiver.
  - Receiver picks $x \in \mathbb{Z}_n$, sends $y = x^2 \mod n$ to sender.
  - Sender sends a square root of $y$ (in $\mathbb{Z}_n$) to receiver. Also sends $m^e \mod n$ to receiver.

Exercise. This OT and 1-out-of-2 OT can be constructed from each other.
Bellare-Micali 1-out-of-2 OT

- Sender randomly picks $C \in G$ and sends it to the receiver.
- Receiver chooses $x \in \mathbb{Z}_p$ and defines $h_b = g^x$, $h_{1-b} = C/h_b$. Sends $h_0, h_1$ to the sender.
- Sender checks that $h_0 h_1 = C$. Uses ElGamal encryption to encrypt $m_i$ with $h_i$. Sends

$$(g^{r_0}, m_0 \cdot h_0^{r_0}), (g^{r_1}, m_1 \cdot h_1^{r_1})$$

- Receiver decrypts the ciphertext that he can decrypt and learns $m_b$.

Exercise. Security?
Simulating receiver’s view (1/4)

- Simulator’s inputs: bit \( b \), message \( M_b \), generator \( g \in G \)
- Simulator randomly samples \( C \in G, \ x \in \mathbb{Z}_p, \ r_b, r_{1-b} \in \mathbb{Z}_p \)
- Outputs

\[
x, h_b = g^x, h_{1-b} = C/h_b \quad (g^{r_b}, M_b \cdot h_b^{r_b}), (g^{r_{1-b}}, h_{1-b}^{r_{1-b}})
\]

- The real view is

\[
x, h_b = g^x, h_{1-b} = C/h_b \quad (g^{r_b}, M_b \cdot h_b^{r_b}), (g^{r_{1-b}}, M_{1-b} \cdot h_{1-b}^{r_{1-b}})
\]

- These have to be indistinguishable for all \( b, M_b, M_{1-b} \)
Simulating receiver’s view (2/4)

- Simulator’s inputs: bit \( b \), message \( M_b \), generator \( g \in G \)
- Simulator randomly samples \( c \in \mathbb{Z}_p, x \in \mathbb{Z}_p, r_b, r_{1-b} \in \mathbb{Z}_p \)
- Outputs

\[
x, h_b = g^x, h_{1-b} = g^{c-x} (g^{r_b}, M_b \cdot g^{x \cdot r_b}), (g^{r_{1-b}}, g^{(c-x) \cdot r_{1-b}})
\]

- The real view is

\[
x, h_b = g^x, h_{1-b} = g^{c-x} (g^{r_b}, M_b \cdot g^{x \cdot r_b}), (g^{r_{1-b}}, M_{1-b} \cdot g^{(c-x) \cdot r_{1-b}})
\]

- These have to be indistinguishable for all \( b, M_b, M_{1-b} \). If there existed \( b, M_0, M_1 \), such that they were distinguishable, then...
Simulating receiver’s view (3/4)

- We are given \((g, g_1, g_2, g_3)\). We want to find \(B\), where
  \[ B = \log_g g_3 = \log_g g_1 \cdot \log_g g_2 \]
- Consider \(g_1 \equiv g^c\) and \(g_2 \equiv g^{r_1-b}\)
- Generate \(x, r_b \leftarrow \mathbb{Z}_p\). Output
  \[ x, g^x, g_1/g^x, (g^{r_b}, M_b \cdot g^{x \cdot r_b}), (g_2, g_3/g_2^x) \]

- If \(B\), then this is the simulated view. If \(\neg B\), then this is
  \[ x, h_b = g^x, h_{1-b} = g^{c-x} \quad (g^{r_b}, M_b \cdot g^{x \cdot r_b}), (g^{r_{1-b}}, \text{rnd}) \]

Let us call it the “random view”
Simulating receiver’s view (4/4)

- We are given \((g, g_1, g_2, g_3)\). We want to find \(B\), where...
- Consider \(g_1 \equiv g^c\) and \(g_2 \equiv g^{r_1-b}\)
- Generate \(x, r_b \leftarrow \mathbb{Z}_p\). Output
  
  \[
  x, g^x, g_1/g^x, (g^{rb}, M_b \cdot g^{x \cdot rb}), (g_2, M_{1-b} \cdot g_3/g_2^x)
  \]

- If \(B\), then this is the real view. If \(\neg B\), then this is still
  
  \[
  x, h_b = g^x, h_{1-b} = g^{c-x} \quad (g^{rb}, M_b \cdot g^{x \cdot rb}), (g^{r_1-b}, \text{rnd})
  \]

- Hence real view \(\approx\) “random view” \(\approx\) simulated view
Naor-Pinkas construction

Let $G, g, p$ be as before.

Receiver picks $s, t, c_{1-b} \in R \mathbb{Z}_p$, defines $c_b = st \mod p$, $x = g^s$, $y = g^t$, $z_i = g^{c_i}$. Sends $x, y, z_0, z_1$ to sender.

Sender checks that $z_0 \neq z_1$. Picks random $r_0, r_0', r_1, r_1' \in \mathbb{Z}_p^*$ and returns to the receiver

$$(((xg^{r_0})^{r_0'}, m_0 \cdot (z_0y^{r_0})^{r_0'}), (((xg^{r_1})^{r_1'}, m_1 \cdot (z_1y^{r_1})^{r_1'}))$$

The receiver.

**Exercise.** What is the receiver going to do with the values it got? What about security?

**Exercise.** Generalize this construction to 1-out-of-n OT.
Consider a family of trapdoor permutations (e.g. RSA). Let \( \mathcal{P} \) be the set of plaintexts and ciphertexts. \( \mathcal{P} \) must be equipped with a group operation “\( \cdot \)”.

Sender generates keypair \( (k_{\text{pub}}, k_{\text{priv}}) \) and picks two elements \( x_0, x_1 \in \mathcal{P} \). Sends \( k_{\text{pub}}, x_0, x_1 \) to receiver.

Receiver picks a plaintext \( r \in \mathcal{P} \), sends \( y = E_{k_{\text{pub}}}(r) \cdot x_b \) to sender.

Sender sends \( D_{k_{\text{priv}}}(y/x_0) \cdot m_0 \) and \( D_{k_{\text{priv}}}(y/x_1) \cdot m_1 \) to receiver.
Chou-Orlandi 1-out-of-$n$ OT

Let $G$, $g$, $p$ be as before.

- Sender has $M_1, \ldots, M_n$. Receiver has $c \in \{1, \ldots, n\}$
- Sender picks $y \leftarrow \mathbb{Z}_p$. Defines $s = g^y$ and $t = s^y = g^{y^2}$
  - May be reused
- Sender sends $s$ to receiver
- Receiver picks $x \leftarrow \mathbb{Z}_p$. Sends $r = s^c g^x$ to sender
- $\forall j \in \{1, \ldots, n\}$: Define $k_j = H(s, r; r^y t^{-j})$
  - $H$ is modelled as a random oracle
  - **Exercise.** How does receiver compute $k_c$?
- Sender sends $\left(\{M_j\}^{k_j}\right)_{j=1}^n$
  - encryption $\approx$ one-time pad. Simulator may need this

Security based on computational Diffie-Hellman (**Exercise**)
OT from homomorphic encryption

- Homomorphic encryption — public-key encryption \((K, E, D)\), where
- The set of plaintexts is a ring
- There is an additional operation \(\boxplus\) on ciphertexts, such that

\[
\begin{align*}
D_{k_{\text{priv}}} (E_{k_{\text{pub}}} (m_1) \boxplus E_{k_{\text{pub}}} (m_2)) &= m_1 + m_2 \\
D_{k_{\text{priv}}} (m_1 \boxdot E_{k_{\text{pub}}} (m_2)) &= m_1 \cdot m_2
\end{align*}
\]

- We’re going to see examples in this course
- 1-out-of-2 oblivious transfer:
  - Receiver generates \((k_{\text{pub}}, k_{\text{priv}})\), sends \(k_{\text{pub}}\) and \(c = E_{k_{\text{pub}}} (b)\) to Sender
  - Sender responds with \(E_{k_{\text{pub}}} (m_0) \boxplus (m_1 - m_0) \boxdot c\)
Optimizations of garbled circuits
Point-and-permute

- For each input and internal gate $g$, generate two (symmetric) encryption keys $k^0_g$ and $k^1_g$.
- Let $p_g = \text{lsb}(k^0_g)$ and $\bar{p}_g = \text{lsb}(k^1_g)$. Make it so that $p_g \neq \bar{p}_g$
- $g$: internal gate. $g_1$ and $g_2$: the two inputs to $g$. Compute

\[
\begin{align*}
&\{\{k^g_{g_2}(00)\}_k^0\}_k^0_{g_1}, \{\{k^g_{g_2}(01)\}_k^1\}_k^0_{g_1}, , \{\{k^g_{g_2}(10)\}_k^0\}_k^1_{g_1}, \{\{k^g_{g_2}(11)\}_k^1\}_k^1_{g_1}\}
\end{align*}
\]

- Represent $g$ by ordering these four ciphertexts by the lsb-s of the two keys used to encrypt each of them

\[
g \mapsto \left[\begin{array}{c}
\{\{k^g(p_{g_1}p_{g_2})\}_k^{p_{g_2}}\}_k^{p_{g_1}}_{g_1}, \{\{k^g(p_{g_1}\bar{p}_{g_2})\}_k^{p_{g_2}}\}_k^{p_{g_1}}_{g_1}

\{\{k^g(\bar{p}_{g_1}p_{g_2})\}_k^{p_{g_2}}\}_k^{\bar{p}_{g_1}}_{g_1}, \{\{k^g(\bar{p}_{g_1}\bar{p}_{g_2})\}_k^{p_{g_2}}\}_k^{\bar{p}_{g_1}}_{g_1}
\end{array}\right]
\]

- Result: evaluator has to consider a single ciphertext per gate

30.09.2021 51
Redundancy is no longer necessary

\[
g \mapsto \begin{bmatrix}
\mathcal{E}(k_{g_1}^{p_{g_1}}, \mathcal{E}(k_{g_2}^{p_{g_2}}, k_g^{g(p_{g_1}p_{g_2})})) & \mathcal{E}(k_{g_1}^{p_{g_1}}, \mathcal{E}(k_{g_2}^{p_{g_2}}, k_g^{g(p_{g_1}\bar{p}_{g_2})})) \\
\mathcal{E}(\bar{p}_{g_1}, \mathcal{E}(k_{g_2}^{p_{g_2}}, k_g^{g(\bar{p}_{g_1}p_{g_2})})) & \mathcal{E}(\bar{p}_{g_1}, \mathcal{E}(k_{g_2}^{p_{g_2}}, k_g^{g(\bar{p}_{g_1}\bar{p}_{g_2})}))
\end{bmatrix}
\]

\(\mathcal{E}\) is a block cipher
Garbled row reduction (GRR)

\[ g \mapsto \begin{bmatrix}
\mathcal{E}(k_{g_1}^{p_{g_1}}, \mathcal{E}(k_{g_2}^{p_{g_2}}, k_g^{(p_{g_1} p_{g_2})})) & \mathcal{E}(k_{g_1}^{p_{g_1}}, \mathcal{E}(k_{g_2}^{\bar{p}_{g_2}}, k_g^{(p_{g_1} \bar{p}_{g_2})})) \\
\mathcal{E}(k_{\bar{g}_1}^{\bar{p}_{g_1}}, \mathcal{E}(k_{g_2}^{p_{g_2}}, k_g^{(\bar{p}_{g_1} p_{g_2})})) & \mathcal{E}(k_{\bar{g}_1}^{\bar{p}_{g_1}}, \mathcal{E}(k_{g_2}^{\bar{p}_{g_2}}, k_g^{(\bar{p}_{g_1} \bar{p}_{g_2})}))
\end{bmatrix} \]
Garbled row reduction (GRR)

\[ g \mapsto \begin{bmatrix} 0 \\ \mathcal{E}(k_{g_1}^{p_{g_1}}, \mathcal{E}(k_{g_2}^{p_{g_2}}, k_g^{(p_{g_1}p_{g_2})})) \\ \mathcal{E}(k_{g_1}^{\bar{p}_{g_1}}, \mathcal{E}(k_{g_2}^{p_{g_2}}, k_g^{(\bar{p}_{g_1}p_{g_2})})) \\ \mathcal{E}(k_{g_1}^{\bar{p}_{g_1}}, \mathcal{E}(k_{g_2}^{\bar{p}_{g_2}}, k_g^{(\bar{p}_{g_1}\bar{p}_{g_2})})) \end{bmatrix} \]

- Given \( p_{g_1}, p_{g_2}, k_{g_1}^{p_{g_1}}, k_{g_1}^{\bar{p}_{g_1}}, k_{g_2}^{p_{g_2}}, k_{g_2}^{\bar{p}_{g_2}} \), garble \( g \) as follows:
  - \( k_g^{g(p_{g_1}p_{g_2})} := \mathcal{D}(k_{g_2}^{p_{g_2}}, \mathcal{D}(k_{g_1}^{p_{g_1}}, 0)) \)
  - \( p_g := \text{lsb}(k_g^{g(p_{g_1}p_{g_2})}) \oplus g(p_{g_1}p_{g_2}) \)
  - Generate \( k_g^{\neg g(p_{g_1}p_{g_2})} \) with the correct last bit
  - Compute the three remaining ciphertexts in the representation of \( g \)
  - Output \( p_g, k_g^0, k_g^1 \) and the three ciphertexts

Result: Each gate is encoded by only 3 ciphertexts
Free-XOR

- The garbler generates a secret bit-string $\Delta$

- For each input and internal gate $g$, generate two (symmetric) encryption keys $k^0_g$ and $k^1_g$, such that $k^0_g \oplus k^1_g = \Delta$.

- But if $g$ is a XOR-gate or “=”-gate then define $k^0_g = k^0_{g_1} \oplus k^0_{g_2}$
  - Then also $k^0_g = k^1_{g_1} \oplus k^1_{g_2}$
  - And $k^1_g = k^0_{g_1} \oplus \Delta = k^0_{g_1} \oplus k^1_{g_2} = k^1_{g_1} \oplus k^0_{g_2}$

- Result:
  - The representation of a XOR- or “=”-gate is empty
  - At such gate, the evaluator XOR-s the two keys that he has for the inputs

- Free-XOR and GRR can be used at the same time

**Exercise.** Represent a 1-bit full adder with 1 non-XOR gate
AND-gate, garbler knows an input

- \( c = a \land b \). Garbler knows the bit \( a \).
- Garbler has the bit \( a \). Evaluator has the either \( k_b^0 \) or \( k_b^1 \).
- Garbled gate \( c \) has a single input from the gate \( b \).
  - Garbler garbles either the constant “0” gate or the identity gate.

\[
E(k_p^b, k_c^0) \quad \text{or} \quad E(k_p^b, k_c^0) \mapsto \begin{bmatrix} E(k_p^b, k_c^0) \\ E(k_p^b, k_c^0) \end{bmatrix}
\]

- GRR removes one ciphertext
- This is compatible with Free-XOR
AND-gate, evaluator knows an input

1. \( c = a \land b \). Evaluator knows the bit \( a \). Let Free-XOR be in use
2. Evaluator has the key \( k_a^v \), he knows \( v \). Evaluator has \( k_b^u \), where \( u \) is either 0 or 1.
3. Garbler encodes the gate as (use GRR, too)

\[
c \mapsto \begin{cases} 
\mathcal{E}(k_a^0, k_c^0) \\
\mathcal{E}(k_a^1, k_c^0 \oplus k_b^0)
\end{cases}
\]

4. Evaluation:
   1. If \( v = 0 \), get \( k_c^0 \) from the first ciphertext
   2. If \( v = 1 \), decrypt second ciphertext, and XOR the result with \( k_b^u \)

\[
k_c^0 \oplus k_b^0 \oplus k_b^u = k_c^0 \oplus u \Delta = k_c^u
\]
Half-gates

- “Half-gate” ≡ a gate where one party knows one of the inputs
- We have the gate $a \land b$
- For any $r \in \{0, 1\}$: $a \land b = (a \land r) \oplus (a \land (b \oplus r))$
- The garbler picks $r = p_b$
- Garbler knows $p_b$. Evaluator knows $b \oplus p_b$
- Garbler encodes $a \land r$ and $a \land (b \oplus r)$. Each requires a single ciphertext
- Result: Each gate is encoded by only 2 ciphertexts. Free-XOR compatible

Can still go lower, [https://ia.cr/2021/749](https://ia.cr/2021/749) gives 1.5 ciphertexts and 5 bits / gate
Garbling arithmetic circuits

Mod 2, with Free-XOR and point-and-permute

- Gate $g$: $k_g^0$ is a sequence of values mod 2
- $\Delta$: sequence of values mod 2, last of them is 1
  - Define $k_g^1$ as componentwise sum (mod 2) of $k_g^0$ and $\Delta$

Mod $m$

- Gate $g$: $k_g^0$ is a sequence of values mod $m$
  - Somehow encode as a bit-string
- $\Delta_m$: sequence of values mod $m$, last of them is 1
  - Define $k_g^\nu$ as componentwise sum (mod $m$) of $k_g^0$ and $\nu \cdot \Delta$
    - $\nu \in \{1, \ldots, m - 1\}$

- A circuit can use several $m$-s
Arithmetic circuits: supported gates

**Basics**
- Binary, non-linear gates
  - 2 ciphertexts, if mod 2. Much more for larger moduli
- Addition and mult. with constant (co-prime with \( m \)) — free
- “Projections”: any functions \( \varphi : \mathbb{Z}_m \rightarrow \mathbb{Z}_{m'} \). \((m - 1)\) ciphertexts

**Derived**
- Multiplication: discrete log + addition + handle 0-s
- Large moduli: use Chinese Remainder Theorem
- Threshold gates: larger modulus + addition + projection
- Etc. etc.
Confidentiality proofs of it all?

Confidentiality against evaluator (the only interesting property)

- Environment selects a bit $b$
- Adversary chooses function $f$ (as a circuit $C$), inputs $x_0, x_1, y$, such that $f(x_0, y) = f(x_1, y)$
- Environment garbles $C$, adversary gets the garbled circuit, and keys for $y$ and for $x_b$
- Adversary tries to guess $b$

Security of the block cipher

- We call it in both directions ($\mathcal{E}$ and $\mathcal{D}$)
- We use many different keys
  - ... and we compute one key from another one (using $\oplus$ and $\mathcal{D}$)
  - ... and “$\oplus \Delta$” is used with both keys and plain-/ciphertexts
Implementation of the block cipher

- Let $\mathcal{H}$ be a hash function
  - Arbitrary input length
  - Outputs have the same length as the keys we use
- Implement $\mathcal{E}(k_1, \mathcal{E}(k_2, M))$ at gate $g$ as
  $$\mathcal{E}(k_1, \mathcal{E}(k_2, M)) = \mathcal{H}(k_1 \Vert k_2 \Vert ID_g) \oplus M$$

- Decryption — same. Using just a single key — similar
- If $\mathcal{H}$ is a random oracle, then the garbling techniques are secure
Correlation robustness of hash functions

The “real” oracle $\text{Cor}_{\mathcal{H}}$
- When first started, generate a random bit-string $\Delta$
- On input $k, k', id$, output

$$\mathcal{H}(k || k' \oplus \Delta || id), \mathcal{H}(k \oplus \Delta || k' || id), \mathcal{H}(k \oplus \Delta || k' \oplus \Delta || id)$$

The “ideal” oracle $\text{Rand}$
- On input $k, k', id$, output three random values
- Store the input-output pairs, and give the same response to a repeating query

A hash function $\mathcal{H}$ is 2-correlation robust, if $\text{Cor}_{\mathcal{H}} \approx \text{Rand}$
Correlation robustness is not enough for black-box security of Free-XOR

What do we mean with it?

Security of black-box constr.s “primitive X ⇒ primitive Y”

- C is a PPT algorithm with oracle access to H
- Want to prove: “H is a secure primitive X ⇒ C^H is a secure primitive Y”
- Present a PPT algorithm S (making oracle queries) and argue that ∀A:
  - “A breaks C^H as Y” ⇒ “S^A breaks H as X”
- We do not say anything about the power of A...
  - It is a program, executing some operations
  - These operations could do anything...
    - E.g. solve the halting problem...
- We pick some operations and define H so, that H is a secure X, but C^H is not a secure Y
The extra operations

- Syntax:
  - $H : \{0, 1\}^* \rightarrow \{0, 1\}^{\text{key length}}$
  - $\text{Break}$ takes five keys and an ID. Returns a key or ⊥

- Define $\mathcal{H} = H$

- Define the semantics of $H$ and $\text{Break}$ probabilistically:
  - $H$ is a random function
  - $\text{Break}(k, k', id, z_1, z_2, z_3) = R$, where

$$
    z_1 = H(k \| k' \oplus R \| id)
$$

$$
    z_2 = H(k \oplus R \| k' \| id)
$$

$$
    z_3 = H(k \oplus R \| k' \oplus R \| id) \oplus R
$$

(the lexicographically first such $R$. Or ⊥, if there is no $R$)
2-correlation robustness of $\mathcal{H}$

- An adversary has to distinguish $\text{Cor}_\mathcal{H}$ from Rand
  - Suppose it does not repeat queries to the oracle
- It can (also) use the operations $H$ and $\text{Break}$
- Rand outputs uniformly random values
- $\text{Cor}_\mathcal{H}$ also outputs uniformly random values, unless
  - Adversary manages to repeat an argument to $H$
    - Either through $\text{Cor}_\mathcal{H}$ or directly
    - That is equivalent to guessing $\Delta$
  - Adversary calls $\text{Break}(k, k', \text{id}, z_1, z_2, z_3)$, such that $\Delta$ is the answer
    - This is equivalent to randomly picking $\Delta$ from some non-uniform distribution
Insecurity of Free-XOR in this context

- Consider a garbled AND-gate with Free-XOR and GRR
  - GRR has removed the row corresponding to inputs 0, 0
- Let the evaluator have keys for the inputs 0, 0 to this gate
- Then the \textit{Break}-operation gives him the value $\Delta$
Circular correlation robustness

The “real” oracle $\text{CorCirc}_H$

- When first started, generate a random bit-string $\Delta$
- On input $k, k', id, b_1, b_2, b_3$ (the latter are bits), output

$$\mathcal{H}(k \oplus b_1 \Delta \parallel k' \oplus b_2 \Delta \parallel id) \oplus b_3 \Delta$$

- The adversary may not query $(k, k', id, 0, 0, b_3)$
- The adversary may not query both $(k, k', id, b_1, b_2, 0)$ and $(k, k', id, b_1, b_2, 1)$

The ideal oracle again implements a random function