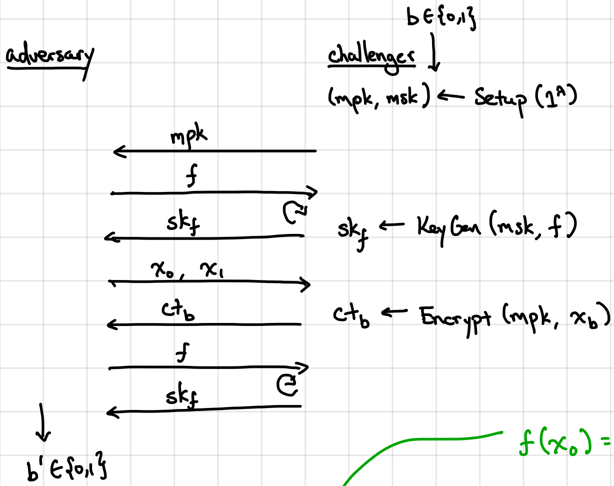
Functional encryption: generalization of attribute-based encryption and predicate encryption
↳ decryption reveals a <u>function</u> of the message

Setup $(1^\lambda) \to (mpk, msk)$
Encrypt $(mpk, x) \to ct_x$
KeyGen $(msk, f) \to sk_f$
Decrypt $(sk_f, ct_x) \to f(x)$

<u>Correctness</u>:
if $(mpk, msk) \leftarrow \text{KeyGen}(1^\lambda)$
$ct_x \leftarrow \text{Encrypt}(mpk, x)$
$sk_f \leftarrow \text{KeyGen}(msk, f)$
then $\text{Decrypt}(sk_f, ct_x) = f(x)$

<u>Security</u>:




$f(x_0) = f(x_1)$ for all functions $f$
$A$ submits to key-generation oracle

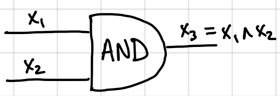
Secure if for all efficient and admissible adversaries $A$:
$$|\Pr[b'=1 \mid b=0] - \Pr[b'=1 \mid b=1]| = \text{negl}$$
Need to be careful with definition (for some classes of functions, a "trivially broken" scheme might satisfy this definition)
↳ But this is still a reasonable definition for a broad range of settings
↳ Can strengthen definition to simulation-based definition – many impossibilities in this setting
FE is a very powerful primitive — some flavors imply obfuscation

<u>Today</u>: consider a simple setting where we only need PKE

<u>Single-key FE</u>: adversary can only see a <u>single</u> key for the FE scheme

<u>Main building block</u>: garbled circuit (more generally: randomized encoding)
↳ common tool in cryptography, core building block for secure computation

Key ingredient: "garbling" protocol (garbled circuits)

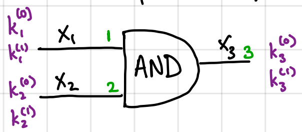



$\Longrightarrow$

truth table:

| $x_1$ | $x_2$ | $x_1 \wedge x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

1) Associate a pair of keys $(k_i^{(0)}, k_i^{(1)})$ with each wire $i$ in the circuit




$k_i^{(b)}$: key associated with wire value $b$ for wire $i$ [symmetric encryption key]

2) Prepare garbled truth table for the gate

↳ Replace each entry of truth table with corresponding key

↳ Encrypt output key with each of the input keys

| $x_1$ | | $x_2$ | | $x_3 = x_1 \wedge x_2$ | |
|---|---|---|---|---|---|
| 0 | $k_1^{(0)}$ | 0 | $k_2^{(0)}$ | 0 | $k_3^{(0)}$ |
| 0 | $k_1^{(0)}$ | 1 | $k_2^{(1)}$ | 0 | $k_3^{(0)}$ |
| 1 | $k_1^{(1)}$ | 0 | $k_2^{(0)}$ | 0 | $k_3^{(0)}$ |
| 1 | $k_1^{(1)}$ | 1 | $k_2^{(1)}$ | 1 | $k_3^{(1)}$ |

$\Longrightarrow$

$$\left.\begin{aligned}
ct_{00} &\leftarrow \mathrm{Encrypt}(k_1^{(0)}, \mathrm{Encrypt}(k_2^{(0)}, k_3^{(0)})) \\
ct_{01} &\leftarrow \mathrm{Encrypt}(k_1^{(0)}, \mathrm{Encrypt}(k_2^{(1)}, k_3^{(0)})) \\
ct_{10} &\leftarrow \mathrm{Encrypt}(k_1^{(1)}, \mathrm{Encrypt}(k_2^{(0)}, k_3^{(0)})) \\
ct_{11} &\leftarrow \mathrm{Encrypt}(k_1^{(1)}, \mathrm{Encrypt}(k_2^{(1)}, k_3^{(1)}))
\end{aligned}\right\} \text{randomly shuffle ciphertexts}$$

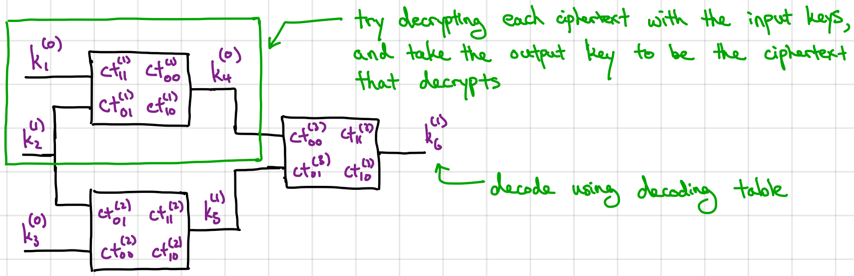
3) Construct decoding table for output values

$$\left.\begin{aligned}
k_3^{(0)} &\mapsto 0 \\
k_3^{(1)} &\mapsto 1
\end{aligned}\right\}$$

Alternatively, can just encrypt output values instead of keys for output wires

General garbling transformation: construct garbled table for each gate in the circuit, prepare decoding table for each output wire in the circuit

Evaluating a garbled circuit:




Invariant: given keys for input wires of a gate, can derive key corresponding to output wire $\Longrightarrow$ enables gate-by-gate evaluation of garbled circuit

↳ Requirement: Evaluator needs to obtain keys (labels) for its inputs (but without revealing which set of labels it requested)

Abstractly: $\mathrm{Garble}(1^\lambda, C) \to (\tilde{C}, \{L_i^{(b)}\}_{i \in [n], b \in \{0,1\}})$ — number of input wires

$\mathrm{Eval}(\tilde{C}, \{L_i^{(b)}\}_{i \in [n]}) \to y$

- Correctness: For all circuits $C : \{0,1\}^n \to \{0,1\}^m$ and all $x \in \{0,1\}^n$:

if $(\tilde{C}, \{L_i^{(b)}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \mathrm{Garble}(1^\lambda, C)$,

$$\Pr[\mathrm{Eval}(\tilde{C}, \{L_i^{(x_i)}\}_{i \in [n]}) = C(x)] = 1$$

Lots of optimizations!

Free XOR: no need to provide garbled truth table for xor gates

Half-Gates: only need 2 ciphertexts (instead of 4) for each AND gate

More recent: only need 1.5 ciphertexts per AND gate! (i.e. $|L_i^{(b)}| = 1.5\lambda + 5$ bits) [RR21]

- <u>Security</u>: There exists an efficient simulator $S$ such that for all circuits $C: \{0,1\}^n \to \{0,1\}^m$ and $x \in \{0,1\}^n$:

  for $(\tilde{C}, \{L_i^{(b)}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \text{Garble}(1^\lambda, C)$:

$$\{(\tilde{C}, \{L_i^{(x_i)}\}_{i \in [n]})\} \overset{c}{\approx} S(1^\lambda, C, C(x))$$

  ↖ can also consider notion where only $|C|$ is provided to $S$

Namely, the garbled circuit and <u>one set</u> of labels can be simulated just given the output $C(x)$.

Using garbled circuits for two-party computation:

→ this is necessary and sufficient for <u>general</u> multiparty computation (MPC)!

<u>Key cryptographic building block</u>: oblivious transfer (OT)

sender $(m_0, m_1)$      receiver $(b \in \{0,1\})$

←

→    ↓ $m_b$

sender has two messages $m_0, m_1$

receiver has a bit $b \in \{0,1\}$

at the end of the protocol, receiver learns $m_b$, sender learns nothing

<u>Yao's garbled circuit protocol</u>:

circuit $C$

Alice (garbler)      Bob (evaluator)

↳ private input $x$      ↳ private input $y$

Bob: 1. Prepare OT queries for bits of $y$

Alice: 1. Prepare garbled circuit for $C$

← OT for labels for $y$

Alice: 2. Prepare OT responses for Bob's inputs. Messages correspond to wire labels.

→ OT responses for labels of Bob's input

→ garbled circuit

→ labels for Alice's input $x$

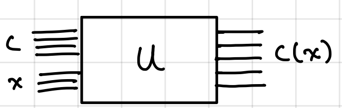
Bob: 2. Evaluate garbled circuit to learn $C(x, y)$

single-key restriction needed because garbled circuit is <u>not</u> reusable

Two-party computation protocol is <u>interactive</u>. But still sufficient for <u>single-key</u> FE.

We will rely on a "universal circuit": $\mathcal{U}(C, x) := C(x)$

($\mathcal{U}$ takes circuit $C$ and $x$ as input and outputs $C(x)$)




Ciphertext will be a garbled circuit for $\mathcal{U}$ along with wire labels for $x$

Secret key (for circuit $C$) will allow recovering <u>non-interactively</u> the wire labels for $C$

} decryption is garbled circuit evaluation