Functional encryption: generalization of attribute-based encryption and predicate encryption

\[ \text{decryption reveals a function of the message} \]

- Setup \((\mathcal{A}) \rightarrow (\mathsf{mpk}, \mathsf{msk})\)
- Encrypt \((\mathsf{mpk}, x) \rightarrow \mathsf{ctx}\)
- KeyGen \((\mathsf{msk}, f) \rightarrow \mathsf{sky}\)
- Decrypt \((\mathsf{sky}, \mathsf{ctx}) \rightarrow f(x)\)

**Correctness:**

\[
\text{if } (\mathsf{mpk}, \mathsf{msk}) \leftarrow \text{KeyGen}(\mathcal{A}) \quad \text{then } \mathsf{ctx} \leftarrow \text{Encrypt}(\mathsf{mpk}, x) \quad \mathsf{sky} \leftarrow \text{KeyGen}(\mathsf{msk}, f) \quad \text{then } \text{Decrypt}(\mathsf{sky}, \mathsf{ctx}) = f(x)
\]

**Security:**

- Adversary \(\mathcal{A}\) challenges \(\mathsf{mpk, msk} \leftarrow \text{Setup}(\mathcal{A})\)
- \(\mathsf{mpk, msk} \leftarrow \text{Setup}(\mathcal{A})\)
- \(\mathsf{sky} \leftarrow \text{KeyGen}(\mathsf{msk}, f)\)
- \(\mathsf{ctx} \leftarrow \text{Encrypt}(\mathsf{mpk}, x)\)
- \(\mathsf{sky} \leftarrow \text{KeyGen}(\mathsf{msk}, f)\)
- \(\mathsf{ctx} \leftarrow \text{Encrypt}(\mathsf{mpk}, x)\)
- \(f(x) = f(x_i) \text{ for all functions } f\)

\(b \leftarrow \mathcal{A}.\mathsf{En()}\)

Secure if for all efficient and admissible adversaries \(A\):

\[
\Pr[b = 1] - \Pr[b = 0] = \text{negl}
\]

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.

Today: consider a simple setting where we only need PKE

- Single-key FE: adversary can only see a single key for the FE scheme

Main building block: garbled circuit (more generally: randomized encoding)

- A common tool in cryptography, core building block for secure computation
Key ingredient: "garbling" protocol (garbled circuits)

\[ \begin{array}{c}
\text{AND} \quad x_1 \land x_2 \\
\downarrow \\
x_3 = x_1 \land x_2
\end{array} \quad \Rightarrow \quad \begin{array}{c|c|c|c|}
X_1 & X_2 & X_1 \land X_2 \\
\hline
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 1 & 1
\end{array} \]

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

\[ \begin{array}{c}
\text{AND} \quad x_1 \land x_2 \\
\downarrow \\
x_3 = x_1 \land x_2
\end{array} \]

\[ k_i^{(a)} \quad \text{key associated with wire value} \quad b \]

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

\[
\begin{array}{c|c|c|c|c|}
X_1 & X_2 & X_3 = X_1 \land X_2 \\
\hline
0 & 0 & k_2^{(a)} & 0 & k_3^{(a)} \\
0 & 1 & k_2^{(a)} & 1 & k_3^{(a)} \\
1 & 0 & 0 & k_2^{(a)} & k_3^{(a)} \\
1 & 1 & 1 & k_2^{(a)} & k_3^{(a)}
\end{array}
\]

\[
\begin{array}{c}
ct_m \leftarrow \text{Encrypt}(k_i^{(a)}, \text{Encrypt}(k_i^{(a)}, k_i^{(a)})) \\
ct_m \leftarrow \text{Encrypt}(k_i^{(a)}, \text{Encrypt}(k_i^{(a)}, k_i^{(a)})) \\
ct_m \leftarrow \text{Encrypt}(k_i^{(a)}, \text{Encrypt}(k_i^{(a)}, k_i^{(a)})) \\
ct_m \leftarrow \text{Encrypt}(k_i^{(a)}, \text{Encrypt}(k_i^{(a)}, k_i^{(a)}))
\end{array}
\]

\(\text{randomly shuffle ciphertexts}\)

3) Construct decoding table for output values

\[ k_3^{(a)} \rightarrow 0 \] Alternatively, can just encrypt output value instead of keys for output wires

\[ k_3^{(a)} \rightarrow 1 \]

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:

\[ \text{try decrypting each ciphertext with the input keys, and take the output key to be the ciphertext that decrypts} \]

\[ \text{decide using decoding table} \]

**Invariant:** given keys for input wires of a gate, can derive key corresponding to output wire \( \Rightarrow \) 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)

**Lots of optimizations!**

- \text{Free XOR}: no need to provide garbled truth table for xor gates
- \text{Half-Gates}: only need 2 ciphertexts (instead of 4) for each AND gate
- More recent: only need 1.5 ciphertexts per AND gate\(^1\) (i.e. \(1^{(a)} = 1.5 \Lambda + 5 \text{ bits}\) [RR21])

\(\text{Abstractly: } \text{Garble}(\Phi, C) \rightarrow (E, \{L_i^{(a)}, S_i^{(a)}, S_i^{(b)}\}) \]

\(\text{Eval}(C, \{L_i^{(a)}, S_i^{(a)}\}) \rightarrow y\)

- \text{Correctness: For all circuits } C: \text{for } \Phi \rightarrow S_i^{(a)} \text{ and all } \Phi \subseteq S_i^{(a)}:

\[ \text{if } (E, \{L_i^{(a)}, S_i^{(a)}, S_i^{(b)}\) \leftarrow \text{Garble}(\Phi, C),
\]

\[ \Pr[\text{Eval}(C, \{L_i^{(a)}, S_i^{(a)}\}) = C(x)] = 1 \]
There exists an efficient simulator \( S \) such that for all circuits \( C : \{0,1\}^n \rightarrow \{0,1\}^m \) and \( x \in \{0,1\}^n \):

\[
\{ (E, \{ L_i \} : i \in \{0,1\}) \} \sim S(\hat{X}, C, C(x))
\]

Namely, the garbled circuit and one set of labels can be simulated just given the output \( C(x) \).

Using garbled circuits for two-party computation:

Key cryptographic building block: oblivious transfer (OT)

- Sender \((m_0, m_1)\)
- Receiver \((b \in \{0,1\})\)

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.

Two-party computation protocol is interactive. But still sufficient for single-key FE.

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

(U takes circuit \( C \) and \( x \) as input and outputs \( C(x) \))

The ciphertext will be a garbled circuit for \( U \) along with wire labels for \( x \)

Secret key (for circuit \( C \)) will allow recovering non-interactively the wire labels for \( C \) during evaluation.