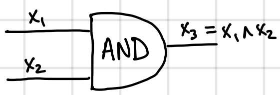
Yao's Protocol (2PC):

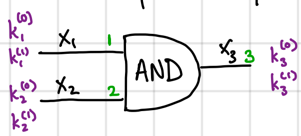
Key ingredient: "garbling" protocol (garbled circuits)



$\Longrightarrow$ 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	$k_3^{(1)}$

$\Longrightarrow$

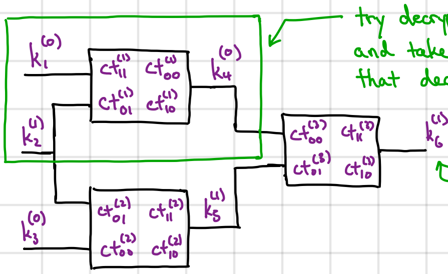
$$\left.\begin{aligned} ct_{00} &\leftarrow \mathrm{Encrypt}(k_1^{(0)}, \mathrm{Encrypt}(k_2^{(0)}, k_3^{(0)})) \\ ct_{01} &\leftarrow \mathrm{Encrypt}(k_1^{(0)}, \mathrm{Encrypt}(k_2^{(1)}, k_3^{(0)})) \\ ct_{10} &\leftarrow \mathrm{Encrypt}(k_1^{(1)}, \mathrm{Encrypt}(k_2^{(0)}, k_3^{(0)})) \\ ct_{11}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

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:



try decrypting each ciphertext with the input keys, and take the output key to be the ciphertext that decrypts

decode using decoding table

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)

Yao's garbled circuit protocol:

circuit C

Alice (garbler) — Bob (evaluator)

Alice: ↳ private input x

Bob: ↳ private input y

Bob: 1. Prepare OT queries for bits of y

Alice: 1. Prepare garbled circuit for C

Bob → Alice: OT for labels for y

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

Alice → Bob: OT responses for labels of Bob's input

Alice → Bob: garbled circuit

Alice → Bob: labels for Alice's input x

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

Correctness: Follows by correctness of OT and of the garbling construction

Security: Relies on security of OT and garbling transformation

↳ Simulate Bob's view given output of computation (using the garbled circuit simulator) ← relies on OT simulator to simulate OT responses

↳ Simulate Alice's view using OT simulator

Variants: 1. If both parties should learn output, Bob can send it to Alice.

2. Can extend to malicious security (need additional rounds and some modifications).

Many optimizations possible:

1. free XOR – no need to send garbled tables for XOR gates in circuit
2. half gates – only need two ciphertexts for each AND gate (not 4)

$\}$ AND and XOR are universal

↳ standard basis for garbled circuits

3. no need to double encrypt – can "encrypt" once using key derived from input keys

Final major topic in this course: post-quantum cryptography and the next generation of cryptography

We will not have time to cover quantum computing in this course. We will just state the implications:

Grover's algorithm: Given black-box access to a function $f : [N] \to \{0,1\}$, Grover's algorithm finds an $x \in [N]$ such that $f(x) = 1$ by making $O(\sqrt{N})$ queries to f.

"Searching an unsorted database of size N in time $O(\sqrt{N})$."

Implications in cryptography: Consider a one-way function over a 128-bit domain. The task of inverting a one-way function is to find $x \in \{0,1\}^{128}$ such that $f(x) = y$ for some fixed target value f. Exhaustive search would take time $\approx 2^{128}$ on a classical computer, but using Grover's algorithm, can perform in time $\approx \sqrt{2^{128}} = 2^{64}$.

$\Rightarrow$ For symmetric cryptography, need to double key-sizes to maintain same level of security (unless there are new quantum attacks on the underlying construction itself.

$\Rightarrow$ Use AES-256 instead of AES-128 (not a significant change!)

though requires large space

Similar algorithm can be applied to obtain a quantum collision-finding algorithm that runs in time $\sqrt[3]{N}$ where N is the size of the domain (compare to $\sqrt{N}$ for the best classic algorithm)

↳ Instead of using SHA-256, use SHA-384 (not a significant change)

Main takeaway: Symmetric cryptography mostly unaffected by quantum computers ~ generally just require a modest increase in key size

↳ e.g., symmetric encryption, MACs, authenticated encryption

Story more complicated for public-key primitives:

- Simon's algorithm and Shor's algorithm provide polynomial-time algorithms for solving discrete log (in any group with an efficiently-computable group operation) and for factoring
- Both algorithms rely on period finding (and more broadly, on solving the hidden subgroup problem)

Thus, if large scale quantum computers come online, we will need new cryptographic assumptions for our public-key primitives

↳ All the algebraic assumptions we have considered so far (e.g., discrete log, factoring, pairings) are broken

How realistic is this threat?
- Lots of progress in building quantum computers recently by both academia and industry (e.g., see initiatives by Google, IBM, etc.)
- To run Shor's algorithm to factor a 2048-bit RSA modulus, estimated to need a quantum computer with ≈ 10000 logical qubits (analog of a bit in classical computers)
  - ↳ With quantum error correction, this requires ≥ 10 million physical qubits to realize
  - ↳ Today: machines with 50-60 physical qubits, so still very far from being able to run Shor's algorithm
- Optimistic estimate: At least 20-30 years away (and some say never...)