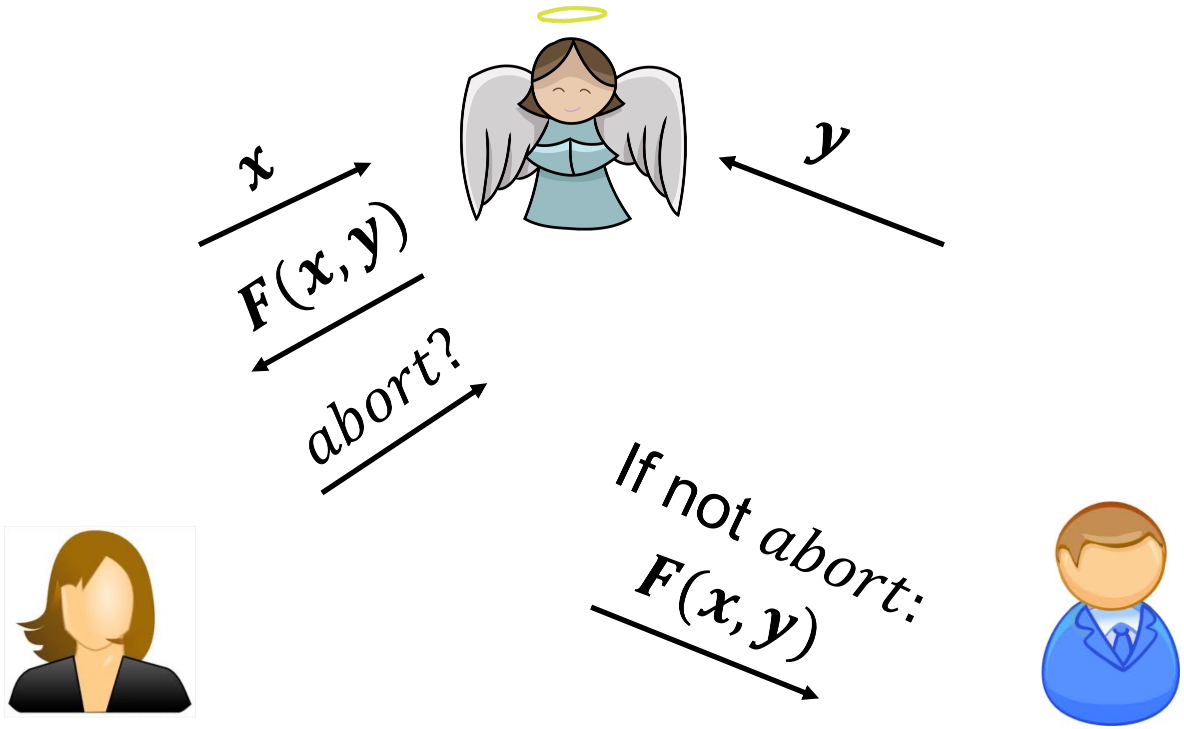
MIT 6.875

Foundations of Cryptography
Lecture 20

Security against Malicious (Active) Adversaries

New (Less) Ideal Model



Secure Two-Party Comp: New Def

(possibly randomized) $F(x, y; r) = (F_A(x, y; r), F_B(x, y; r))$

Input: x **Input: y**

Alice Bob

There exists a PPT simulator SIM_A such that for any x and y:

$$\big(SIM_A\big(x, F_A(x, y)\big), F(x, y)\big) \cong \big(View_A(x, y), F(x, y)\big)$$

i.e. the joint distribution of the view and the output is correct

Malicious Parties: Issues to Handle

1. Input (In)dependence: A malicious Alice could choose her input to depend on Bob's, something she cannot do in the ideal world.

Example: $F\big((a,b),x\big) = (\perp, ax+b)$

2. Randomness: A malicious Bob could choose his "random string" in the protocol the way she wants, something she cannot do in the ideal world.

Example: our OT *protocol*

unavoidable

3. (Un)fairness: A malicious party could block the honest party from learning the output, while learning it herself.

4. Deviate from Protocol Instructions.

The “GMW Compiler”

Theorem **[Goldreich-Micali-Wigderson’87]**: Assuming one-way functions exist, there is a general way to transform any semi-honest secure protocol computing a (possibly randomized) function F into a maliciously secure protocol for F.

Input Independence

1. Input (In)dependence: A malicious party could choose her input to depend on Bob's, something she cannot do in the ideal world.

Solution: Each party commits to their input in sequence, and provides a **zero-knowledge proof of knowledge** of the underlying input.

Solution: Coin-Tossing Protocol

2. Randomness: A malicious party could choose her "random string" in the protocol the way she wants, something she cannot do in the ideal world.

<u>*Def:*</u> Realize the functionality $F(1^n, 1^n) = (r, Com(r))$.

$$\xrightarrow{Com(r_1)}$$

$$\xleftarrow{r_2}$$

Output $r = r_1 \oplus r_2$

Output $(Com(r_1), r_2)$

Zero Knowledge Proofs

4. Deviate from Other Protocol Instructions.

Solution: Each message of each party is a *deterministic* function of their input, their random coins and messages from party B.

When party A sends a message $m = m(x_A, r_A, \overline{msg_B})$, they also prove in zero-knowledge that they did so correctly. That is, they prove in ZK the following NP statement:

$$(m, \overline{msg_B}, XCom, RCom)\text{: } \exists\, x_A, r_A \text{ s.t.}$$
$$m = m(x_A, r_A, \overline{msg_B}) \wedge XCom = \mathrm{Com}(x_A) \wedge$$
$$RCom = \mathrm{Com}(r_A)$$

Optimizations

Optimization 1: Preprocessing OTs

Random OT tuple (or AND tuple, or Beaver tuple after D. Beaver): Alice has (α, γ_a) and Bob has (β, γ_b) which are random s.t. $\boldsymbol{\gamma_a \oplus \gamma_b = \alpha\beta}$.

Theorem: Given O(1) many *random* OT tuples, we can do OT with information-theoretic security, exchanging O(1) bits.

Optimization 2: OT Extension

Theorem
[Beaver'96, Ishai-Kushilevitz-Nissim-Pinkas'03]:

Given O(λ) many *random* OT tuples, we can generate n OT tuples exchanging O(n) bits --- as opposed to the trivial O($n\lambda$) bits --- and using only symmetric-key crypto.

Complexity of the 2-party solution

Number of OT protocol invocations $= 2 \ * \ \#AND$ gates

Can be made into O(#inputs · λ): Yao's garbled circuits

Number of rounds = AND-depth of the circuit

Can be made into O(1) rounds: Yao's garbled circuits

Communication in bits =

$$O(\#AND \cdot \lambda + \#outputs)$$

Can be made into O(λ #inputs) using FHE: but FHE is computationally more expensive concretely.

$\boldsymbol{O(1)}$-Round
Secure Two-Party Computation
(on the board)