CS4501 Cryptographic Protocols
Lecture 18: OT Extension, Malicious Adversaries

https://jackdoerner.net/teaching/#2026/Spring/CS4501

Motivating OT Extension

- Over the last few lectures, we have introduced OWFs, PRGs, and PRFs. I asserted that each of these three implies the others (we prove this in grad crypto).
- I also asserted that in practice these kinds of "symmetric" primitives are very efficient, whereas "asymmetric primitives" like the group operations in the DHKE protocol are generally much less efficient.
- Our dishonest majority protocols (GMW, Yao, BMR) make *extensive* use of OT. Our OT protocol uses asymmetric operations. Does it have to?
- Unfortunately, *it seems so*. OT implies key agreement (*Can you see why?*) and Impagliazzo and Rudich proved in 1989 that proving a key agreement protocol secure using only OWF security *as a black box* is as hard as proving $P \neq NP$.
- Notice that our OT protocol was based on an assumption about a group. We constructed a OWF with security that was implied by the same assumption about the same group, but we still had to use the group, and *not* the OWF, when we constructed OT!

Motivating OT Extension

- Unfortunately, *it seems so*. OT implies key agreement (*Can you see why?*) and Impagliazzo and Rudich proved in 1989 that proving a key agreement protocol secure when that protocol uses a OWF *as a black box* is as hard as proving $P \neq NP$.
- Notice that our OT protocol was based on an assumption about a group. We constructed a OWF with security that was implied by the same assumption about the same group, but we still had to use the group, and *not* the OWF, when we constructed OT!

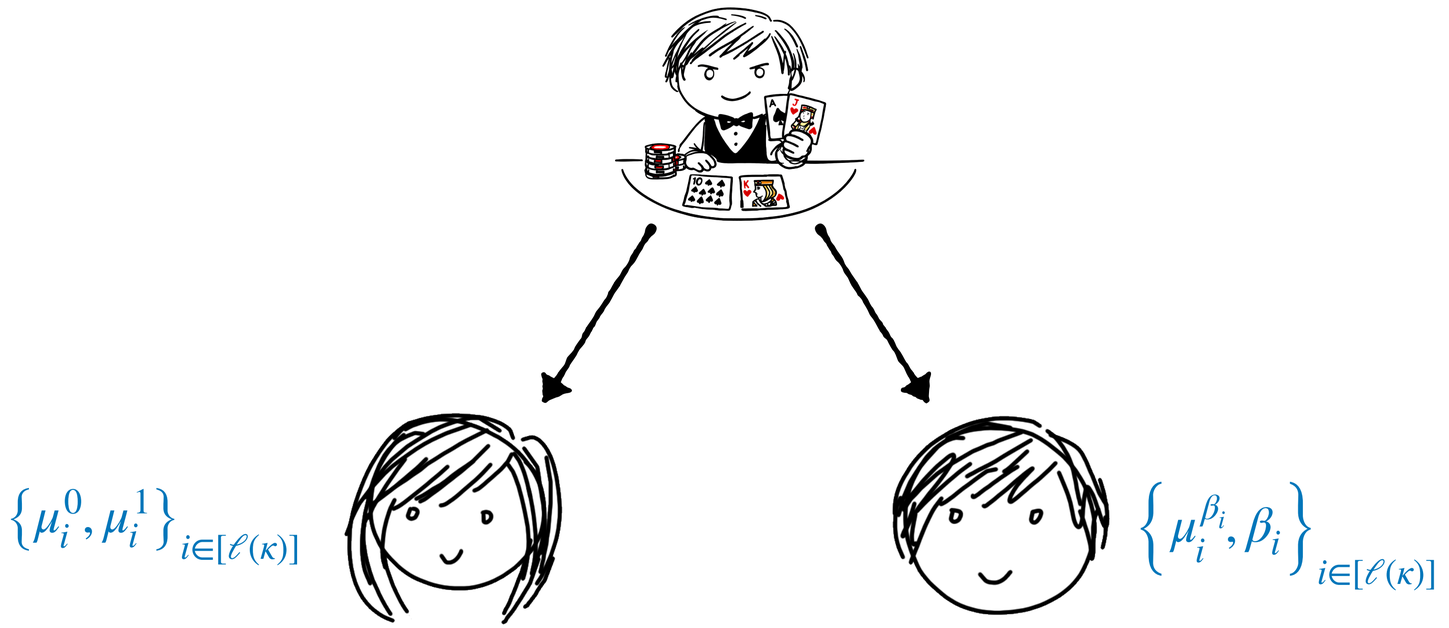
- So it seems that in order to construct OT, we *must* use some kind of concretely costly primitive (unless, perhaps, we prove $P \neq NP$).
- However, there is no proof that the *number of times* we use our concretely costly primitive goes up with the *number of OT instances* we need to realize.
- I claim the opposite. In fact, I claim that if you give me κ OT instances, then I can give you back $\ell(\kappa)$ OT instances, for any polynomial ℓ, using only a PRG!

First Step: OT in the Preprocessing Model

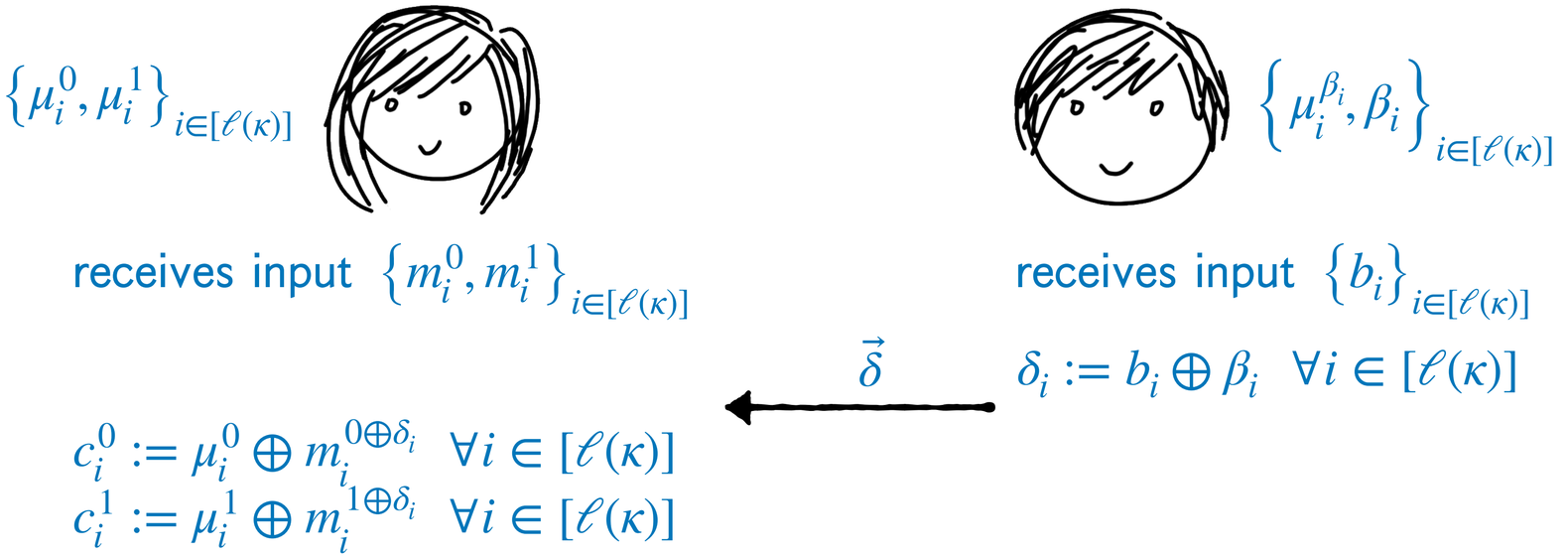
- Back in Lecture 11, we introduced the *preprocessing model*. At the time, we used it to move the bulk of the bandwidth costs on the BGW protocol into an *offline* phase that happens before the parties know their inputs.
- It is also extremely easy and efficient to construct OT in the preprocessing model!
- Recall the typical formula: a *trusted dealer* samples *correlated randomness* and sends it to all of the participants. Online, we *consume* this correlated randomness using simple and efficient information-theoretic techniques.
- The OT correlation $\mathsf{C}_{\mathsf{OT}}((m_0, m_1), (m_2, b))$ outputs 1 if and only if $m_2 = m_b$. A *random* member of this correlation comprises two random messages $\mu^0, \mu^1 \leftarrow \mathscr{M}$ and a random bit $\beta \leftarrow \{0,1\}$, plus μ^β which is fixed by the other three values.

First Step: OT in the Preprocessing Model

- The OT correlation $\mathsf{C}_{\mathsf{OT}}((m_0, m_1), (m_2, b))$ outputs 1 if and only if $m_2 = m_b$. A *random* member of this correlation comprises two random messages $\mu^0, \mu^1 \leftarrow \mathscr{M}$ and a random bit $\beta \leftarrow \{0,1\}$, plus μ^β which is fixed by the other three values.
- Let $\mathscr{M} = \{0,1\}^\kappa$ and suppose that a dealer sends a *vector* of $\ell(\kappa)$ random OT correlations to Alice and Bob.

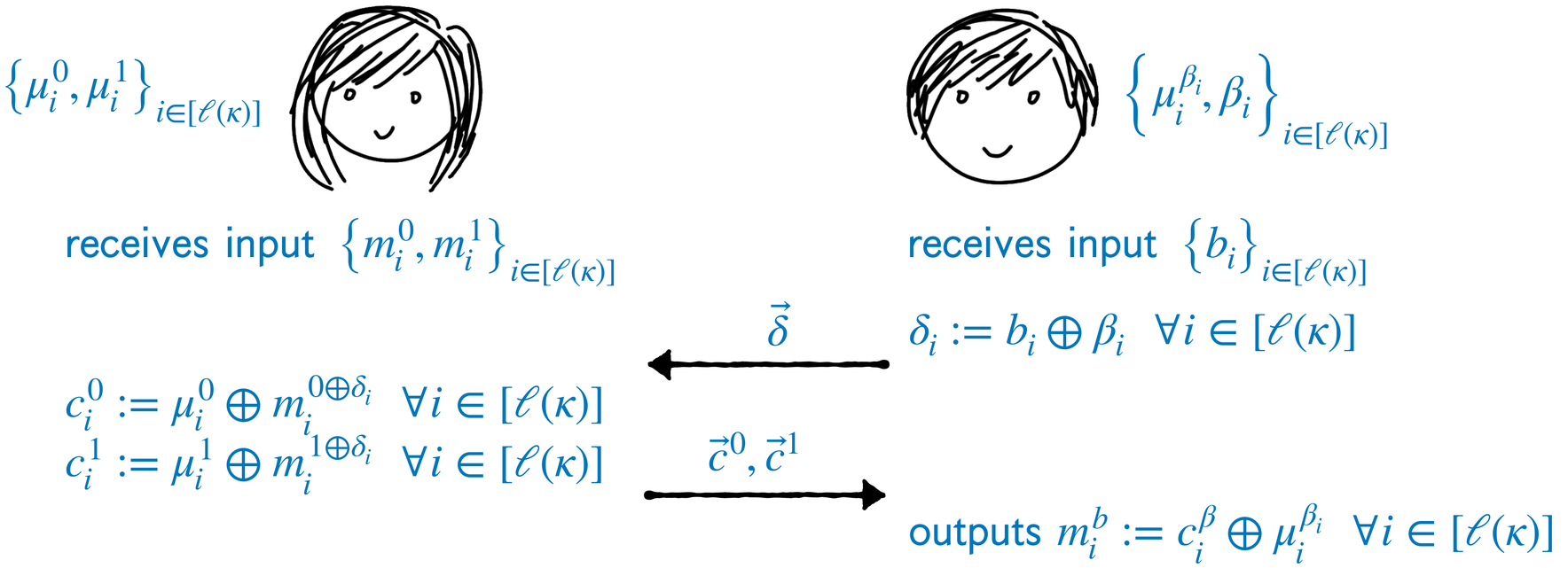


First Step: OT in the Preprocessing Model



- Later, when the parties receive their inputs, the two parties can *derandomize* the correlation they already possess much more efficiently than running OT from scratch.
- Bob computes a vector of *adjustment bits* δ_i and sends them, telling Alice whether or not each bit of his random correlation must be inverted to match his true input.
- Then Alice pads her messages with her correlation, swapping each pair according to δ_i.

First Step: OT in the Preprocessing Model



- Alice sends her encrypted (and possibly swapped) inputs to Bob.
- He can remove the pad from exactly one ciphertext from each pair, and because of the adjustment bits he sent, that one will contain the message he requested.
- I claim that the above protocol *perfectly* realizes $\ell(\kappa)$ parallel instances of the OT functionality, in the preprocessing model. Hopefully you can prove this is true!

Second Step: Realizing the Dealer

- Next, I claim that we can realize the *dealer* for $\ell(\kappa)$ random instances of the OT correlation using only κ instances of OT, plus a PRG. *Can you see how?*
- Let $\ell'(\kappa) = \ell(\kappa) \cdot (2\kappa + 1)$ and let $G : \{0,1\}^* \to \{0,1\}^*$ be a PRG that outputs $\ell'(\kappa)$ bits, given κ bits of input. Note that ℓ' is a polynomial if ℓ is.
- Now let C_{Dealer} be a 2-ary circuit that:
  1. Takes κ bits of input from each party and combines them using bitwise XOR.
  2. Uses G to expand the combined inputs to $\ell(\kappa) \cdot (2\kappa + 1)$ bits.
  3. Assigns the variables $\{\mu_i^0, \mu_i^1, \beta_i\}_{i \in [\ell(\kappa)]} \in \{0,1\}^{(2\kappa+1)\times\ell(\kappa)}$ using the output of G.
  4. Outputs $\{\mu_i^0, \mu_i^1\}_{i \in [\ell(\kappa)]}$ to P_1 and $\{\mu_i^{\beta_i}, \beta_i\}_{i \in [\ell(\kappa)]}$ to P_2.
- By Theorem 1, $\pi_{\mathsf{Yao}}(C_{\mathsf{Dealer}})$ realizes $\mathcal{F}_{\mathsf{SFE}}(2, C_{\mathsf{Dealer}}, \{0,1\}^\kappa, \{0,1\}^\kappa)$, and since Bob only supplies κ input bits, $\pi_{\mathsf{Yao}}(C_{\mathsf{Dealer}})$ only uses κ instances of $\mathcal{F}_{\mathsf{OT}}(2,1)$.
- By the PRG security of G, $\mathcal{F}_{\mathsf{SFE}}(2, C_{\mathsf{Dealer}}, \{0,1\}^\kappa, \{0,1\}^\kappa)$ is indistinguishable from the Dealer for $\ell(\kappa)$ instances of OT!

Second Step: Realizing the Dealer

- There are many ways to perform OT extension that don't involve garbling.
- Much like garbling, there is a long history of *many* optimizations.
- In 2019, Boyle, Couteau, Gilboa, Ishai, Kohl, and Scholl proved that the dealer can be realized using an amount of communication that grows *logarithmically* in $\ell(\kappa)$, plus κ "base" OT instances. This means that in an amortized sense, performing OT on two messages costs about the same as transmitting them!

- Even more intriguingly, Orlandi, Scholl, and Yakoubov proved that if you make certain assumptions, you can post your public-key online, download somebody else's public key, and then realize the OT dealer *without interacting at all.*

Malicious Adversaries!

Syllabus (tentative):

A taxonomy of adversaries; a variety of techniques

(now the taxonomy should be clearer than it was before)

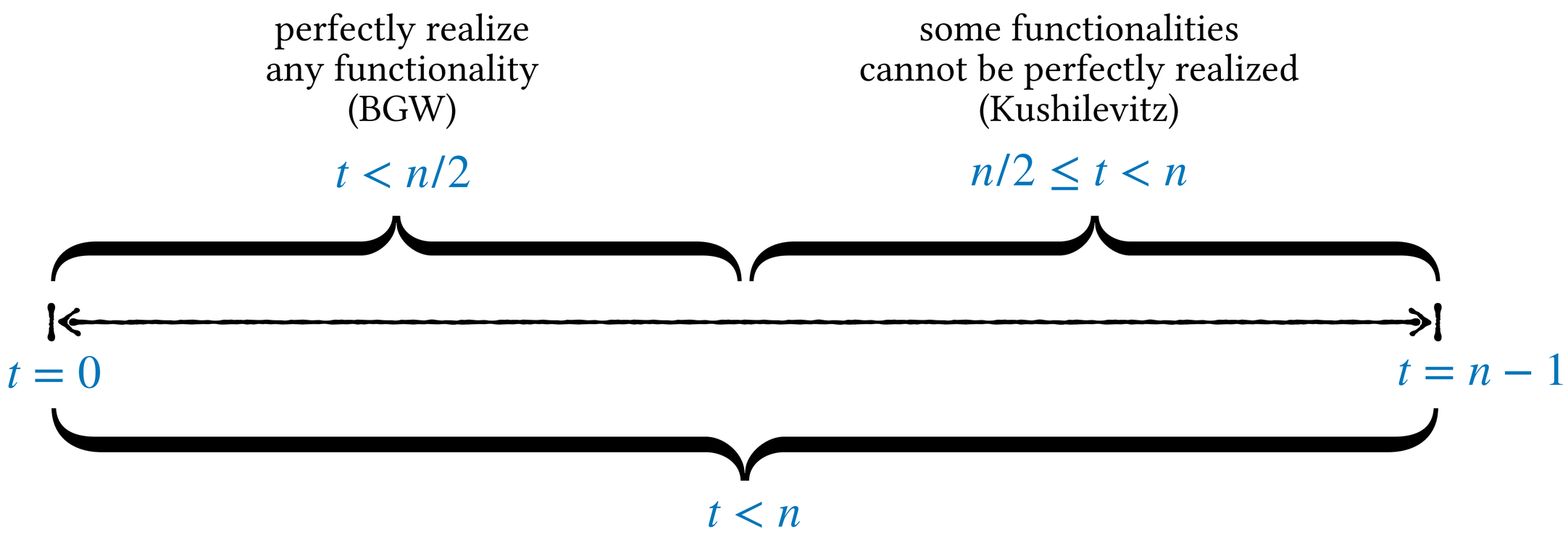
	Part 1: *Information-theoretic* techniques. Adversaries with unbounded power	**Part 2:** *Cryptographic* techniques. Adversaries with bounded power
Semi-honest Adversaries: *follow* the rules of the protocol	Secret Sharing BGW protocol for an honest majority	Oblivious Transfer GMW protocol for a dishonest majority Yao's protocol for two parties Fully Homomorphic Encryption?
Malicious Adversaries: *break* the rules of the protocol	Verifiable Secret Sharing BGW protocol for honest supermajority	Coin Tossing Zero-Knowledge Proofs Byzantine Agreement + Broadcast GMW Compiler

Overarching Questions:

How do we characterize unknown adversaries? How do we formalize intuitive security notions?
What kinds computation can we perform securely in each setting?

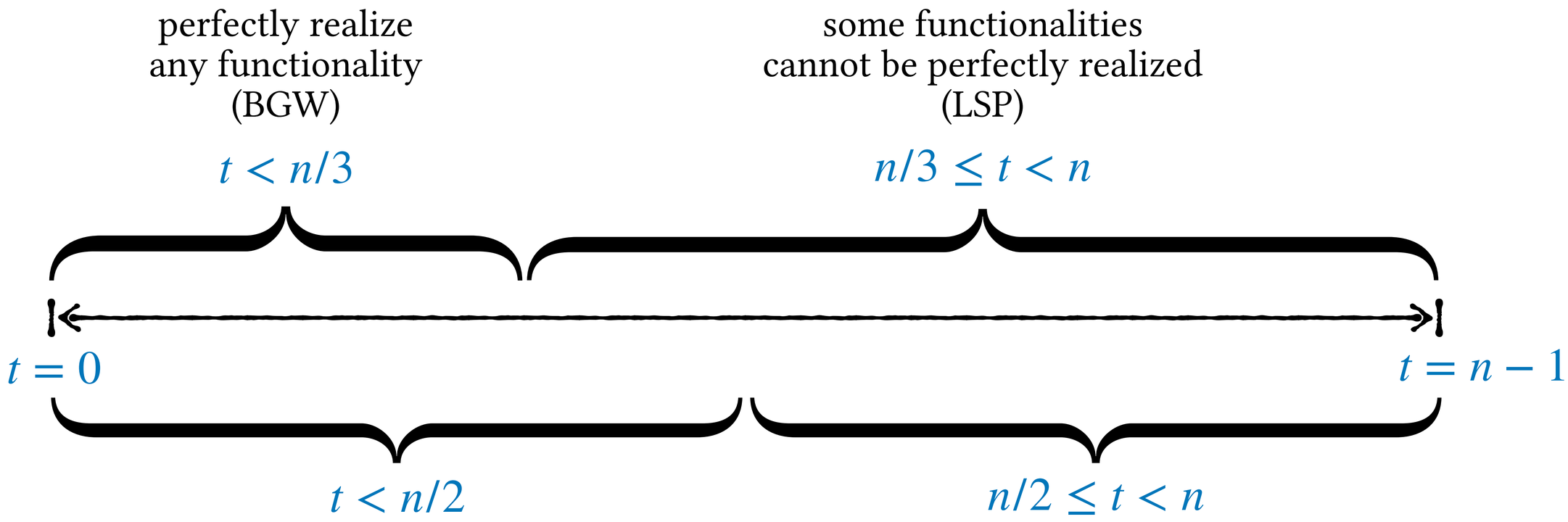
The Landscape of Semi-Honest Security

(Let n be the number of parties and t be the number of corruptions)



The Landscape of Malicious Security

(Let n be the number of parties and t be the number of corruptions)



Today: What can malicious adversaries do?

- We will revisit the various protocols we have introduced over the last 8 weeks and ask ourselves what kind of attacks a malicious adversary could perform.
- We will identify four general kinds of attacks and relate each one to an important problem in multi-party computation.
- We will develop some general intuition about how, if we were able to solve the problems above, then we could *force* parties to behave honestly (but we won't formalize this until later in the semester).

First Example: GC

Suppose that Alice is corrupted by a *malicious* $\mathcal{A}$. What could go wrong?

w_1	Garbled
0	$k_1^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_1^1 \leftarrow \text{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
1	0	0	$\tilde{c}^1_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^0}(k_4^0))$
1	1	1	$\tilde{c}^2_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^1}(k_4^1))$
0	1	0	$\tilde{c}^3_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^1}(k_4^0))$
0	0	0	$\tilde{c}^4_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^0}(k_4^0))$

w_2	Garbled
0	$k_2^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_2^1 \leftarrow \text{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
0	1	1	$\tilde{c}^1_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^1}(k_3^1))$
1	1	0	$\tilde{c}^2_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^1}(k_3^0))$
1	0	1	$\tilde{c}^3_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^0}(k_3^1))$
0	0	0	$\tilde{c}^4_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	$k_3^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_3^1 \leftarrow \text{Gen}(1^\kappa)$

We'll focus on this table to start. How could Alice Cheat?

What are the consequences of her different strategies?

Garbled
$\tilde{c}^1_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^0}(k_4^0))$

Garbled
$\tilde{c}^1_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^0}(k_3^0))$

$k_1^{x_1}$

w_3	Garbled
0	k_3^0
1	k_3^1

First Example: GC

Suppose that Alice is corrupted by a *malicious* $\mathcal{A}$. What could go wrong?

- She could switch the rows, flipping Bob's output bit. Result: She gets correct output, but he doesn't. He cannot tell anything has gone wrong.
- She could send him two bad keys. He gets no output!
- She could send him *one* bad key. If his output is 0, he cannot tell anything is wrong. If his output would be 1, he gets no output instead.

  If he needs his output to continue in some bigger protocol, then she might learn what his output was by observing whether he can continue in that protocol! We call this a *selective failure attack.*

We'll focus on this table to start. How could Alice Cheat?

What are the consequences of her different strategies?

Garbled
$\tilde{c}^1_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^4_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^0))$

Garbled
$\tilde{c}^1_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^0))$
$\tilde{c}^3_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^0))$

$k_1^{x_1}$

w_3	Garbled
0	k_3^0
1	k_3^1

y_2

w_3	Garbled
0	$k_3^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_3^1 \leftarrow \mathsf{Gen}(1^\kappa)$

First Example: GC

Suppose that Alice is corrupted by a *malicious* $\mathcal{A}$. What could go wrong?

w_1	Garbled
0	$k_1^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_1^1 \leftarrow \text{Gen}(1^\kappa)$

w_2	Garbled
0	$k_2^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_2^1 \leftarrow \text{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
1	0	0	$\tilde{c}^1_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^0}(k_4^0))$
1	1	1	$\tilde{c}^2_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^1}(k_4^1))$
0	1	0	$\tilde{c}^3_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^1}(k_4^0))$
0	0	0	$\tilde{c}^4_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^0}(k_4^0))$

w_4	Garbled
0	$k_4^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_4^1 \leftarrow \text{Gen}(1^\kappa)$

x_1

x_2

w_1

w_1

w_2

w_3

w_3

$\wedge$

$\oplus$

y_2

What about this table. Does she have any new opportunities here?

w_1	w_2	w_4	Garbled
0	1	1	$\tilde{c}^1_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^1}(k_3^1))$
1	1	0	$\tilde{c}^2_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^1}(k_3^0))$
1	0	1	$\tilde{c}^3_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^0}(k_3^1))$
0	0	0	$\tilde{c}^4_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	$k_3^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_3^1 \leftarrow \text{Gen}(1^\kappa)$

Garbled
$\tilde{c}^1_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^0}(k_4^0))$

Garbled
$\tilde{c}^1_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^0}(k_3^0))$

$k_1^{x_1}$

w_3	Garbled
0	k_3^0
1	k_3^1

First Example: GC

Suppose that Alice is corrupted by a *malicious* $\mathcal{A}$. What could go wrong?

- She could garble a completely different gate! This would change the function they are computing Bob has no way to tell this has occurred.
- She could replace the keys for one of the values on the output wire with bad keys. This selective failure attack enables her to learn the value of the internal wire according to whether or not Bob sends output to her.
- She could replace the keys for one of the values on Bob's input wire with bad keys. This selective failure attack enables her to learn Bob's input!

w_4	Garbled
0	$k_4^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_4^1 \leftarrow \text{Gen}(1^\kappa)$

What about this table. Does she have any new opportunities here?

Garbled
$\tilde{c}_{134}^1 := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}_{134}^2 := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}_{134}^3 := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}_{134}^4 := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^0}(k_4^0))$

Garbled
$\tilde{c}_{123}^1 := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}_{123}^2 := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}_{123}^3 := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}_{123}^4 := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^0}(k_3^0))$

$k_1^{x_1}$

w_3	Garbled
0	k_3^0
1	k_3^1

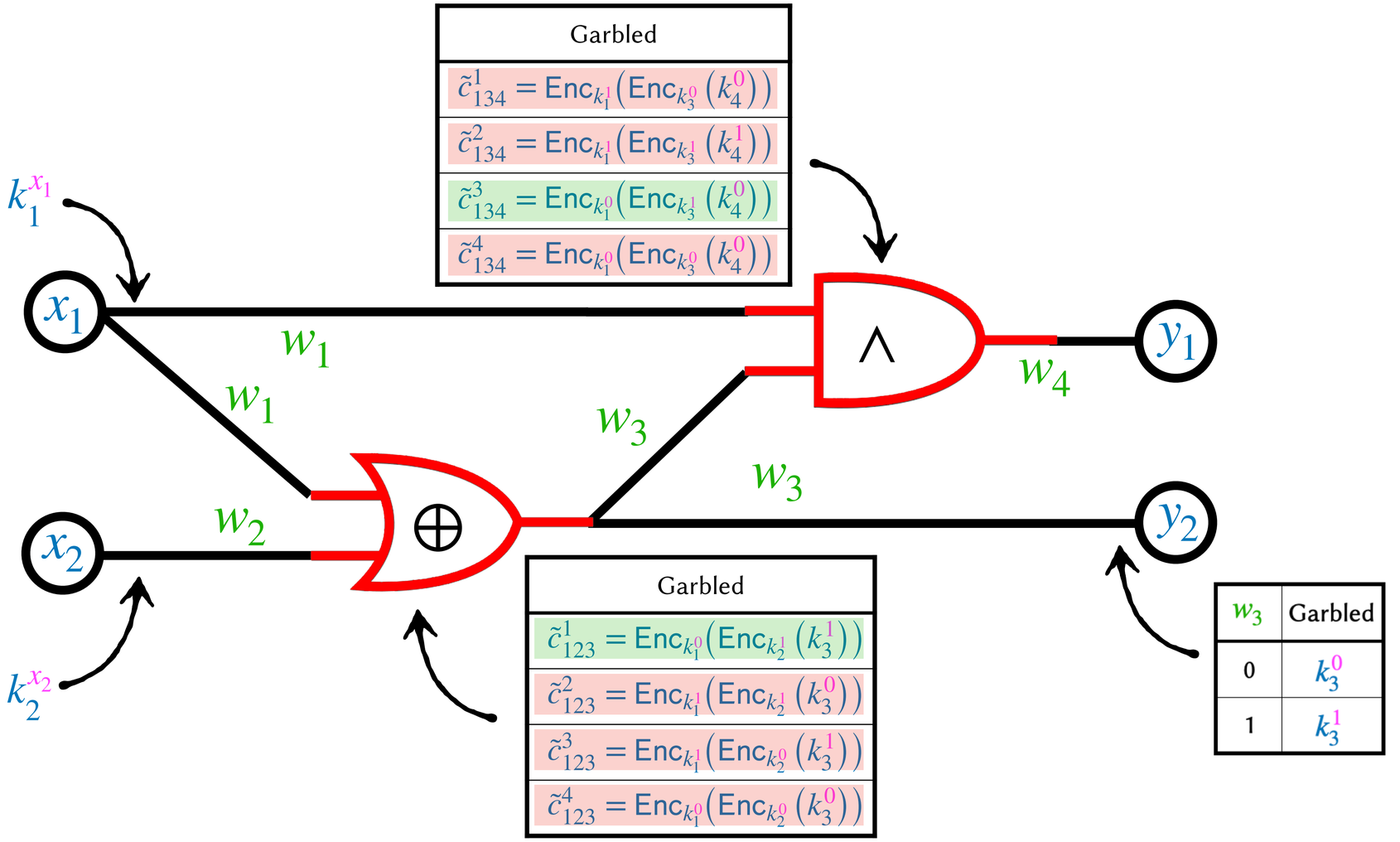
y_2

w_3	Garbled
0	$k_3^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_3^1 \leftarrow \text{Gen}(1^\kappa)$

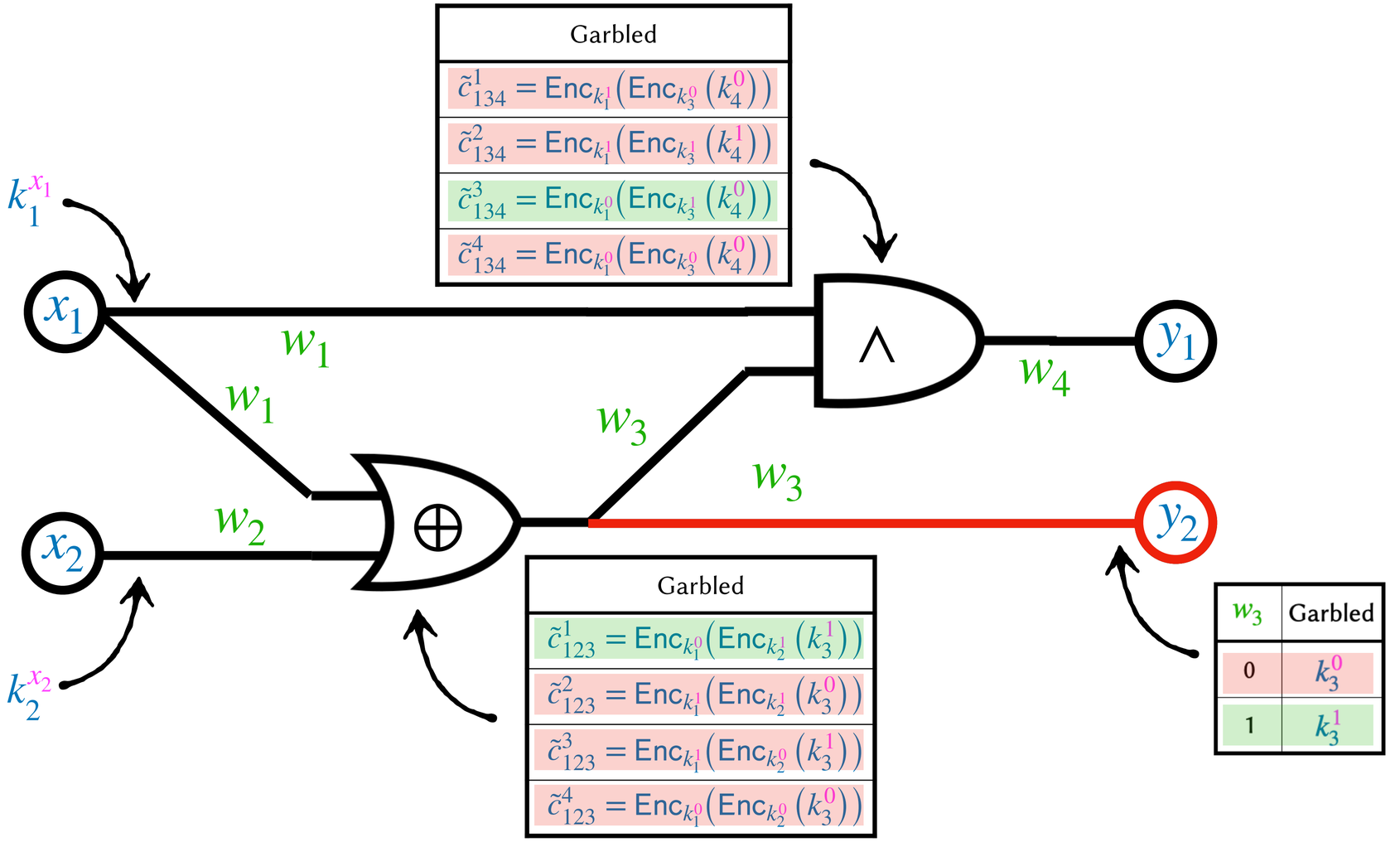
			d
			$\text{nc}_{k_2^1}(k_3^1))$
			$\text{nc}_{k_2^1}(k_3^0))$
1	0	1	$\tilde{c}_{123}^3 := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^0}(k_3^1))$
0	0	0	$\tilde{c}_{123}^4 := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^0}(k_3^0))$

1	$k_2^1 \leftarrow \text{Gen}(1^\kappa)$

First Example: GC



First Example: GC



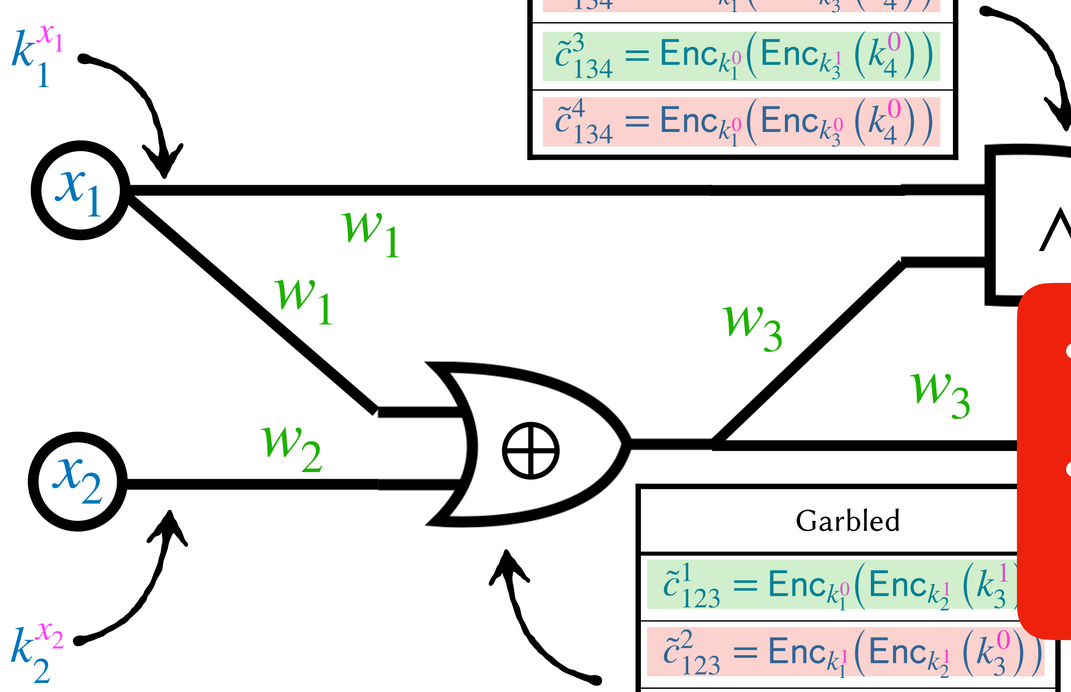
First Example: GC

But what about Bob the Evaluator? What if he cheats?

w_4	Garbled
0	$k_4^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_4^1 \leftarrow \mathsf{Gen}(1^\kappa)$

$k_4^{w_4}$

Garbled
$\tilde{c}^1_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$



Garbled
$\tilde{c}^1_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

1	k_3^1

How can Bob cheat?
What are the consequences?

- He can refuse to communicate
- He can send the wrong value to Alice, but he would achieve the same by not communicating

A first flavor of attack

- Deviation from local instructions by a corrupt party.
- Maybe completely undetectable. If not detected, the output could be changed in an arbitrary way, or something arbitrary could be leaked to the adversary.
- Sometimes *the act of detecting it* also leaks something to the adversary!
- In the specific scenario of Garbled Circuits, what would we need to achieve in order to protect Bob from this kind of attack? *Can you think of something?*
- Bob needs to to detect if *all* of the rows of *every* gate are correct, even if he cannot actually decrypt those rows!
- It seems intuitive that this kind of attack is impossible if Bob has a way to verify that Alice has followed exactly the instructions that were specified for her.

Second Example: $\pi_{\mathsf{GRR}}(n, t, p)$ $(n = 3,\ t = 1)$.

Inputs: Every P_i for $i \in [3]$ has input $\langle w \rangle_i = f(i)$ where $f \in \mathscr{P}_{p,t,w}$ and input $\langle w' \rangle_i = f'(i)$ where $f' \in \mathscr{P}_{p,t,w'}$.

1. Every P_i locally computes $\hat{g}(i) = f(i) \cdot f'(i) = \langle w \rangle_i \cdot \langle w' \rangle_i$. Note that $\hat{g} \in \mathscr{P}_{p,2,w \cdot w'}$.

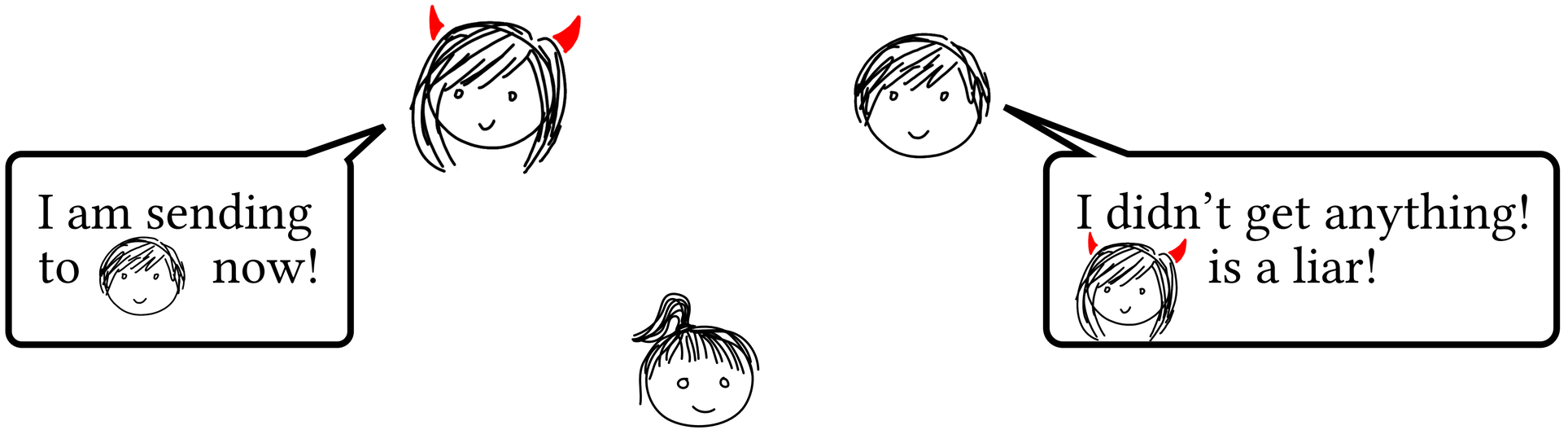
2. Every P_i samples $\langle \hat{g}(i) \rangle \leftarrow \mathsf{Share}_{p,n,t}(\hat{g}(i))$ and sends $\langle \hat{g}(i) \rangle_j$ to P_j for $j \in [3] \backslash \{i\}$.

3. Every P_i computes $\langle w \cdot w' \rangle_i = \sum_{j \in [3]} \ell_j(0) \cdot \langle \hat{g}(j) \rangle_i$.

Outputs: Each P_i ends with $\langle w \cdot w' \rangle_i$.

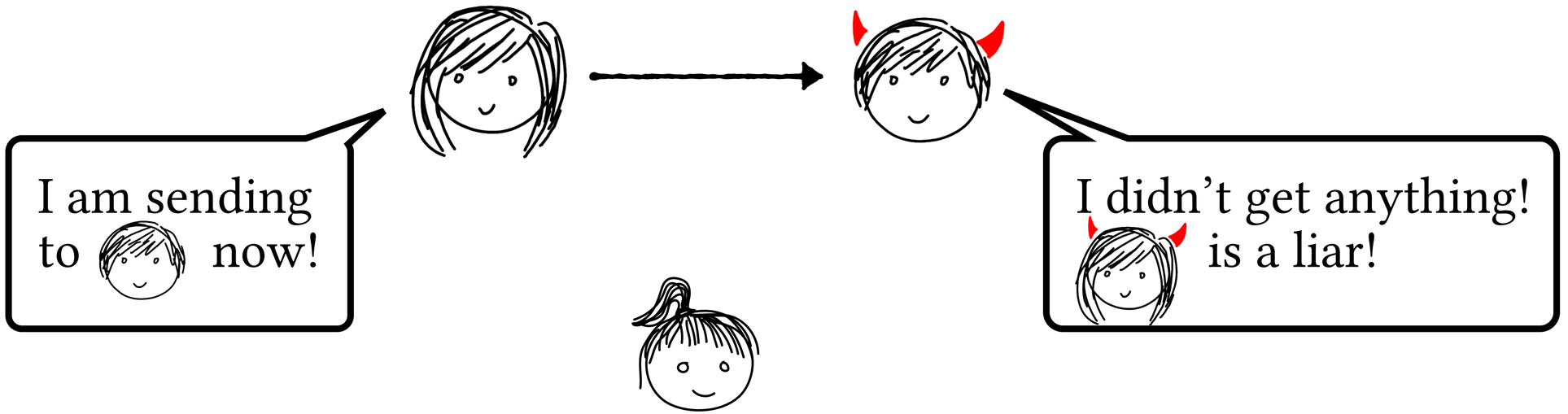
- Suppose P_1 is corrupt, and follows the protocol exactly from the point of view of P_2 and P_3. What new attack can it perform now that there are *two* honest parties?
- What if in Step 2, P_1 samples $\langle \hat{g}(i) \rangle \leftarrow \mathsf{Share}_{p,n,t}(\hat{g}(i))$ and sends $\langle \hat{g}(i) \rangle_2$ to P_2, then samples $\langle \hat{g}(i) \rangle' \leftarrow \mathsf{Share}_{p,n,t}(\hat{g}(i))$ and sends $\langle \hat{g}(i) \rangle'_3$ to P_3? In both individual cases, it behaved completely honestly and followed the instructions!

Example 2.5



- Suppose we have 3 parties, and P_1 (who is corrupt) is *supposed* to send a message to P_2 (who is honest), but she doesn't send that message. How does P_3 know?

Example 2.5



- Suppose we have 3 parties, and P_1 (who is corrupt) is *supposed* to send a message to P_2 (who is honest), but she doesn't send that message. How does P_3 know?
- If the answer is "P_2 complains", then can't P_2 also complain when it's corrupt but P_1 is honest?
- In general, we need to avoid giving any party the ability to frame another one?

A second flavor of attack

- It isn't enough that a party behaves honestly - it must behave honestly and *consistently* with respect to all of the other parties.
- This means (e.g.) that P_2 must be able to check that the interaction between P_1 and P_3 is *consistent* with its interactions with P_1 and P_3.
- On the other hand, the communication between P_1 and P_3 is supposed to be a secret that P_2 doesn't learn….

A second and a half flavor of attack

- If a problem occurs in the interaction between two parties out of a larger set, then the others *might* need to be able to disambiguate who the cheater is.
- This implies that every party must be aware of the communication that occurs between the other pairs of parties, at least some of the time (to disambiguate "didn't send a message" from "tried to frame the sender", among other things).

Third Example: 1-out-of-2 OT.

- Let $\mathcal{G}$ be a PPT algorithm that takes the security parameter 1^κ and outputs $(\mathbb{G}, g, q)$, such that $q \in \mathbb{N}$ is prime, $|\mathbb{G}| = q > \ell$, $|q| \geq \kappa$, and $\langle g \rangle = \mathbb{G}$ under some operation (denoted $\cdot$).

receives inputs $1^\kappa, m_0, m_1$

$(\mathbb{G}, g, q) \leftarrow \mathcal{G}(1^\kappa)$

$r_0 \leftarrow \mathbb{Z}_q,\ R_0 := g^{r_0},\ C_0 := X_0^{r_0} \cdot g^{m_0}$

$r_1 \leftarrow \mathbb{Z}_q,\ R_1 := g^{r_1},\ C_1 := X_1^{r_1} \cdot g^{m_1}$

$(\mathbb{G}, g, q)$

(X_0, X_1)

(R_0, C_0, R_1, C_1)

receives inputs $1^\kappa, b$

$x \leftarrow \mathbb{Z}_q,\ X_b := g^x,\ X_{1-b} \leftarrow \mathbb{G}$

outputs $m'_b \in [0,\ell]$ s.t. $g^{m'_b} = C_b \cdot R_b^{-x}$

- If Alice violates the instructions and sends a bad ciphertext, what happens?
- As long as she sends two elements of the correct group, Bob decrypts *something*, but it might not be one of her inputs (she might not even know what it is).
- This seems to be similar to the kinds of attacks we investigated for Yao's GC.

Third Example: 1-out-of-2 OT.

- Let $\mathcal{G}$ be a PPT algorithm that takes the security parameter 1^κ and outputs $(\mathbb{G}, g, q)$, such that $q \in \mathbb{N}$ is prime, $|\mathbb{G}| = q > \ell$, $|q| \geq \kappa$, and $\langle g \rangle = \mathbb{G}$ under some operation (denoted $\cdot$).

receives inputs $1^\kappa, m_0, m_1$

$(\mathbb{G}, g, q) \leftarrow \mathcal{G}(1^\kappa)$

$r_0 \leftarrow \mathbb{Z}_q,\ R_0 := g^{r_0},\ C_0 := X_0^{r_0} \cdot g^{m_0}$

$r_1 \leftarrow \mathbb{Z}_q,\ R_1 := g^{r_1},\ C_1 := X_1^{r_1} \cdot g^{m_1}$

$(\mathbb{G}, g, q)$

(X_0, X_1)

(R_0, C_0, R_1, C_1)

receives inputs $1^\kappa, b$

$x \leftarrow \mathbb{Z}_q,\ X_b := g^x,\ X_{1-b} \leftarrow \mathbb{G}$

outputs $m'_b \in [0,\ell]$ s.t. $g^{m'_b} = C_b \cdot R_b^{-x}$

- How can Bob cheat? What can he gain?
- Even if he follows all of the instructions, he might sample something from the wrong distribution. If he always samples $X_{1-b} := g^0$ then he learns both messages.
- If he picks $x^* \leftarrow \mathbb{Z}_q$ and sets $X_{1-b} := g^{x^*}$, then X_{1-b} is distributed correctly...

Third Example: 1-out-of-2 OT.

- Let $\mathcal{G}$ be a PPT algorithm that takes the security parameter 1^κ and outputs $(\mathbb{G}, g, q)$, such that $q \in \mathbb{N}$ is prime, $|\mathbb{G}| = q > \ell$, $|q| \geq \kappa$, and $\langle g \rangle = \mathbb{G}$ under some operation (denoted $\cdot$).

receives inputs $1^\kappa, m_0, m_1$

$(\mathbb{G}, g, q) \leftarrow \mathcal{G}(1^\kappa)$

$r_0 \leftarrow \mathbb{Z}_q,\ R_0 := g^{r_0},\ C_0 := X_0^{r_0} \cdot g^{m_0}$

$r_1 \leftarrow \mathbb{Z}_q,\ R_1 := g^{r_1},\ C_1 := X_1^{r_1} \cdot g^{m_1}$

$(\mathbb{G}, g, q)$

(X_0, X_1)

(R_0, C_0, R_1, C_1)

receives inputs $1^\kappa, b$

$x \leftarrow \mathbb{Z}_q,\ X_b := g^x,\ X_{1-b} \leftarrow \mathbb{G}$

outputs $m'_b \in [0,\ell]$ s.t. $g^{m'_b} = C_b \cdot R_b^{-x}$

- How can we possibly force him to sample a uniform group element such that he doesn't know the discrete logarithm?
- (Did your attempt at a solution involve one-wayness? Is that actually enough?)
- If we could somehow build a protocol to guarantee that X_{1-b} is the product of *independent* samples from *both* parties, would that be enough?

Third Example: 1-out-of-2 OT.

- Let $\mathcal{G}$ be a PPT algorithm that takes the security parameter 1^κ and outputs $(\mathbb{G}, g, q)$, such that $q \in \mathbb{N}$ is prime, $|\mathbb{G}| = q > \ell$, $|q| \geq \kappa$, and $\langle g \rangle = \mathbb{G}$ under some operation (denoted $\cdot$).

receives inputs $1^\kappa, m_0, m_1$

$(\mathbb{G}, g, q) \leftarrow \mathcal{G}(1^\kappa)$

$r_0 \leftarrow \mathbb{Z}_q,\ R_0 := g^{r_0},\ C_0 := X_0^{r_0} \cdot g^{m_0}$

$r_1 \leftarrow \mathbb{Z}_q,\ R_1 := g^{r_1},\ C_1 := X_1^{r_1} \cdot g^{m_1}$

$(\mathbb{G}, g, q)$

(X_0, X_1)

(R_0, C_0, R_1, C_1)

receives inputs $1^\kappa, b$

$x \leftarrow \mathbb{Z}_q,\ X_b := g^x,\ X_{1-b} \leftarrow \mathbb{G}$

outputs $m'_b \in [0,\ell]$ s.t. $g^{m'_b} = C_b \cdot R_b^{-x}$

- Can you think of something *else* Bob can do, even if he samples honestly and follows the protocol instructions exactly?
- What happens if he ignores the actual input b that the experiment gave him and pretends that $b = 1$. Of course he'll always output m_1... what's the problem?

Third Example: 1-out-of-2 OT.

- Let $\mathcal{G}$ be a PPT algorithm that takes the security parameter 1^κ and outputs $(\mathbb{G}, g, q)$, such that $q \in \mathbb{N}$ is prime, $|\mathbb{G}| = q > \ell$, $|q| \geq \kappa$, and $\langle g \rangle = \mathbb{G}$ under some operation (denoted $\cdot$).

receives inputs $1^\kappa, m_0, m_1$

$(\mathbb{G}, g, q) \leftarrow \mathcal{G}(1^\kappa)$

$r_0 \leftarrow \mathbb{Z}_q,\ R_0 := g^{r_0},\ C_0 := X_0^{r_0} \cdot g^{m_0}$

$r_1 \leftarrow \mathbb{Z}_q,\ R_1 := g^{r_1},\ C_1 := X_1^{r_1} \cdot g^{m_1}$

$(\mathbb{G}, g, q)$

(X_0, X_1)

(R_0, C_0, R_1, C_1)

receives inputs $1^\kappa, b$

$x \leftarrow \mathbb{Z}_q,\ X_b := g^x,\ X_{1-b} \leftarrow \mathbb{G}$

outputs $m'_b \in [0,\ell]$ s.t. $g^{m'_b} = C_b \cdot R_b^{-x}$

- What happens if he ignores the actual input b that the experiment gave him and pretends that $b = 1$. Of course he'll always output m_1... what's the problem?
- Our security definition says the simulator has to work for *all input vectors*, including the one where it receives $b = 0$ and m_0. How can it simulate his view (which always includes m_1) if this happens?

A third flavor of attack

- Freedom to sample local randomness "conveniently" on the part of a corrupt party.
- How can you check whether a sample really came from a claimed distribution?
- Even if the distribution is *correct* on its own, there could still be an attack because the adversary knows some correlated value that it shouldn't.

A fourth flavor of "attack"

- Corrupted parties can behave as though they got a different input than they were given by the experiment, but otherwise behave honestly.
- The problem stems from the fact that the simulator is forced to use inputs and outputs provided by the experiment + ideal functionality…
- This seems like a shortcoming of our model of security. It might help if the *simulator* determined the corrupt parties' inputs in the ideal world.
- But how does the simulator know what Bob's input is in the OT protocol?

Toward security against malicious adversaries

- The set of attacks we looked at isn't a rigorous taxonomy by any means!
- There are many kinds of attacks that are outside out model (for example, attacks based upon adaptive corruption or asynchrony) and a few that are inside the model, but that we didn't explicitly show (for example, "rushing" attacks).
- Nevertheless, it gives us a way to start developing ideas about malicious security, and in the end we will prove that we can handle all in-model attacks.
- In particular, over the next few lectures, we will introduce three functionalities that help us to handle the first three flavors of attack.

    1. **Reliable Broadcast** will enable all of the parties to maintain a *consistent* view of the communication that occurs within the protocol.

        The broadcast functionality $\mathcal{F}_{\mathsf{BC}}$ takes a message m from one party, and delivers that message to all of other parties. If any party receives m, it can be sure that *everyone else* also received m.

Toward security against malicious adversaries

1. **Reliable Broadcast** will enable all of the parties to maintain a *consistent* view of the communication that occurs within the protocol.

   The broadcast functionality $\mathcal{F}_{\mathsf{BC}}$ takes a message m from one party, and delivers that message to all of other parties. If any party receives m, it can be sure that *everyone else* also received m.

2. **Coin Tossing** will enable the parties to collaboratively sample random values in such a way that no group of corrupted parties can influence the distribution or learn any useful ancillary information.

   Tossed coins can be either *public* (known to all parties), or else *committed* (parties that do not know the flipped coin receive some evidence that binds the other parties to the value of that coin), but in either case are *agreed upon*.

   The (public) coin-tossing functionality $\mathcal{F}_{\mathsf{CT}}$ samples a uniform $b \leftarrow \{0,1\}$ and outputs it to all parties. Every party is sure that all others received b, and that it is unbiased and uncorrelated with any values previously in the view of $\mathcal{A}$.

Toward security against malicious adversaries

3. **Zero-Knowledge Proofs of Knowledge** will enable one party to prove the truth of any *efficiently verifiable statement* to another while revealing *nothing more than* the truth of the statement. In particular, the parties will be able to prove statements of the form

   > "Given my input, all communication that has occurred and, the randomness that we sampled, my next message in the protocol was computed honestly"

   without revealing the communication, randomness, or internal state on which the truth of such statements depends.

   Furthermore, such proofs will enable our simulators to always *extract* a well-formed input for every corrupt party, regardless of how it behaves.

   The ZKP functionality $\mathcal{F}_{\mathsf{ZK}}^{R}$ for relation R takes a statement $x \in \{0,1\}^*$ and a secret *witness* w from P_1 and outputs $(\texttt{"accept"}, x)$ to P_2 if $R(x, w) = 1$, or $(\texttt{"reject"}, x)$ otherwise. Think of w explaining why x is true, and R checks.

Toward security against malicious adversaries

- We will prove that there are some regimes in which some of these functionalities *cannot be realized*. This places strong limits on the kinds of secure computation that are possible in the presence of a malicious adversary.
- We will also show that in regimes where we *can* realize all three, we can combine them into a *protocol compiler* that takes as input any *semi-honest protocol*, and produces a protocol that realizes same functionality* with security against *malicious* adversaries.

  *if a majority of parties are dishonest, the functionality will have to be weakened in a simple and predictable way: we will give the adversary the ability to cause an *abort* which prevents the functionality from producing output.
- In order the handle the fourth flavor of attack, we will abandon the simplified security definitions that we have been using since the beginning of the semester, and instead we will formalize the full definition that was introduced in pictures only.

An intuitive preview of the GMW Compiler

- How will our compiler work?
- First, we will move all communication onto a *broadcast channel*. If the original protocol says two parties should communicate privately, they must encrypt their messages to one another, and broadcast the ciphertexts to *everyone*.
- At the beginning, the parties use a coin-tossing functionality to sample whatever private randomness every party might eventually need. Everyone receives a *commitment* to everyone else's randomness, which binds them to what was sampled.
- Similarly, all parties produce commitments that bind them to their *inputs*.
- All messages in the original protocol can be computed deterministically from the inputs, randomness, and prior transcript. Whenever a party in the original protocol sends a message, the party in the compiled protocol must prove in zero knowledge that the message they sent was computed *correctly* given their committed input and randomness, and the transcript of prior messages on the broadcast channel.

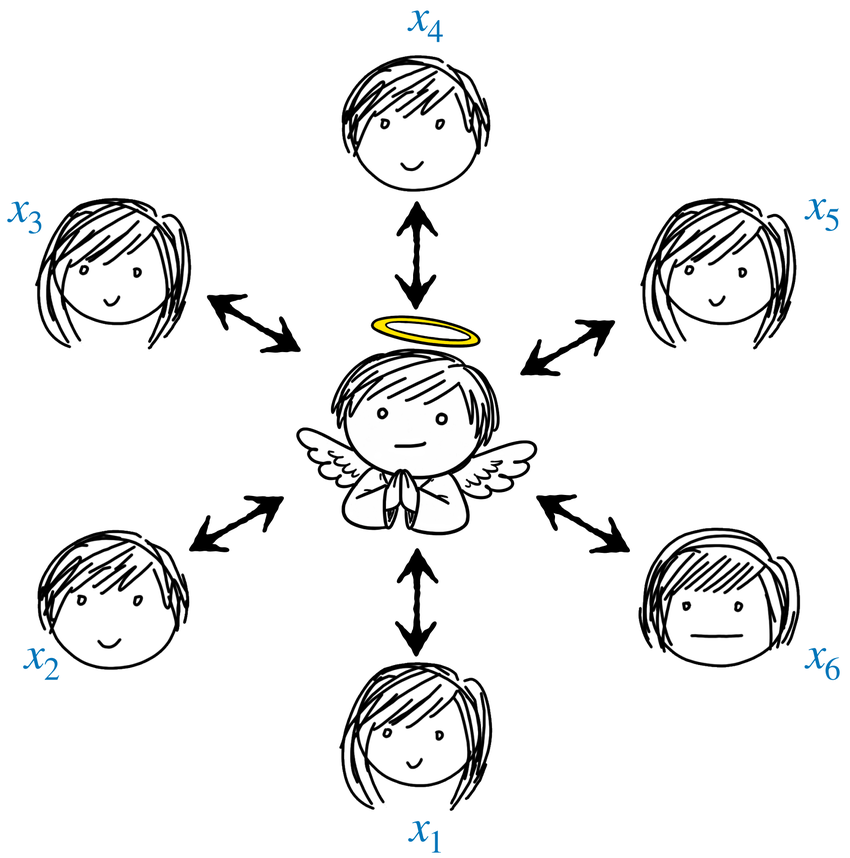
An intuitive preview of the GMW Compiler

- What if somebody cheats?
- Because broadcast, coin-tossing, and zero-knowledge proofs are *ideal*, cheating can only take the form of failing to send a message or sending a message that fails its corresponding proof of correctness. Since all parties see the same broadcast channel, all parties can agree on who cheated.
- If there are $t \geq n/2$ parties, then the honest parties *abort* and identify the cheater.
- If there are $t < n/2$ parties, then our *commitments* can take a special form known as *verifiable secret sharing*. Since the honest parties agree who cheated, they can kick out the cheater, reconstruct the cheater's inputs and randomness from secret shares (which they verified to be correct when they received them), and try again!
- This is a very intuitive overview. The devil is in the details, and there are a lot of them. We will spend the rest of the semester exploring those details.
- First we need to be clear about what kind of security we aim to achieve!

Security for protocols,
one more time!

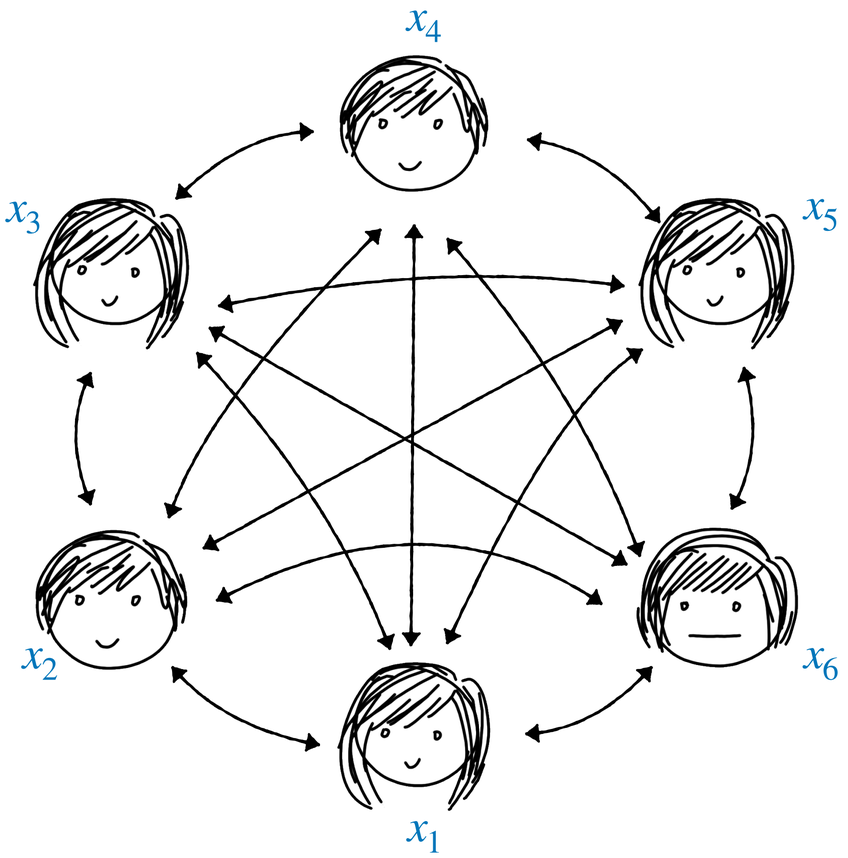
The Ideal World

1. Each P_i sends its input x_i to the ideal functionality ($\mathcal{F}$)
2. $\mathcal{F}$ computes $y = f(x_1, \ldots, x_n)$.
3. $\mathcal{F}$ sends y to every party, and they output it.

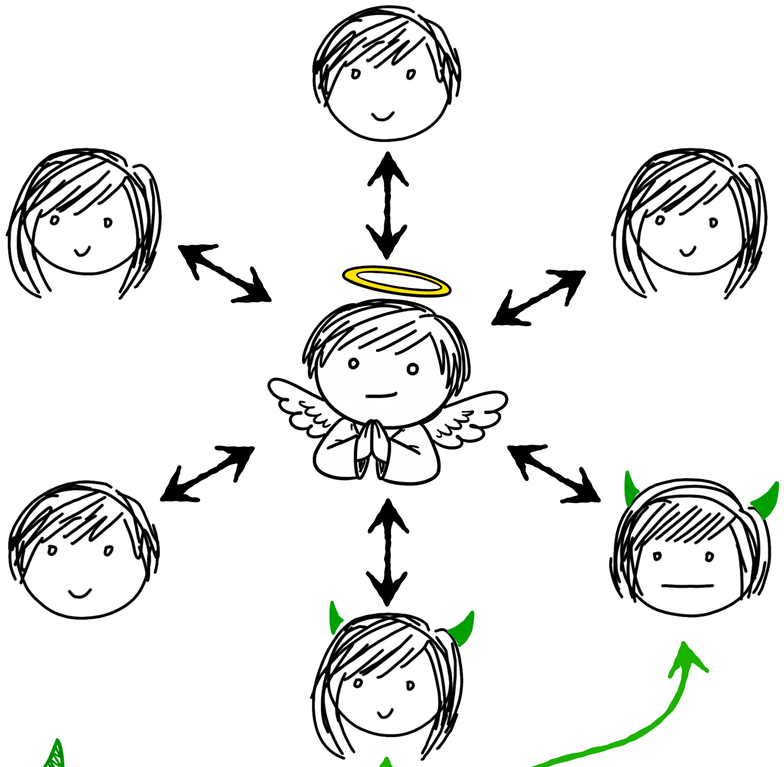


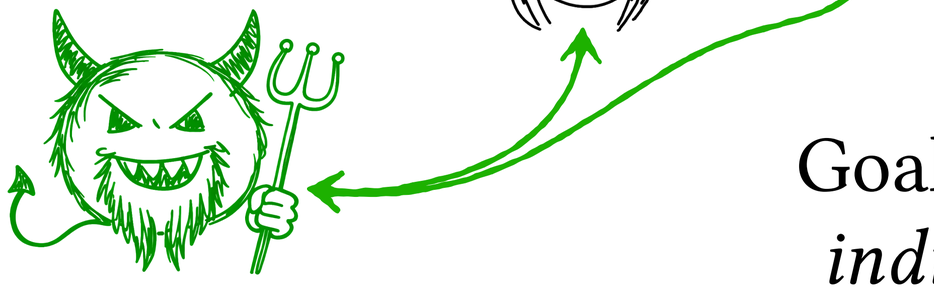
The Real World

1. The parties $(P_1, \ldots, P_n)$ run a protocol π on inputs $(x_1, \ldots, x_n)$
2. When π terminates, the parties output y.



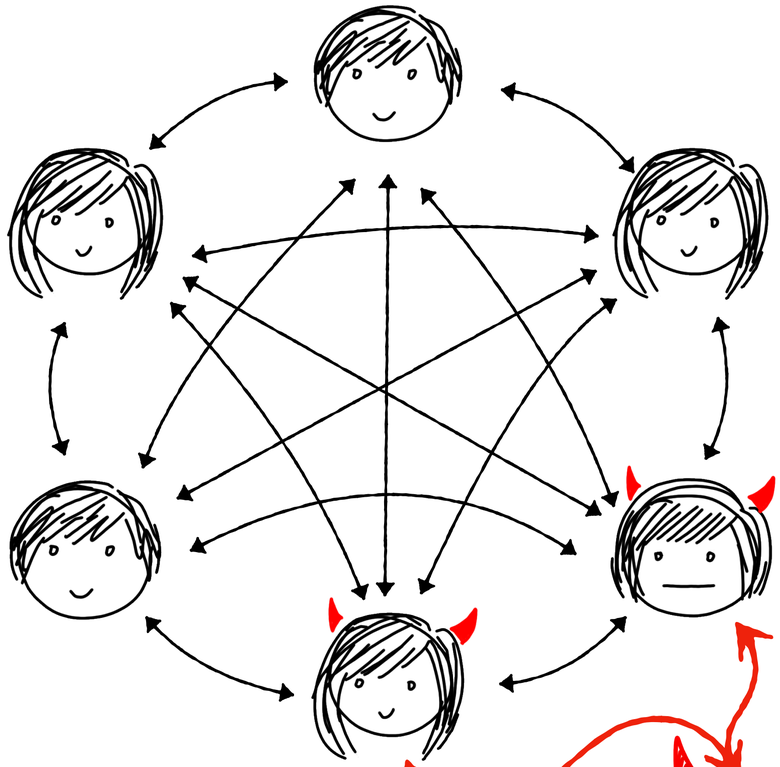
For every real adversary, an ideal adversary





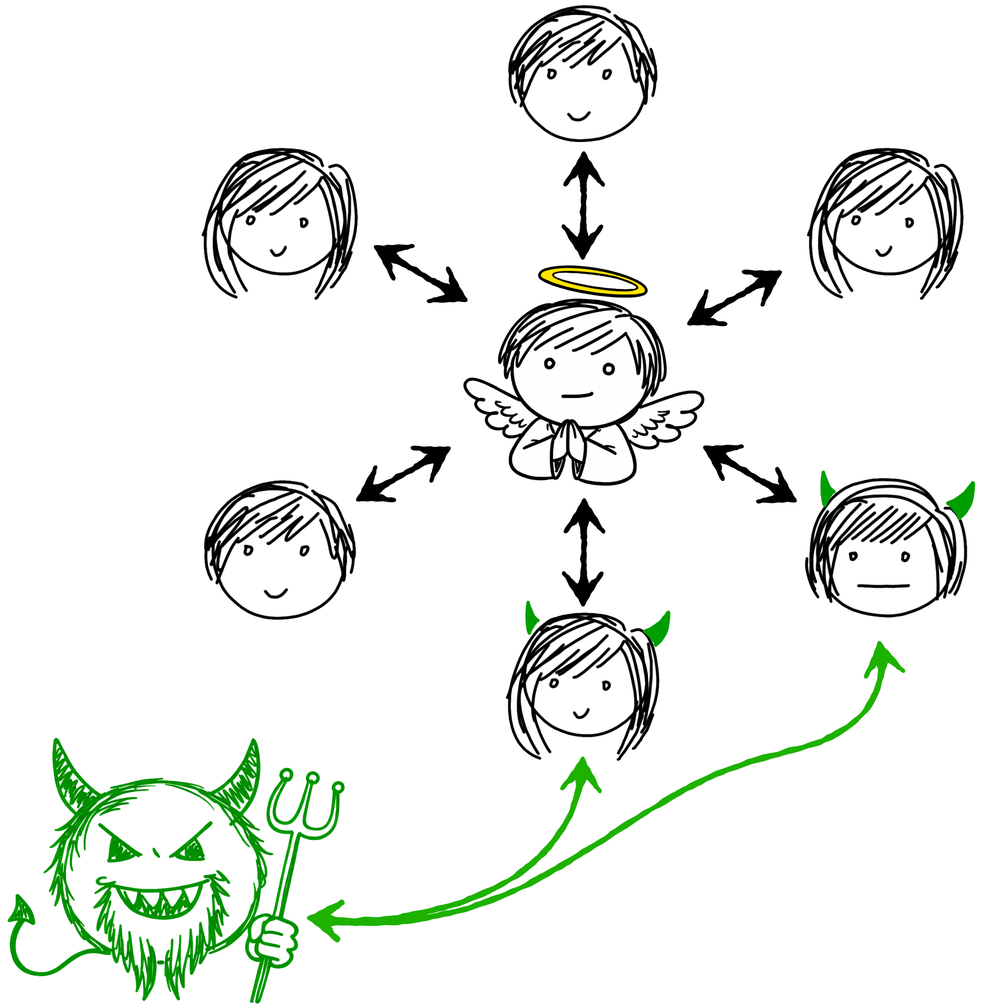
the Simulator $\mathcal{S}$

Goal of $\mathcal{S}$: produce output *indistinguishable* from $\mathcal{A}$

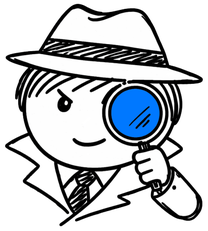


the Adversary $\mathcal{A}$

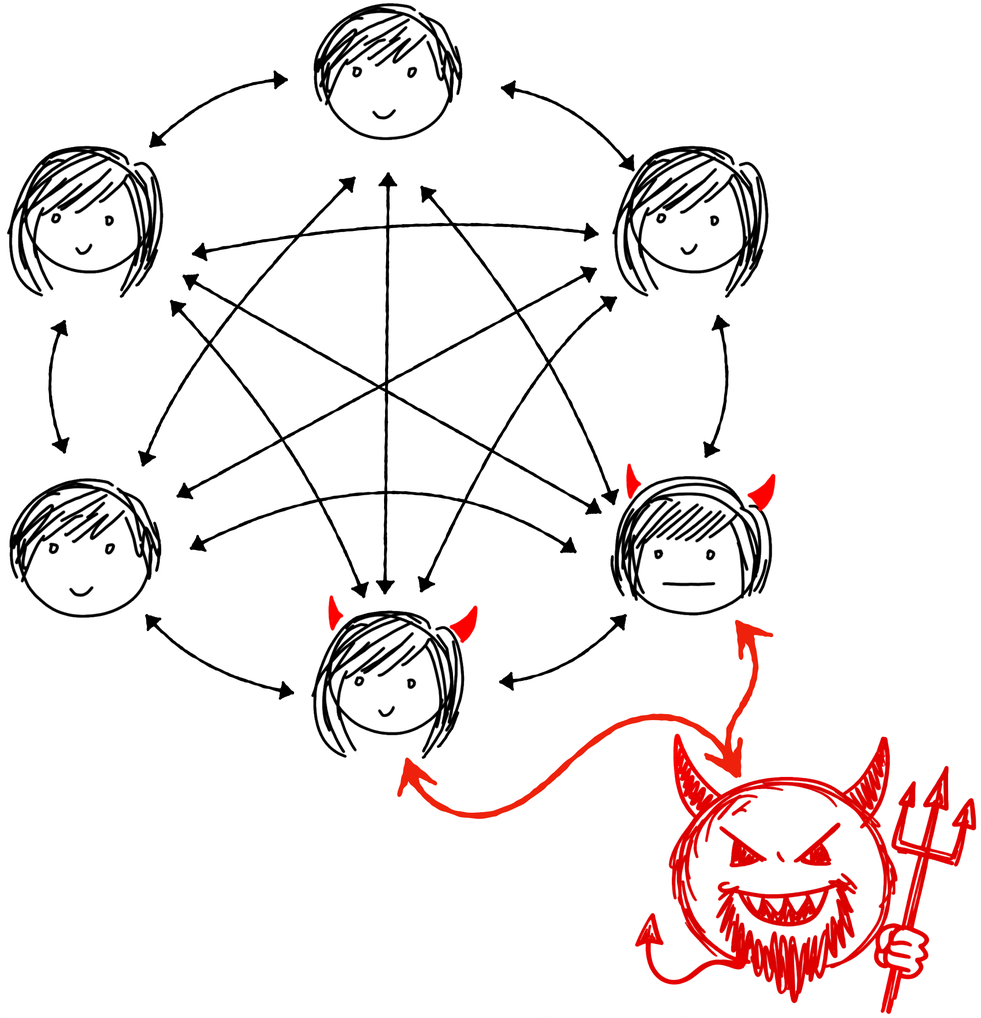
Is $\mathcal{S}$ is Indistinguishable from $\mathcal{A}$? Who will Judge?



the Simulator $\mathcal{S}$



the Distinguisher $\mathcal{D}$

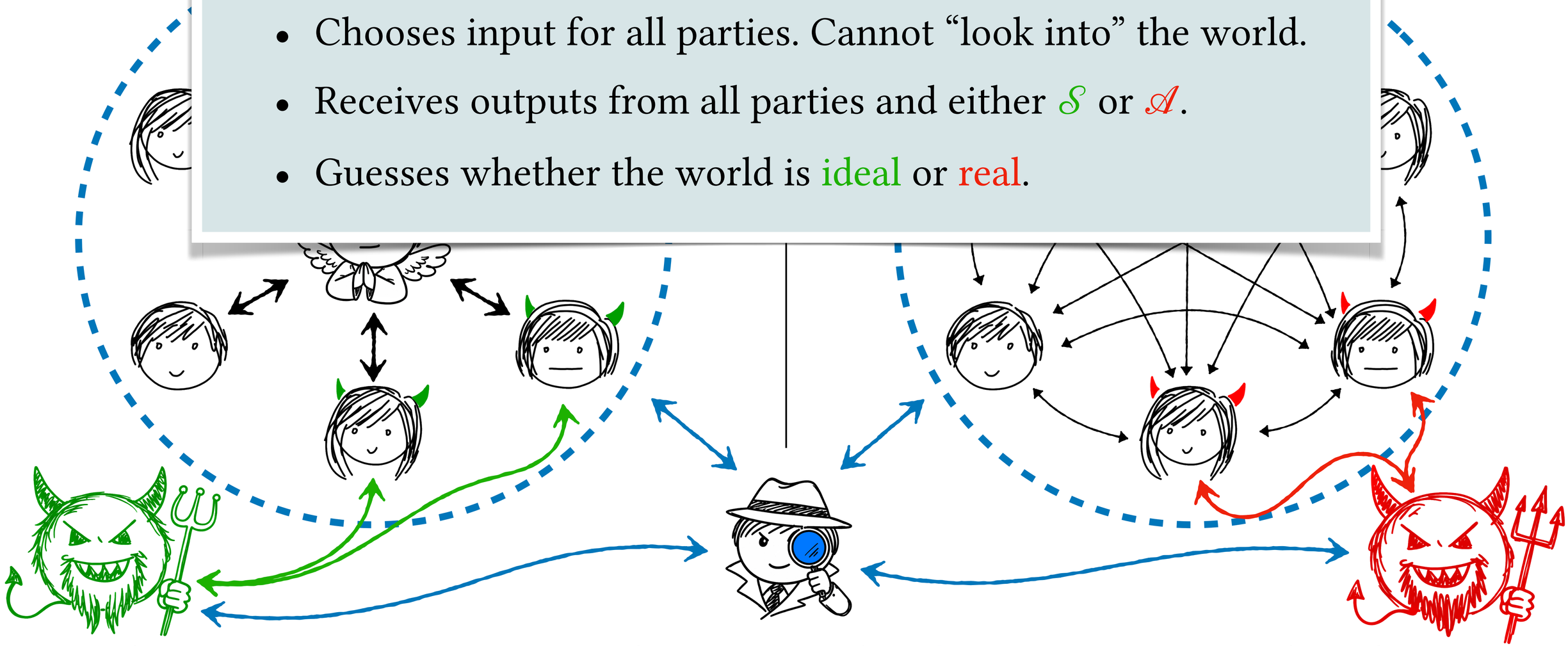


the Adversary $\mathcal{A}$

Is $\mathcal{S}$ is dge?

the Distinguisher $\mathcal{D}$

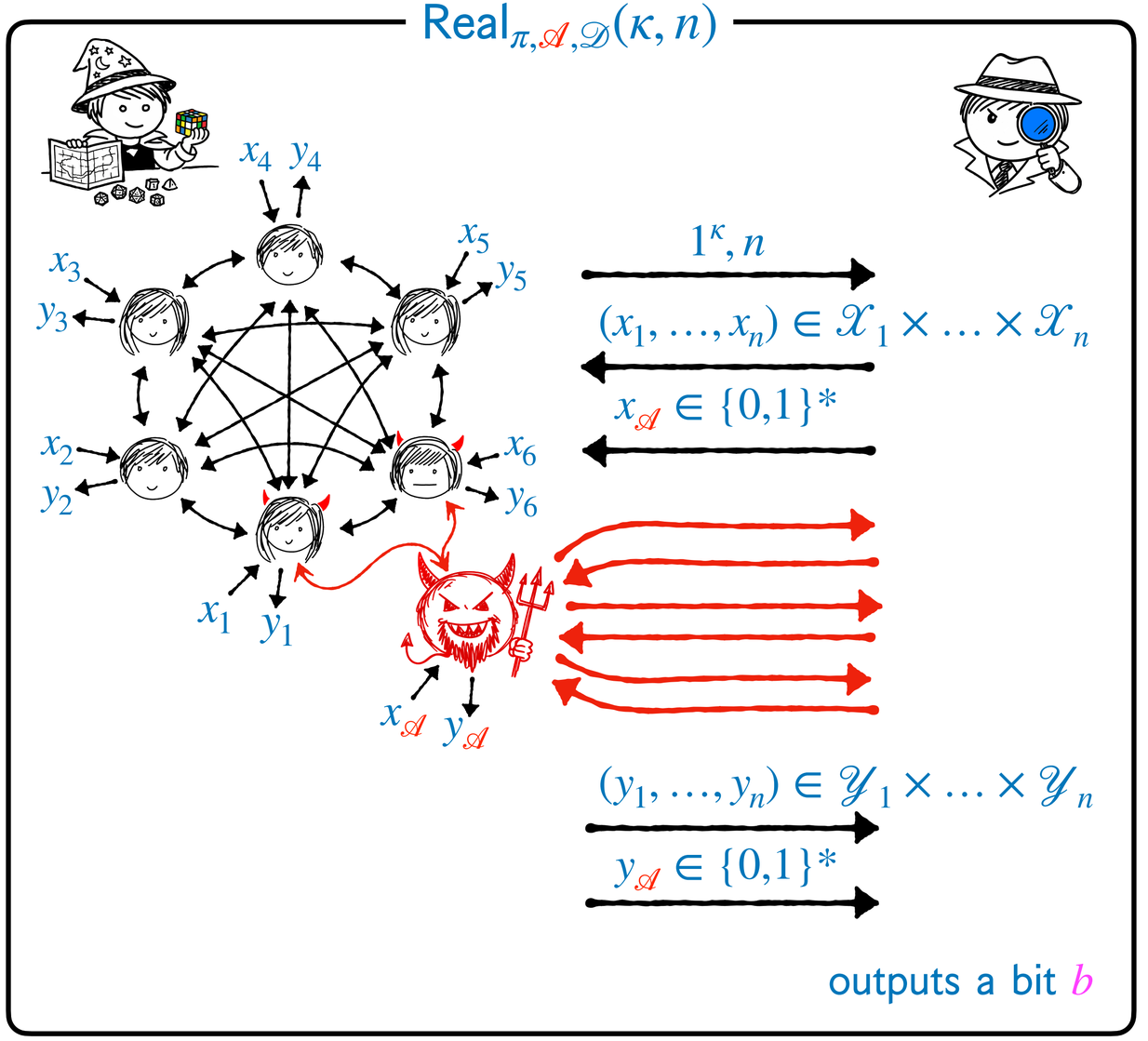
- Interacts with one of the worlds and attempts to determine *which* one by running an experiment.
- Chooses input for all parties. Cannot "look into" the world.
- Receives outputs from all parties and either $\mathcal{S}$ or $\mathcal{A}$.
- Guesses whether the world is ideal or real.



the Simulator $\mathcal{S}$ | the Distinguisher $\mathcal{D}$ | the Adversary $\mathcal{A}$

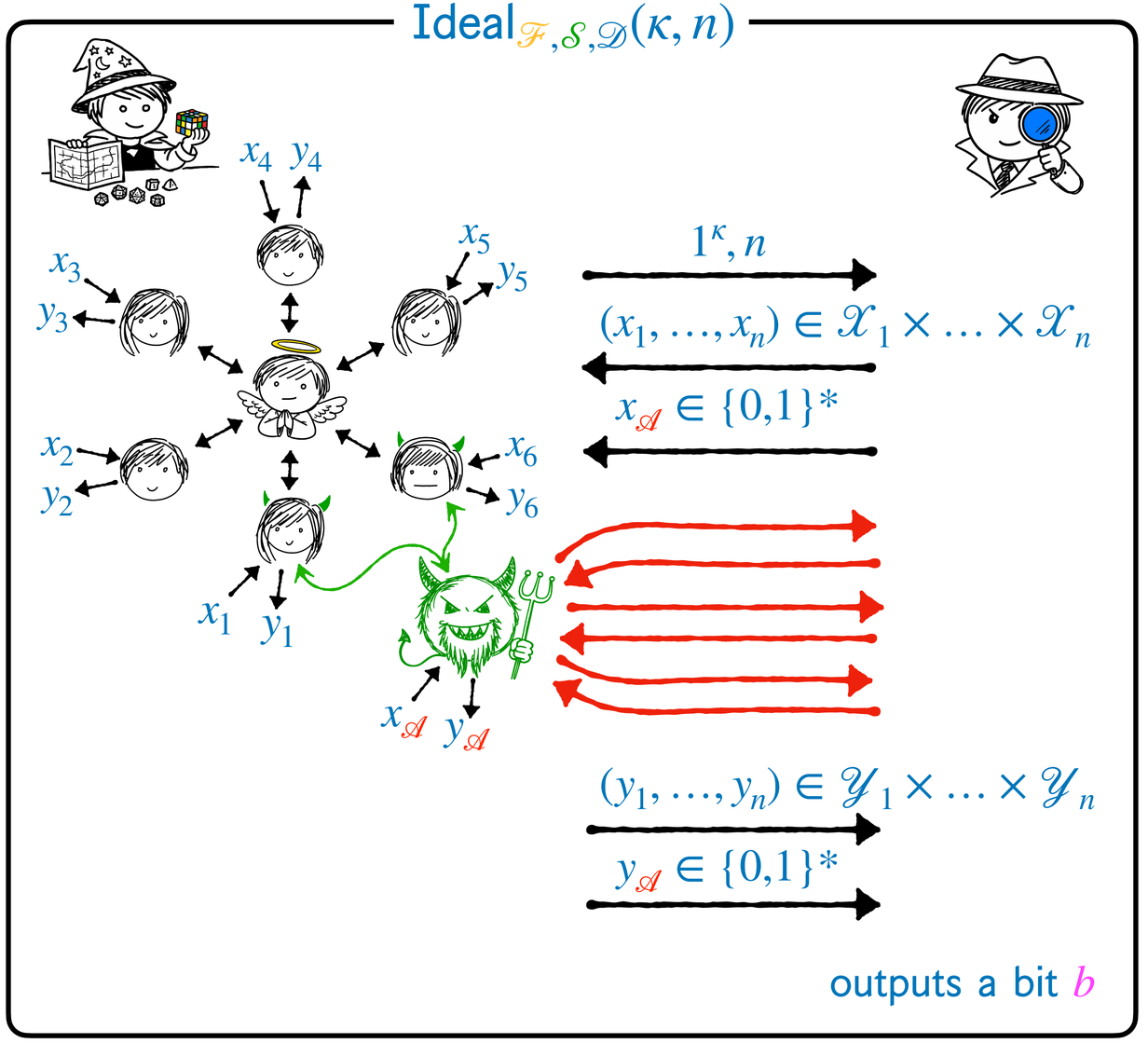
Definition 1: The Real-World Experiment

1. The challenger communicates the parameters.
2. $\mathscr{D}$ supplies inputs for all parties and $\mathscr{A}$.
3. $\mathscr{A}$ corrupts some parties.
4. The protocol runs. During this time, $\mathscr{A}$ fully controls any corrupted parties.
5. $\mathscr{A}$ may send messages to $\mathscr{D}$, and receive responses.
6. The protocol ends when everyone halts with some output (including $\mathscr{A}$). These outputs are sent to $\mathscr{D}$.
7. $\mathscr{D}$ outputs a bit.



Definition 2: The Ideal-World Experiment

1. The challenger communicates the parameters.
2. $\mathscr{D}$ supplies inputs for all parties and $\mathscr{A}$ (but it is received by $\mathscr{S}$).
3. $\mathscr{S}$ corrupts some parties.
4. The "dummy protocol" runs. During this time, $\mathscr{S}$ fully controls any corrupted parties (but they only interact with $\mathscr{F}$).
5. $\mathscr{S}$ may send messages to $\mathscr{D}$ (pretending to be $\mathscr{A}$).
6. The protocol ends when everyone halts with some output (including $\mathscr{S}$). These outputs are sent to $\mathscr{D}$.
7. $\mathscr{D}$ outputs a bit.



Definition 3: Malicious Security

Definition 3. Let $n, t \in \mathbb{N}$ such that $t < n$, let $\mathcal{F}$ be a PPT ideal functionality that interacts with n parties and (possibly) an adversary, and let π be a PPT n-party protocol.

We say that π *realizes* $\mathcal{F}$ in the presence of a *malicious* adversary that statically corrupts up to t parties if for every PPT $\mathcal{A}$ that statically, maliciously corrupts up to t parties, there exists some PPT simulator $\mathcal{S}$, such that for every PPT distinguisher $\mathcal{D}$, there exists a negligible function ε such that for all $\kappa \in \mathbb{N}$

$$\left| \Pr[\mathsf{Real}_{\pi,\mathcal{A},\mathcal{D}}(\kappa, n) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{D}}(\kappa, n) = 1] \right| \leq \varepsilon(\kappa)$$

- For our previous definitions we had some intuition that "anything the adversary could achieve by corrupting the protocol it could also achieve using just the inputs and outputs of the corrupt party (via the Sim algorithm)"
- Now we have something richer! The adversary can *deviate arbitrarily* and the simulator *interacts* with the ideal functionality. Anything anything the adversary could achieve by corrupting the protocol it could also achieve in *the ideal interaction*.

CS4501 Cryptographic Protocols
Lecture 18: OT Extension, Malicious Adversaries

https://jackdoerner.net/teaching/#2026/Spring/CS4501