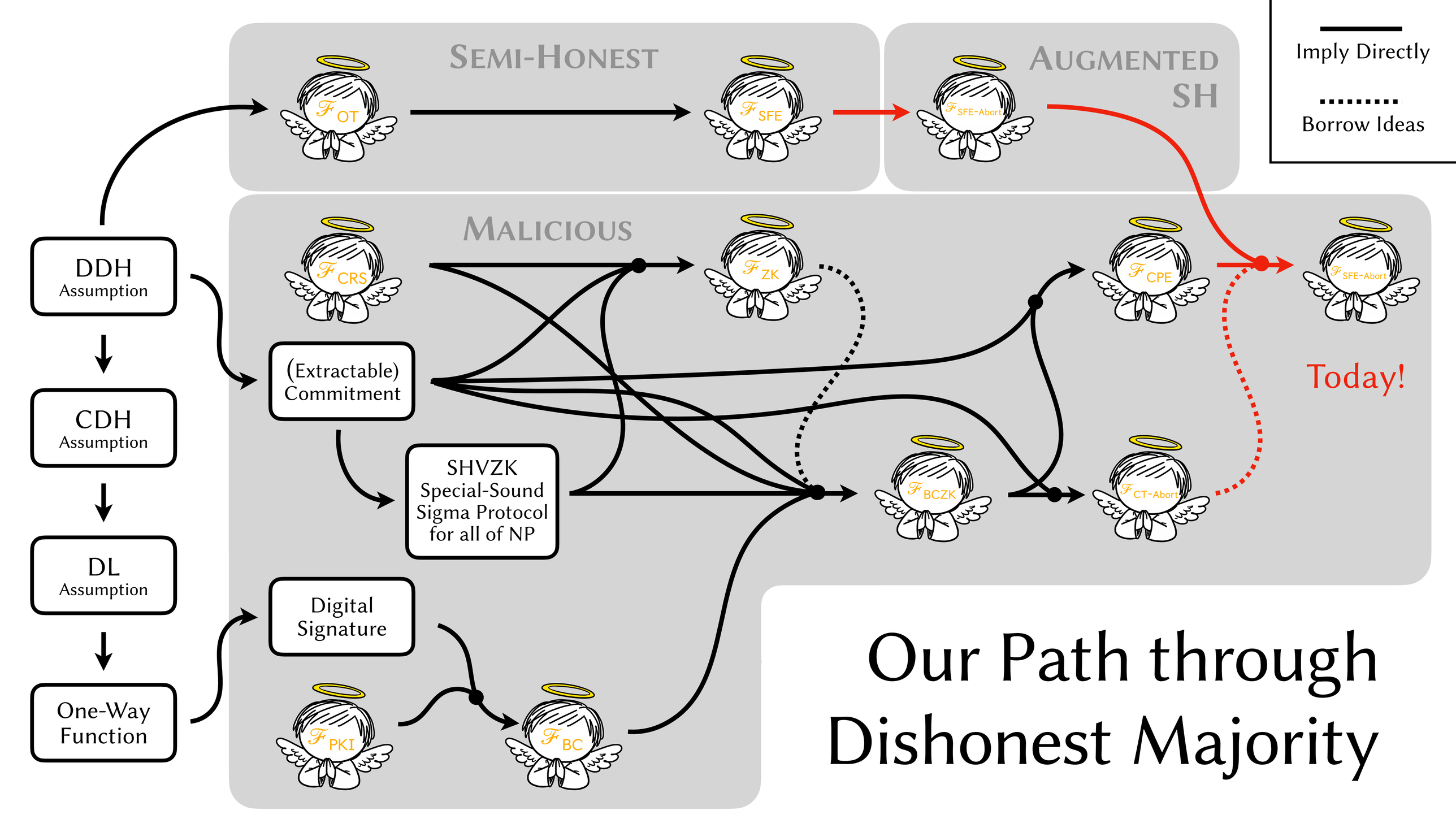
CS4501 Cryptographic Protocols
Lecture 27: The GMW Compiler

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


Imply Directly
Borrow Ideas
Semi-Honest
Augmented SH
Malicious
$\mathcal{F}_{OT}$
$\mathcal{F}_{SFE}$
$\mathcal{F}_{SFE\text{-}Abort}$
$\mathcal{F}_{CRS}$
$\mathcal{F}_{ZK}$
$\mathcal{F}_{CPE}$
$\mathcal{F}_{SFE\text{-}Abort}$
Today!
(Extractable) Commitment
SHVZK Special-Sound Sigma Protocol for all of NP
$\mathcal{F}_{BCZK}$
$\mathcal{F}_{CT\text{-}Abort}$
Digital Signature
$\mathcal{F}_{PKI}$
$\mathcal{F}_{BC}$
DDH Assumption
CDH Assumption
DL Assumption
One-Way Function
Our Path through Dishonest Majority

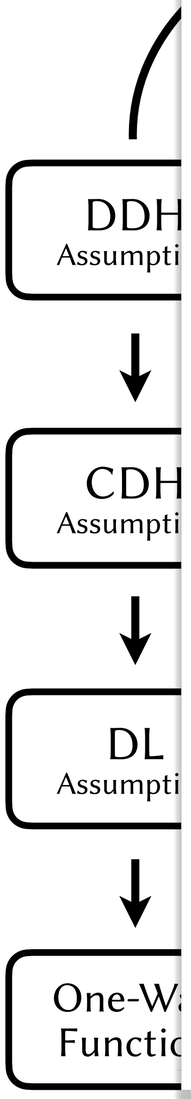
This is not the only way this map could have been drawn,
and these are not the only implications that we know!

I chose this pathway for the course because it makes each
step (individually) simple and it uses only one group assumption.

Actually, *any* assumption that implies OT suffices, and there
are other ways to break up the GMW compiler into individual parts.

There are also other approaches to securely compute any function,
and sometimes special ways to securely compute specific functions.

The important thing is *not* the specific construction that we
learned, but the collection of ideas that went into it!



Directly

ow Ideas

day!

h

y

1. Augmented Semi-Honest Security

Recall: Semi-Honest Security

Definition 1 (Simplified Computational Semi-Honest SFE):

Let $\kappa, n, t \in \mathbb{N}$ such that $t < n$ and n is upper-bounded by some polynomial in κ, let $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$ be a (possibly randomized) n-ary function, and let π be a n-party protocol that runs in PPT relative to κ.

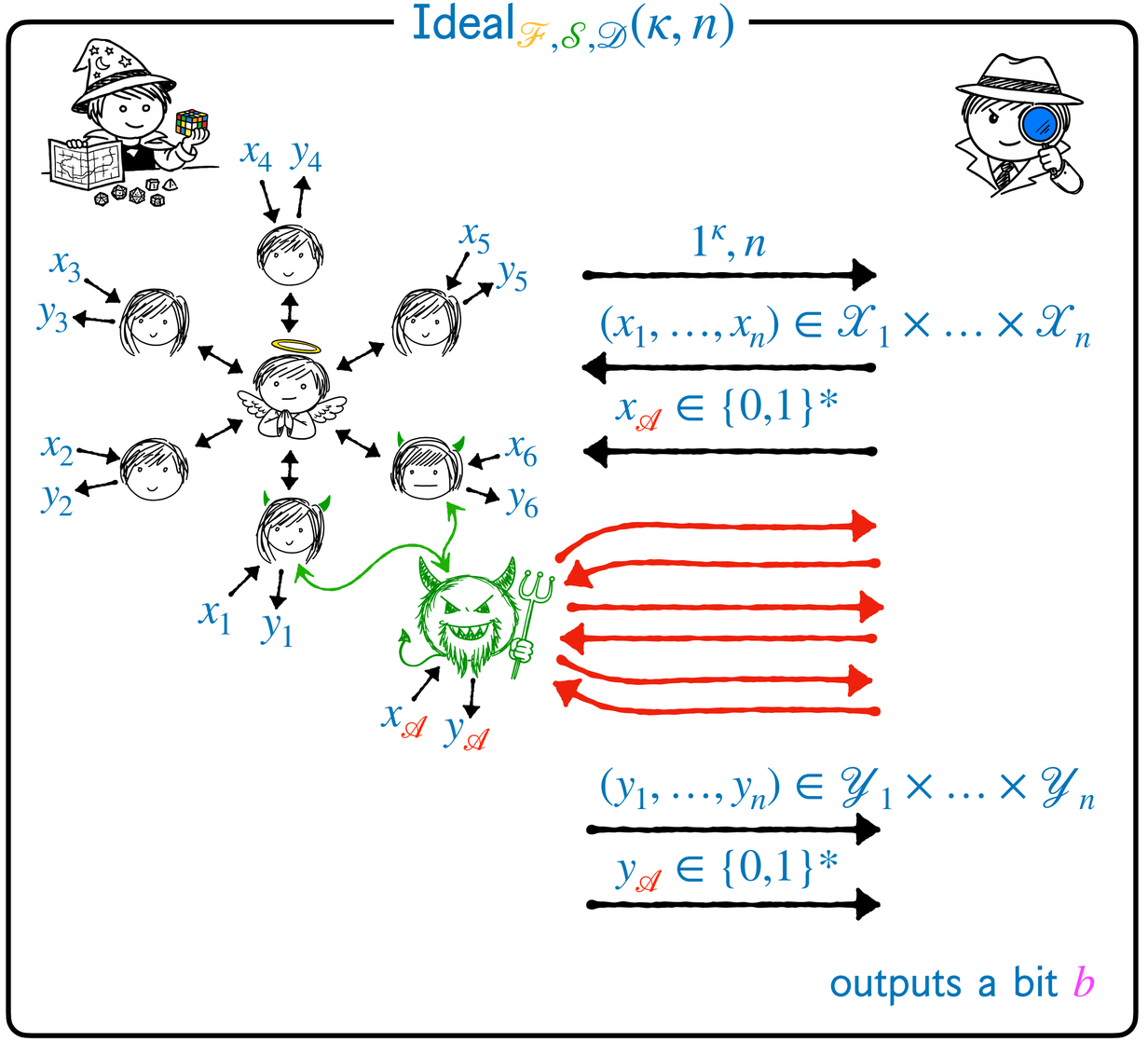
We say that π *securely computes* f in the presence of a bounded semi-honest adversary that statically corrupts up to t parties if there exists a PPT *simulator algorithm* Sim such that for every $I \subseteq [n]$ of size $|I| \leq t$ and every $\vec{x} \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ we have

$$\left\{ \left(\mathsf{Sim}\left(1^\kappa, I, \vec{x}_I, f_I(\vec{x})\right), f(\vec{x}) \right) \right\}_{\kappa \in \mathbb{N}} \approx_c \left\{ (\mathsf{VIEW}_I, \mathsf{OUTPUT}_\pi) \right\}_{\kappa \in \mathbb{N}}$$

Early in the class we handwaved the idea that this simplified definition is related to the *full* semi-honest model. Now we will connect the simplified definition *explicitly* to a model that is neither fully malicious, nor fully semi-honest, but *between* them.

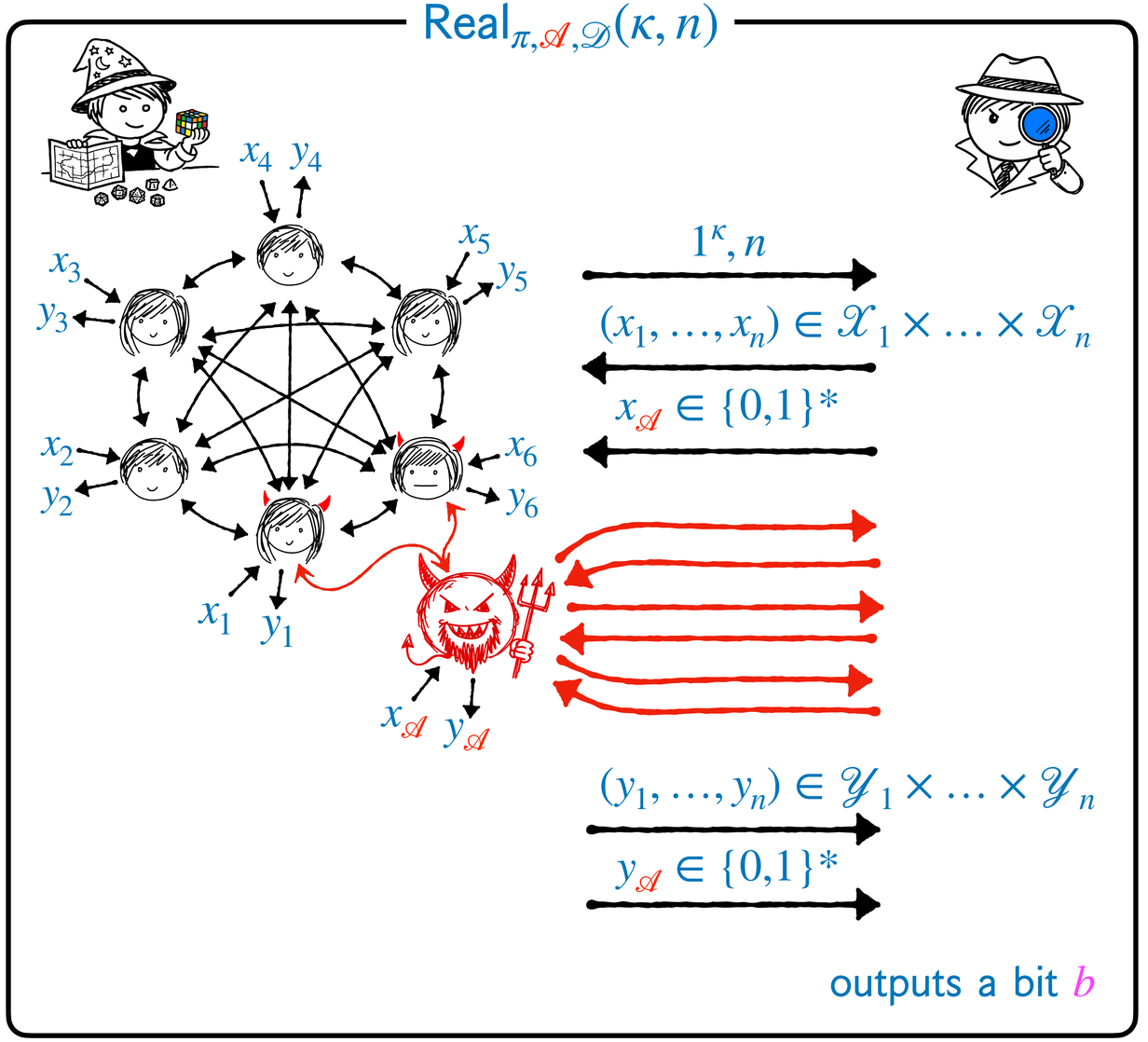
Recall **Def 2**: The Ideal Malicious Experiment

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.



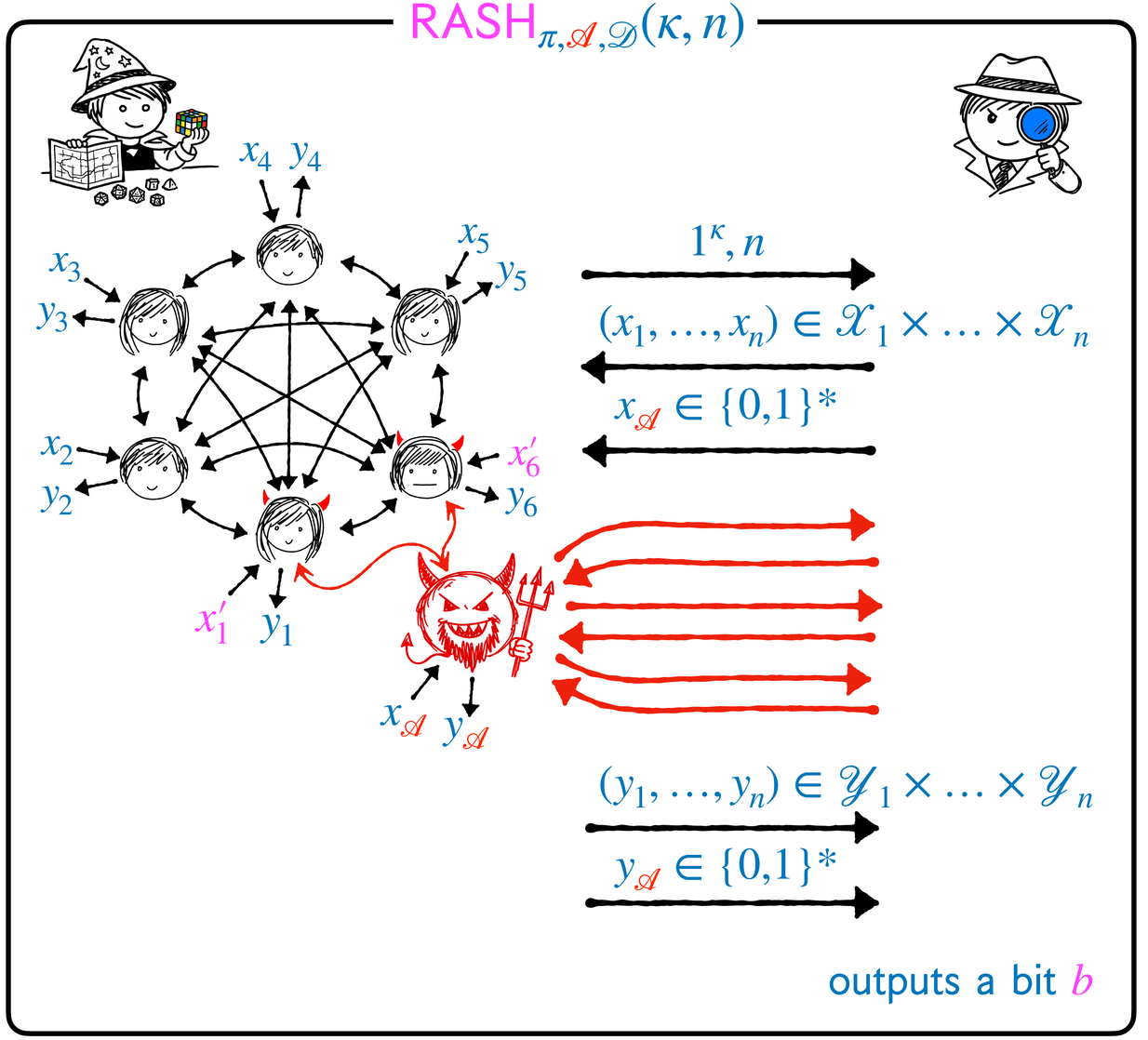
Recall **Def 3**: The Real Malicious Experiment

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 4: The Real Augmented SH Expt.

1. The challenger communicates the parameters.
2. $\mathscr{D}$ supplies inputs for all parties and $\mathscr{A}$.
3. $\mathscr{A}$ corrupts some parties and chooses new inputs for them.
4. The protocol runs *semi-honestly*. During this time, $\mathscr{A}$ may instruct any corrupt party to go silent.
5. $\mathscr{A}$ may send messages to $\mathscr{D}$, and receive responses, but $\mathscr{D}$ cannot adaptively influence the protocol.
6. The protocol ends when everyone halts. The outputs are sent to $\mathscr{D}$.
7. $\mathscr{D}$ outputs a bit.



Augmented Semi-Honest vs Malicious Security

Recall **Definition 5.** Let $\kappa, n, t \in \mathbb{N}$ such that $t < n$ and n is upper-bounded by some polynomial in κ, 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 such PPT $\mathcal{A}$, 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 $\kappa \in \mathbb{N}$

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

Definition 6. Let $\kappa, n, t \in \mathbb{N}$ such that $t < n$ and n is upper-bounded by some polynomial in κ, 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 an *augmented semi-honest* adversary that statically corrupts up to t parties if for every such PPT $\mathcal{A}$ 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 $\kappa \in \mathbb{N}$

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

Augmented Semi-Honest vs Malicious Security

Definition 6. Let $\kappa, n, t \in \mathbb{N}$ such that $t < n$ and n is upper-bounded by some polynomial in κ, 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 an *augmented semi-honest* adversary that statically corrupts up to t parties if for every such PPT $\mathcal{A}$ 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 $\kappa \in \mathbb{N}$

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

Notice: the *ideal* augmented semi-honest experiment is *exactly the same* as the ideal malicious experiment. Because the parties in the ideal experiment only communicate with $\mathcal{F}$, their malicious attack strategies are limited to changing their inputs or failing to communicate.

It is *not* immediate that security under the simplified Semi-Honest definition implies security under this one, or vice versa, although they do *often* coincide. Counterexamples are interesting!

We will prove that our simplified SH definition implies this one for a *specific* kind of protocol.

Augmented Semi-Honest from Simplified SH

Definition 7 ($\mathcal{F}^f_{\text{SFE-Abort}}$): This functionality is like $\mathcal{F}^f_{\text{SFE}}$, except that after computing $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n)$, it sends $\{y_i\}_{i \in I}$ to $\mathcal{S}$, where I indexes the corrupt parties, and allows $\mathcal{S}$ to determine whether $\mathcal{F}^f_{\text{SFE-Abort}}$ delivers the outputs, or aborts by delivering $\perp$ to all parties.

Theorem 1: Let $\kappa, n, t \in \mathbb{N}$ such that $t < n$ and n is upper-bounded by some polynomial in κ, and let $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$ be any (possibly randomized) n-ary function such that $\mathcal{Y}_i$ is an abelian group with respect to the $\oplus$ operation, for every $i \in [n]$.

Consider $g(x_1, \ldots, x_n) = ((\langle y_1 \rangle_1, \ldots, \langle y_n \rangle_1), \ldots, (\langle y_1 \rangle_n, \ldots, \langle y_n \rangle_n))$, where $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n)$ and $\langle y_i \rangle \leftarrow \mathcal{Y}_i^n$ such that $\langle y_i \rangle_1 \oplus \ldots \oplus \langle y_i \rangle_n = y_i$ for every $i \in [n]$.

Suppose that there exists some protocol π_g that security computes g in the presence of a semi-honest adversary that statically corrupts up to t parties, according to the *simplified* definition.

Then there exists a protocol π_f that realizes $\mathcal{F}^f_{\text{SFE-Abort}}$ in the presence of an *augmented* semi-honest adversary that that statically corrupts up to t parties.

Augmented Semi-Honest from Simplified SH

Proof Sketch: Let us define the protocol π_f, which uses π_g as a subroutine.

1. The parties run π_g and every P_i receives $(\langle y_1 \rangle_i, \ldots, \langle y_n \rangle_i)$.
2. For every $j \in [n] \backslash \{i\}$, P_i sends $\langle y_j \rangle_i$ to P_j.
3. P_i computes $y_i := \langle y_i \rangle_1 \oplus \ldots \oplus \langle y_i \rangle_n$ and outputs y_i.

We construct the (interactive) simulator $\mathcal{S}_f$ for π_f as follows:

1. As usual, $\mathcal{S}_f$ emulates an instance of the real-world augmented semi-honest experiment involving $\mathcal{A}$. After $\mathcal{A}$ chooses a set $I \subset [n]$ of parties to corrupt, and chooses new inputs $\{x_i'\}_{i \in I}$ for those corrupt parties, $\mathcal{S}_f$ corrupts the corresponding parties in the ideal world.

2. For $i \in I$, $\mathcal{S}_f$ samples $(\langle \hat{y}_1 \rangle_i, \ldots, \langle \hat{y}_n \rangle_i) \leftarrow \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$.

Augmented Semi-Honest from Simplified SH

2. For $i \in I$, $\mathcal{S}_f$ samples $(\langle \hat{y}_1 \rangle_i, \ldots, \langle \hat{y}_n \rangle_i) \leftarrow \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$.

3. By the simplified semi-honest security of π_g, there must exist some algorithm Sim_g such that

$$\left\{ \left(\mathsf{Sim}_g \left(1^\kappa, I, \vec{x}_I, g_I(\vec{x}) \right), g(\vec{x}) \right) \right\}_{\kappa \in \mathbb{N}} \approx_c \left\{ (\mathsf{VIEW}_I, \mathsf{OUTPUT}_{\pi_g}) \right\}_{\kappa \in \mathbb{N}}$$

   $\mathcal{S}_f$ computes the corrupt views $\{\mathsf{view}_i\}_{i \in I} \leftarrow \mathsf{Sim}_g \left(1^\kappa, I, \{x'_i\}_{i \in I}, \left\{ (\langle \hat{y}_1 \rangle_i, \ldots, \langle \hat{y}_n \rangle_i) \right\}_{i \in I} \right)$. Each view contains a random tape and a set of messages received from honest parties.

   $\mathcal{S}_f$ uses this information to emulate the semi-honest execution of π_g among the corrupt parties. $\mathcal{A}$ witnesses this execution, but cannot alter it. After any new item is added to a corrupt party's view, $\mathcal{A}$ may indicate that the party should halt, at which point $\mathcal{S}_f$ ceases emulating π_g, and halts (after $\mathcal{A}$ does).

4. If the emulation of π_g reaches the end without halting, then $\mathcal{S}_f$ instructs the ideal-world corrupt parties to send their *new* inputs $\{x'_i\}_{i \in I}$ to $\mathcal{F}^f_{\mathsf{SFE-Abort}}$.

Augmented Semi-Honest from Simplified SH

4. If the emulation of π_g reaches the end without halting, then $\mathcal{S}_f$ instructs the ideal-world corrupt parties to send their *new* inputs $\{x_i'\}_{i \in I}$ to $\mathcal{F}^f_{\text{SFE-Abort}}$.

5. $\mathcal{S}_f$ receives $\{y_i\}_{i \in I}$ *directly* from $\mathcal{F}^f_{\text{SFE-Abort}}$. Note that these outputs are consistent with the new inputs of the corrupt parties.

6. For every $i \in I$, $\mathcal{S}_f$ finds $\langle y_i \rangle \in \mathcal{Y}_i^n$ such that $\langle y_i \rangle_1 \oplus \ldots \oplus \langle y_i \rangle_n = y_i$ and $\langle y_i \rangle_j = \langle \hat{y}_i \rangle_j$ for every $j \in I$. In other words, $\mathcal{S}_f$ *equivocates* the simulated XOR shares.

   For every $i \in I$ and $h \in [n] \backslash I$, $\mathcal{S}_f$ sends $\langle y_i \rangle_h$ to P_i in the emulated experiment on behalf of P_h.

7. If $\mathcal{S}_f$ receives $\langle y_h \rangle_i$ from P_i on behalf of P_h in the emulated experiment for every $i \in I$ and $h \in [n] \backslash I$, then $\mathcal{S}_f$ sends "OK" to $\mathcal{F}^f_{\text{SFE-Abort}}$. Otherwise, $\mathcal{S}_f$ sends "Abort".

   Note that for each $h \in [n] \backslash I$, either $\langle y_h \rangle_i$ will be received for all $i \in I$, or for none, because $\mathcal{A}$ is only permitted to make a party go completely silent, and cannot send messages selectively.

Augmented Semi-Honest from Simplified SH

- Although the simulator we just saw seemed complex, the security argument is quite simple.
- Notice that the *only* distributional difference between the simulation and the real world view is that the parties run π_g in the real world, whereas in the simulation, $\mathcal{S}_f$ uses Sim_g. The shares in the final round have *exactly* the same distribution in both worlds.
- Thus it follows directly from the semi-honest security of π_g that π_f realizes $\mathcal{F}_{\text{SFE-Abort}}^{f}$. ∎

Corollary 1: Let $n, t \in \mathbb{N}$ such that $t < n$. If the Decisional Diffie-Hellman Assumption is true, then there exists an n-party protocol that realizes $\mathcal{F}_{\text{SFE-Abort}}^{f}$ for any randomized, polynomial-time function f in the presence of an *augmented* semi-honest adversary that statically corrupts up to t parties.

Proof: By Theorem 1, plus the semi-honest dishonest-majority feasibility theorem from Lecture 15.

That is, we can use the semi-honest GMW (or BMR) protocol to construct the required protocol π_g. ∎

Notice: the corollary gives us a *plain model* protocol! This will be important soon!

2. The GMW Compiler for Dishonest Majorities

GMW Compiler Intuition

- How will our compiler work? We start with a semi-honest protocol and change it:
- The internal state of every party will always be committed publicly, starting with the inputs it chooses.
- Every party contributes a share of every other party's randomness. They all commit these shares, and then everyone except the recipient releases them, in order to ensure that the result is free of bias.
- Any time a party is supposed to send a message, it must prove to everyone that it derived the message correctly from its publicly committed values, and that it broadcasted an encryption of that message that only the designated recipient can decrypt. Any message not delivered in this way is ignored.
- The attacks that remain after these changes are the ones that involve changing the input before it is committed, or failing to speak at all. These are exactly the attacks that we reason about in the *augmented* semi-honest model.

Inputs for our Compiler

- Previously we constructed a compiler for *sigma protocols*, which have a simple, fixed structure.
- The GMW compiler will take *any* protocol as input. We need a way to represent protocols in our model so that the compiler can modify them generically.
- We assume that protocols occur in synchronous rounds, and parties sample all the randomness they will ever need at the beginning of the protocol. In any round, a party's message is a *deterministic* function of its input, its randomness, and the messages it previously received.
- Suppose we have a ρ-round n-party input protocol π_{in}. Let x_i and r_i be the input and randomness of P_i, and let $m^s_{i\to j}$ be the message from P_i to P_j in round s. For every $i, j \in [n]$ we can define the function $\mathsf{nextmsg}_{i\to j}$ of π_{in} such that for every $s \in [\rho]$,

$$m^s_{i\to j} = \mathsf{nextmsg}_{i\to j}\left(x_i, r_i, \left\{m^1_{k\to i}\right\}_{k\in[n]\setminus\{i\}}, \ldots, \left\{m^{s-1}_{k\to i}\right\}_{k\in[n]\setminus\{i\}}\right)$$

- Furthermore, $m^\rho_{i\to i}$ is defined to be the output of P_i, and $m^s_{i\to i} = \lambda$ if $s < \rho$.
- Let $\widetilde{\mathsf{nextmsg}}_{i\to j}$ be similar to $\mathsf{nextmsg}_{i\to j}$, except that it takes $\{\tilde{r}_{k\to i}\}_{k\in[n]}$ in place of r_i, and computes $r_i := \tilde{r}_{1\to i} \oplus \ldots \oplus \tilde{r}_{n\to i}$ internally.
- Now imagine representing $\widetilde{\mathsf{nextmsg}}_{i\to j}$ for round s as a boolean circuit. We'll call this circuit $C^s_{i\to j}$.

Commit-and-Privately-Evaluate Functionality

Recall **Definition 8 ($\mathcal{F}_{\text{CPE}}$):**

This functionality interacts with n parties, $P_1, \ldots, P_n$.

- At initialization time, $\mathcal{F}_{\text{CPE}}$ sets $W_i := \lambda$ in memory for every $i \in [n]$.
- Upon receiving (commit, w) from P_i, where $w \in \{0,1\}^*$, $\mathcal{F}_{\text{CPE}}$ updates $W_i := W_i \| w$ in memory and sends $(\text{received}, i)$ to all other parties.
- Upon receiving (eval, j, C) from P_i, where $j \in [n] \backslash \{i\}$ and C is description of a boolean circuit that takes $|W_i|$ bits of input:
  1. $\mathcal{F}_{\text{CPE}}$ sends $(\text{image}, i, C, C(W_i))$ to P_j.
  2. $\mathcal{F}_{\text{CPE}}$ sends $(\text{evaluated}, i, j, C)$ to every P_k such that $k \in [n] \backslash \{i, j\}$.
  3. $\mathcal{F}_{\text{CPE}}$ updates $W_j := W_j \| C(W_i)$ in memory.

Notice that received evaluations accumulate to the witness, just like received messages in a protocol accumulate to the state of a party!

GMW Compiler in the $\mathcal{F}_{\mathsf{CPE}}$ Hybrid Model

Phase 1: Input Commitment:

1. Every P_i with input x_i and sends (commit, x_i) to $\mathcal{F}_{\mathsf{CPE}}$. Everyone else receives $(\mathsf{received}, i)$.

Phase 2: Coin Tossing. For every $j \in [n]$:

2. Every P_i samples $\tilde{r}_{i\to j}$ and sends $(\mathsf{commit}, \tilde{r}_{i\to j})$ to $\mathcal{F}_{\mathsf{CPE}}$. Everyone else receives $(\mathsf{received}, i)$.
3. Every P_i such that $i \neq j$ sends $(\mathsf{eval}, j, C_{\mathsf{Identity}(|\tilde{r}_{i\to j}|)})$ to $\mathcal{F}_{\mathsf{CPE}}$, where $C_{\mathsf{Identity}(\ell)}$ is a circuit that outputs its last ℓ input bits. P_j receives $(\mathsf{image}, i, C_{\mathsf{Identity}(|\tilde{r}_{i\to j}|)}, \tilde{r}_{i\to j})$, and everyone else receives $(\mathsf{evaluated}, i, j, C_{\mathsf{Identity}(|\tilde{r}_{i\to j}|)})$.

Phase 3: For round $s \in [\rho]$:

4. Every P_i sends $(\mathsf{eval}, j, C^s_{i\to j})$ to $\mathcal{F}_{\mathsf{CPE}}$, for all $j \in [n]\backslash\{i\}$. Each P_j receives $(\mathsf{image}, i, C^s_{i\to j}, m^s_{i\to j})$, and everyone else receives $(\mathsf{evaluated}, i, j, C^s_{i\to j})$.
5. Every P_i computes $m^s_{i\to i} := \widetilde{\mathsf{nextmsg}}_{i\to i}\left(x_i, \left\{\tilde{r}_{k\to i}\right\}_{k\in[n]}, \left\{m^1_{k\to i}\right\}_{k\in[n]\backslash\{i\}}, \ldots, \left\{m^{s-1}_{k\to i}\right\}_{k\in[n]\backslash\{i\}}\right)$ and if $m^s_{i\to i} \neq \lambda$, then P_i outputs $m^s_{i\to i}$ and halts.

GMW Compiler in the $\mathscr{F}_{\mathsf{CPE}}$ Hybrid Model

2. Every P_i samples $\tilde{r}_{i\to j}$ and sends $(\mathsf{commit}, \tilde{r}_{i\to j})$ to $\mathscr{F}_{\mathsf{CPE}}$. Everyone else receives $(\mathsf{received}, i)$.
3. Every P_i such that $i \neq j$ sends $(\mathsf{eval}, j, C_{\mathsf{Identity}(|\tilde{r}_{i\to j}|)})$ to $\mathscr{F}_{\mathsf{CPE}}$, where $C_{\mathsf{Identity}(\ell)}$ is a circuit that outputs its last ℓ input bits. P_j receives $(\mathsf{image}, i, C_{\mathsf{Identity}(|\tilde{r}_{i\to j}|)}, \tilde{r}_{i\to j})$, and everyone else receives $(\mathsf{evaluated}, i, j, C_{\mathsf{Identity}(|\tilde{r}_{i\to j}|)})$.

Phase 3: For round $s \in [\rho]$:

4. Every P_i sends $(\mathsf{eval}, j, C^s_{i\to j})$ to $\mathscr{F}_{\mathsf{CPE}}$, for all $j \in [n]\backslash\{i\}$. Each P_j receives $(\mathsf{image}, i, C^s_{i\to j}, m^s_{i\to j})$, and everyone else receives $(\mathsf{evaluated}, i, j, C^s_{i\to j})$.
5. Every P_i computes $m^s_{i\to i} := \widetilde{\mathsf{nextmsg}}_{i\to i}\left(x_i, \left\{\tilde{r}_{k\to i}\right\}_{k\in[n]}, \left\{m^1_{k\to i}\right\}_{k\in[n]\backslash\{i\}}, \ldots, \left\{m^{s-1}_{k\to i}\right\}_{k\in[n]\backslash\{i\}}\right)$ and if $m^s_{i\to i} \neq \lambda$, then P_i outputs $m^s_{i\to i}$ and halts.

Aborts:

6. If at any point in the preceding three phases, P_i expects to receive a particular message from $\mathscr{F}_{\mathsf{CPE}}$, and that message does not arrive, or it contains an incorrect circuit description, then P_i immediately aborts by outputting $\perp$ and halting.

Proving Security for the GMW Compiler

Theorem 2: Let $\kappa, n, t \in \mathbb{N}$ such that $t < n$ and n is upper-bounded by some polynomial in κ, let π_{in} be any plain-model protocol for n parties, and let π_{out} be a GMW-compiled version of π_{in}. For every malicious PPT $\mathscr{A}$ that statically corrupts up to t parties, there exists an *augmented* semi-honest PPT $\mathscr{A}'$ that statically corrupts up to t parties, such that for every PPT $\mathscr{D}$ we have

$$\Pr[\mathsf{Real}_{\pi_{\mathsf{out}},\mathscr{A},\mathscr{D}}(\kappa, n) = 1] = \Pr[\mathsf{RASH}_{\pi_{\mathsf{in}},\mathscr{A}',\mathscr{D}}(\kappa, n) = 1]$$

Notice: This is not a Real-Ideal proof! What is going on?

> We're proving that for every malicious attack on the compiled protocol, we can find an equivalent augmented semi-honest attack on the uncompiled protocol.

Proof Sketch: $\mathscr{A}'$ internally emulates an instance of the real-world experiment involving $\mathscr{A}$ and π_{out}, in which $\mathscr{A}'$ plays the role of the honest parties, the distinguisher, and $\mathscr{F}_{\mathsf{CPE}}$.

1. When $\mathscr{A}$ indicates which parties it wishes to corrupt, $\mathscr{A}'$ corrupts the corresponding parties in the RASH experiment, retrieves their original inputs $\{x_i\}_{i \in I}$, and embeds those inputs in the emulated real experiment.
2. **Phase 1:** When each corrupt party P_i commits an input x_i' in the emulated real experiment, $\mathscr{A}'$ declares in the RASH experiment that it will adjust the input of P_i to x_i'.

Proving Security for the GMW Compiler

2. **Phase 1:** When each corrupt party P_i commits an input x_i' in the emulated real experiment, $\mathcal{A}'$ declares in the RASH experiment that it will adjust the input of P_i to x_i'.

3. **Phase 2:** When for some $j \in I$, every corrupt P_i commits to $\tilde{r}_{i\to j}$, $\mathcal{A}'$ retrieves r_j from the view of P_j in the RASH experiment, and samples $\tilde{r}_{h\to j}$ for every $h \in [n]\backslash I$ such that $\tilde{r}_{1\to j} \oplus \ldots \oplus \tilde{r}_{n\to j} = r_j$. These values are opened via $\mathcal{F}_{\mathsf{CPE}}$ on behalf of the honest parties, when appropriate.

4. Note that it is trivial to simulate any instance when an honest party is expected to commit to a value in the first two phases.

5. **Phase 3:** For each round $s \in [\rho]$:

   a. For every $i \in I$ and $j \in [n]\backslash I$, $\mathcal{A}'$ retrieves $m^s_{j\to i}$ from the view of the corrupt P_i in the RASH experiment, and sends $(\mathsf{image}, j, C^s_{j\to i}, m^s_{j\to i})$ to the corrupt P_i in the emulated real experiment, on behalf of $\mathcal{F}_{\mathsf{CPE}}$. It sends $(\mathsf{evaluated}, j, i, C^s_{j\to i})$ to the other emulated parties.

   b. For every $i \in I$, when corrupt P_i in the emulated real world experiment sends $(\mathsf{eval}, j, C^s_{i\to j})$ to $\mathcal{F}_{\mathsf{CPE}}$ for some $j \in [n]\backslash\{i\}$, $\mathcal{A}'$ computes $m^s_{i\to j}$ using $C^s_{i\to j}$ on x_i', r_i, and $\{m^{s'}_{k\to i}\}_{k\in[n]\backslash\{i\}, s'\in[s-1]}$. If $j \in I$, then $\mathcal{A}'$ sends $(\mathsf{image}, j, C^s_{i\to j}, m^s_{i\to j})$ to corrupt P_j in the emulated real experiment. It sends $(\mathsf{evaluated}, i, j, C^s_{i\to j})$ to the other emulated corrupt parties.

Proving Security for the GMW Compiler

5. **Phase 3:** For each round $s \in [\rho]$:
   a. For every $i \in I$ and $j \in [n] \backslash I$, $\mathscr{A}'$ retrieves $m^s_{j \to i}$ from the view of the corrupt P_i in the RASH experiment, and sends $(\mathsf{image}, j, C^s_{j \to i}, m^s_{j \to i})$ to the corrupt P_i in the emulated real experiment, on behalf of $\mathscr{F}_{\mathsf{CPE}}$. It sends $(\mathsf{evaluated}, j, i, C^s_{j \to i})$ to the other emulated parties.
   b. For every $i \in I$, when corrupt P_i in the emulated real world experiment sends $(\mathsf{eval}, j, C^s_{i \to j})$ to $\mathscr{F}_{\mathsf{CPE}}$ for some $j \in [n] \backslash \{i\}$, $\mathscr{A}'$ computes $m^s_{i \to j}$ using $C^s_{i \to j}$ on x'_i, r_i, and $\{m^{s'}_{k \to i}\}_{k \in [n] \backslash \{i\}, s' \in [s-1]}$. If $j \in I$, then $\mathscr{A}'$ sends $(\mathsf{image}, j, C^s_{i \to j}, m^s_{i \to j})$ to corrupt P_j in the emulated real experiment. It sends $(\mathsf{evaluated}, i, j, C^s_{i \to j})$ to the other emulated corrupt parties.
   c. For every $i \in I$, after corrupt P_i in the emulated real world experiment sends $(\mathsf{eval}, j, C^s_{i \to j})$ to $\mathscr{F}_{\mathsf{CPE}}$ for *every* $j \in [n] \backslash \{i\}$, where $C^s_{i \to j}$ describes the expected circuit, $\mathscr{A}'$ permits P_i to speak and/or produce output in round of the RASH experiment.
6. **Aborts:** If at any point a corrupt P_i in the emulated real world experiment sends the wrong circuit description, or fails entirely to make a commitment or evaluation via $\mathscr{F}_{\mathsf{CPE}}$ when expected, $\mathscr{A}'$ forces P_i to go silent in the RASH experiment afterward.

Proving Security for the GMW Compiler

6. **Aborts:** If at any point a corrupt P_i in the emulated real world experiment sends the wrong circuit description, or fails entirely to make a commitment or evaluation via $\mathcal{F}_{\mathsf{CPE}}$ when expected, $\mathcal{A}'$ forces P_i to go silent in the RASH experiment afterward.

Much like in our proof of Theorem 1, the adversary $\mathcal{A}'$ that we have described is complex, but we have a very simple sketch of an argument that the distribution it produces in $\mathsf{RASH}_{\pi_{\mathsf{in}},\mathcal{A}',\mathcal{D}}(\kappa, n)$ is identical to the distribution that $\mathcal{A}$ produces in $\mathsf{Real}_{\pi_{\mathsf{out}},\mathcal{A},\mathcal{D}}(\kappa, n)$.

In particular, observe that in both experiments the corrupt parties' views are identically distributed (that is, the new inputs x_i' for $i \in I$ are the ones determined by $\mathcal{A}$, and r_i is uniform), and the messages they send are forced to be computed deterministically via the appropriate next-message function in both cases.

In $\mathsf{RASH}_{\pi_{\mathsf{in}},\mathcal{A}',\mathcal{D}}(\kappa, n)$, the above is guaranteed by the security model itself, whereas in $\mathsf{Real}_{\pi_{\mathsf{out}},\mathcal{A},\mathcal{D}}(\kappa, n)$ it is guaranteed because $\mathcal{F}_{\mathsf{CPE}}$ allows the honest parties to check that all messages between all party-pairs are computed using the correct circuit on the correct view (or else an abort occurs).

Since all variables in view of the distinguisher are identically distributed in both experiments, it must be the case that the experiment outputs are also identically distributed. ∎

Putting it Together

Recall **Theorem 2:** Let $\kappa, n, t \in \mathbb{N}$ such that $t < n$ and n is upper-bounded by some polynomial in κ, let π_{in} be any plain-model protocol for n parties, and let π_{out} be a GMW-compiled version of π_{in}. For every malicious PPT $\mathscr{A}$ that statically corrupts up to t parties, there exists an *augmented* semi-honest PPT $\mathscr{A}'$ that statically corrupts up to t parties, such that for every PPT $\mathscr{D}$ we have

$$\Pr[\mathsf{Real}_{\pi_{\mathsf{out}},\mathscr{A},\mathscr{D}}(\kappa, n) = 1] = \Pr[\mathsf{RASH}_{\pi_{\mathsf{in}},\mathscr{A}',\mathscr{D}}(\kappa, n) = 1]$$

Recall **Corollary 1:** Let $k, n, t \in \mathbb{N}$ such that $t < n$ and n is upper-bounded by some polynomial in κ. If the Decisional Diffie-Hellman Assumption is true, then for any (possibly randomized) n-ary PPT function f, there exists an n-party protocol π_{in} such that for every PPT *augmented* semi-honest $\mathscr{A}'$ that statically corrupts up to t parties, there exists a PPT simulator $\mathcal{S}_{\mathsf{in}}$, such that for every PPT $\mathscr{D}$ there exists a negligible function ε such that

$$\left| \Pr[\mathsf{Ideal}_{\mathscr{F}^f_{\mathsf{SFE-Abort}},\mathcal{S}_{\mathsf{in}},\mathscr{D}}(\kappa, n) = 1] - \Pr[\mathsf{RASH}_{\pi_{\mathsf{in}},\mathscr{A}',\mathscr{D}}(\kappa, n) = 1] \right| \leq \varepsilon(\kappa)$$

Putting it Together

Recall **Corollary 1:** Let $k, n, t \in \mathbb{N}$ such that $t < n$ and n is upper-bounded by some polynomial in κ. If the Decisional Diffie-Hellman Assumption is true, then for any (possibly randomized) n-ary PPT function f, there exists an n-party protocol π_{in} such that for every PPT *augmented* semi-honest $\mathcal{A}'$ that statically corrupts up to t parties, there exists a PPT simulator $\mathcal{S}_{\mathsf{in}}$, such that for every PPT $\mathcal{D}$ there exists a negligible function ε such that

$$\left| \Pr[\mathsf{Ideal}_{\mathcal{F}^f_{\mathsf{SFE-Abort}}, \mathcal{S}_{\mathsf{in}}, \mathcal{D}}(\kappa, n) = 1] - \Pr[\mathsf{RASH}_{\pi_{\mathsf{in}}, \mathcal{A}', \mathcal{D}}(\kappa, n) = 1] \right| \leq \varepsilon(\kappa)$$

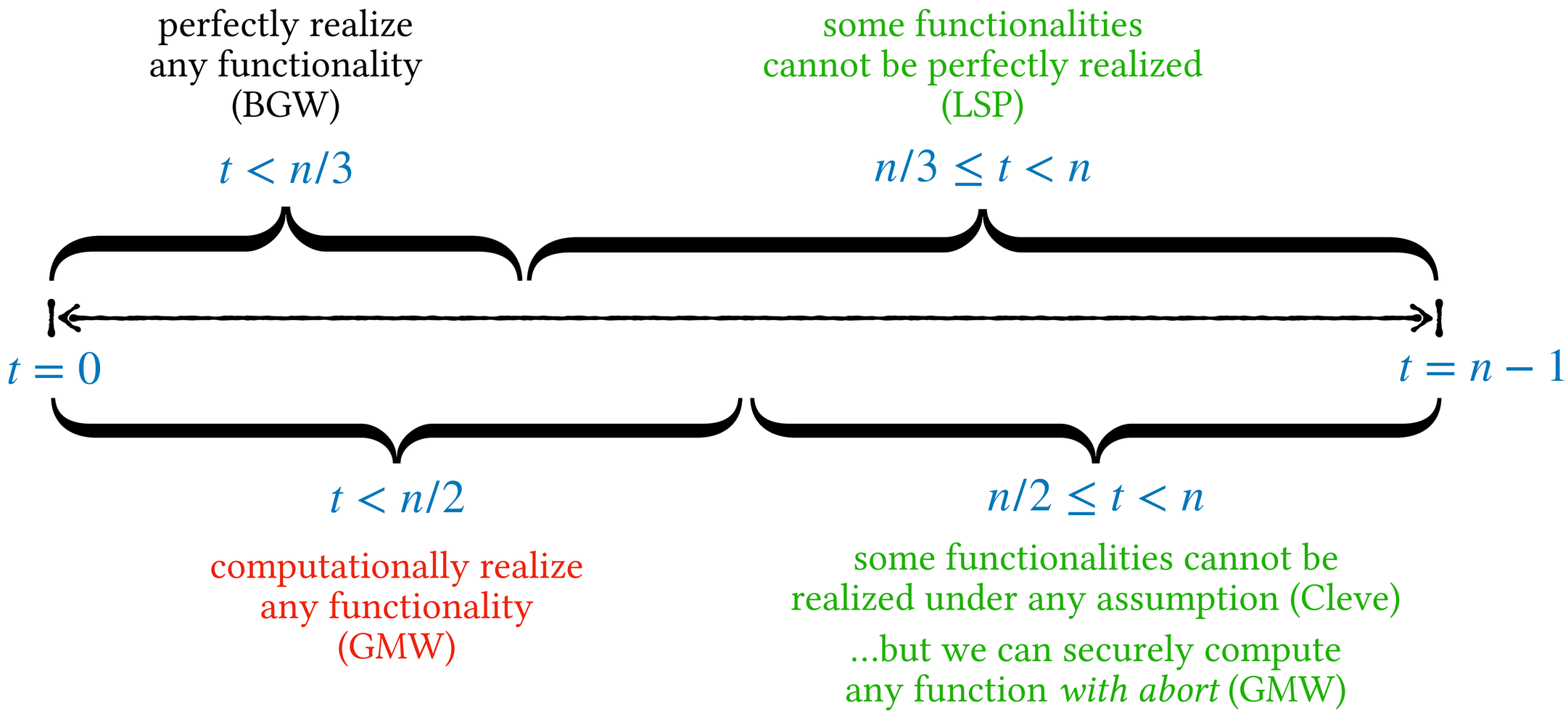
Corollary 2 (Malicious MPC Feasibility Theorem): Let $k, n, t \in \mathbb{N}$ such that $t < n$ and n is upper-bounded by some polynomial in κ. If the Decisional Diffie-Hellman Assumption is true, then for any (possibly randomized) n-ary PPT function f, there exists an n-party protocol π_{out} in the $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{PKI}})$-hybrid model such that for every *malicious* PPT $\mathcal{A}$ that statically corrupts up to t parties, there is a PPT simulator $\mathcal{S}_{\mathsf{out}}$, such that for every PPT $\mathcal{D}$ there exists a negligible function ε such that

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

Proof: By Theorem 1 + Corollary 2 + realizing $\mathcal{F}_{\mathsf{CPE}}$ under DDH in the $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{PKI}})$-hybrid model.

The Landscape of Malicious Security

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



3. Guaranteed Output for Honest Majorities

Robust GMW Compiler, for Honest Majority

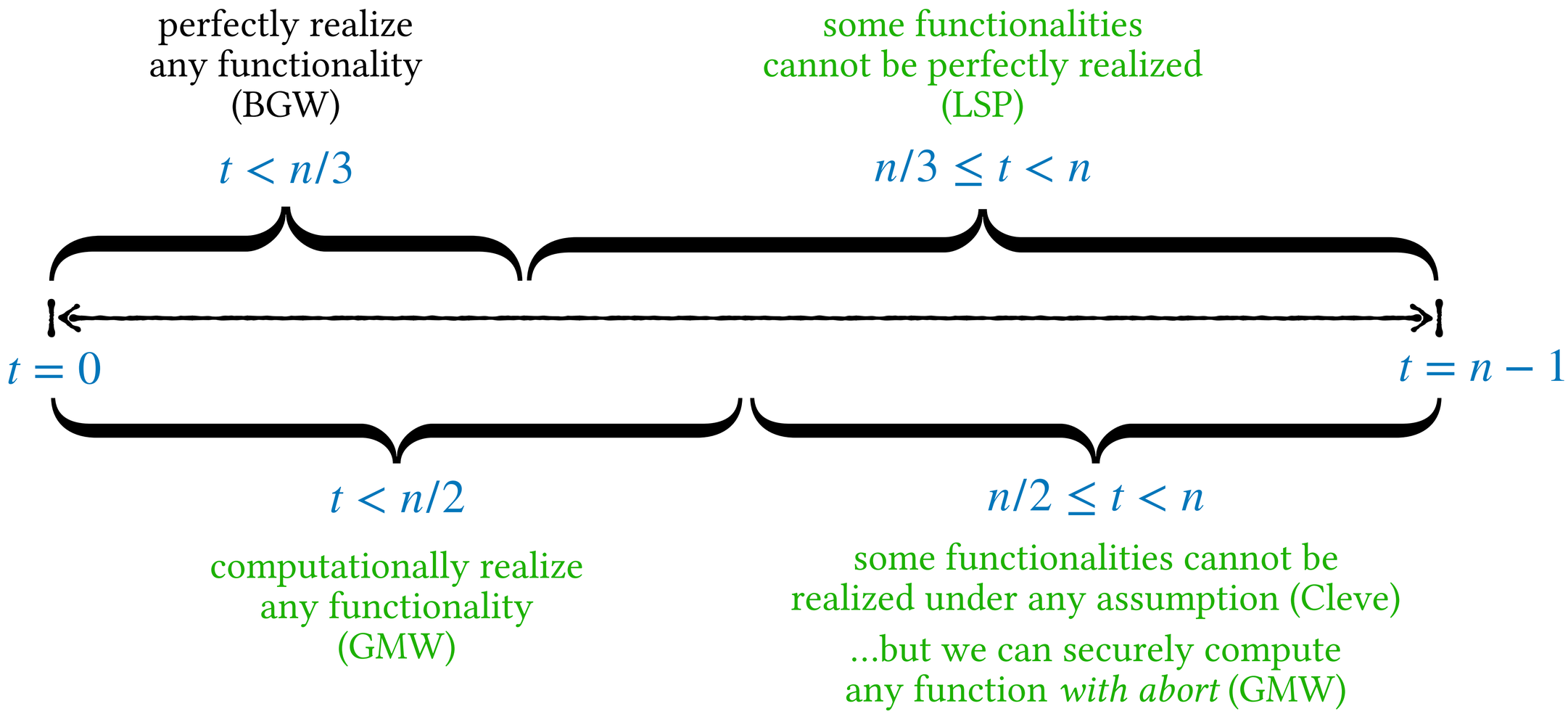
- We will sketch how to alter the previous construction to achieve guaranteed output when $t < n/2$
- First, consider that when using $\mathcal{F}_{\mathsf{CPE}}$, when a party P_i performs a private evaluation toward another party P_j, *only* P_i can cause the process to fail, all other parties always agree upon the outcome, and P_j can later prove statements about the *output*.
- Second, notice that once inputs and randomness are committed in the first two phases of the GMW compiler, the remainder of the protocol is deterministic (in the $\mathcal{F}_{\mathsf{CPE}}$-hybrid model).
- **Main Idea:**
  1. Whenever a party P_i should commit a value v_i using $\mathcal{F}_{\mathsf{CPE}}$ (either its input, or a share of randomness), it also commits an random value r_{share}. It then evaluates a circuit toward each P_j that outputs $\langle v_i \rangle_j$ where $\langle v_i \rangle := \mathsf{Share}_{n,t}(v_i; r_{\mathsf{share}})$.
  2. If P_i is observed to behave maliciously *before* the above process is complete, the other parties agree that P_i has cheated and assume that $v_i := 0$ (i.e. a default value).
  3. If the above process runs to completion, then every party is certain that they collectively possess a *correct* secret-sharing of v_i.

Robust GMW Compiler, for Honest Majority

1. Whenever a party P_i should commit a value v_i using $\mathcal{F}_{\mathsf{CPE}}$ (either its input, or a share of randomness), it also commits an random value r_{share}. It then evaluates a circuit toward each P_j that outputs $\langle v_i \rangle_j$ where $\langle v_i \rangle := \mathsf{Share}_{n,t}(v_i; r_{\mathsf{share}})$.
2. If P_i is observed to behave maliciously *before* the above process is complete, the other parties agree that P_i has cheated and assume that $v_i := 0$ (i.e. a default value).
3. If the above process runs to completion, then every party is certain that they collectively possess a *correct* secret-sharing of v_i.
4. If P_i is observed to behave maliciously (i.e. it fails to commit or evaluate, or evaluates a wrong circuit) *after* the above process is complete, then at least $n - t$ honest parties P_h can reveal $\langle v_i \rangle_h$ (and prove that they have done so correctly), whereupon all parties can reconstruct v_i.
5. Similar techniques ensure P_i's received messages are revealed to everyone when P_i cheats.
6. Now, when a party P_i misbehaves, the other parties do *not* abort, but instead make its internal state *public*. Thereafter, P_i is excluded from the process, and the other parties each run its code on its now-public state in order to generate the messages it should have sent.

The Landscape of Malicious Security

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



4. Wrapping Up

Syllabus (tentative):

A taxonomy of adversaries; a variety of techniques

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

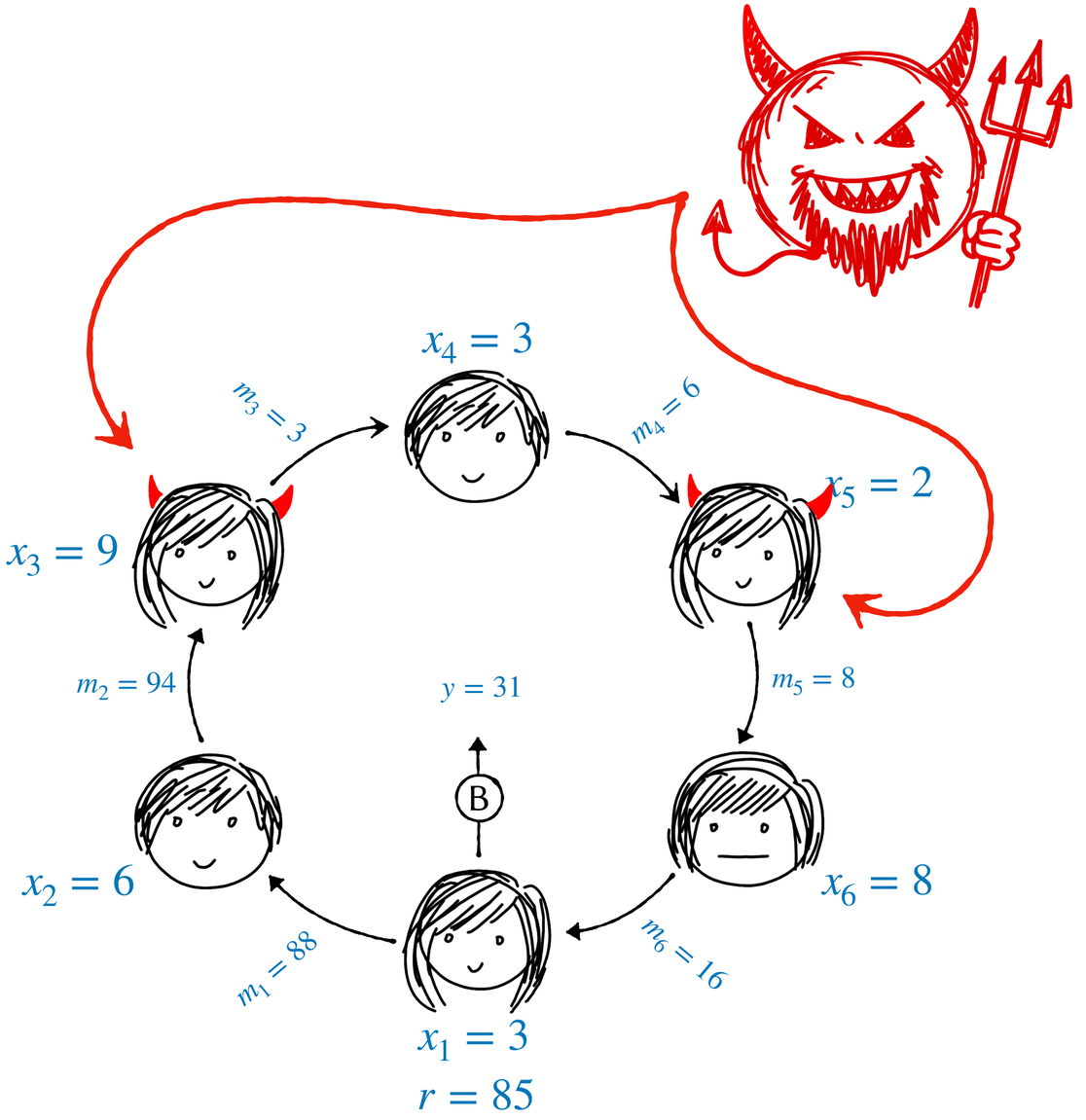
	Part 1: *Information-theoretic* techniques. Adversaries with unbounded power	**Part 2:** *Cryptographic* techniques. Adversaries with bounded power
Semi-honest Adversaries: *follow* the rules of the protocol	Secret Sharing BGW protocol for an honest majority	Oblivious Transfer GMW protocol for a dishonest 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?

You Have Come a Long Way

- Seven weeks ago, I drew this little circle for you, and you proved that we can compute a sum together, as long as at most one person is (semi-honestly) corrupt.
- It probably seemed tricky back then. Hopefully not so much anymore!
- You have waded through an amazing tower of abstractions and techniques in order to learn exactly when and how we have the power to compute together without trusting one another.
- You might not be comfortable with everything yet. That's OK. This is only the beginning of your journey. There's so much more to learn!



What to Learn About Next

- Perfect and statistical security against malicious adversaries.
- Other techniques for transforming semi-honest protocols into malicious ones.
- What happens when channels are asynchronous?
- Security against adaptive corruptions.
- Arbitrary concurrent composition.
- Adversaries with quantum computers.
- Parties with quantum communication channels.
- What if the network isn't fully connected?
- What if want to hide the network topology*?*
- What if we want to hide the function we're securely computing?
- Advanced techniques for improving efficiency (e.g. FHE, HSS, Oblivious RAM).
- Implementations!

Cryptography Needs You!

- Cryptography is still a new field. MPC is newer still.
- Many basic questions are unanswered. How many rounds are necessary to securely compute a function, if $t \leq n/3$ and you want to achieve perfect security? We don't know!
- In 2016, there were essentially no deployments of MPC in the real world. Unless you were a Danish sugar beet farmer, MPC didn't do you any good in practice 10 years ago.
- In 2026, NIST is standardizing MPC protocols for the first time, and the first large-scale deployments are happening. Maybe MPC will be part of the internet infrastructure soon.
- We can hope for more: can it help solve our day-to-day problems? How many people use ChatGPT on inputs that should be private? How many people *trust* ChatGPT?
- Throughout this course I've shown you little photos of the people who developed the theorems. Almost all of them are still alive and working on cryptography today. Go out and find them, and solve new problems with them!

Other Fields Need You Too!

- No one theorem or construction in this class is the important one.
- What I hope you take away most of all is a methodology for reasoning about and limiting the power of unknown adversaries.
- In this class we focused on adversaries that attack interactive protocols, but we can also ask about adversaries in other settings.
- What about adversaries that try to learn about the input of a computation by looking at the *output* of a computation? Our model and techniques say nothing about this!

  Nevertheless, we can achieve meaningful notions of privacy against such adversaries.
  The field that studies this is called *Differential Privacy*.
- What about adversaries that query machine learning models? Can they force those models to behave incorrectly? Can they do this without being detected?

  What about adversaries that influence the training data of machine learning models? How much damage can they cause? Can they hide that damage?

  The theory of machine learning has a long way to go before it catches up with practice!

CS4501 Cryptographic Protocols
Lecture 27: The GMW Compiler

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