MANCHESTER 1824
The University of Manchester

Universally Composable Subversion-Resilient Cryptography

Suvradip Chakraborty
ETH Zurich

Bernardo Magri
The University of Manchester

Jesper Buus Nielsen
Aarhus University

Daniele Venturi
Sapienza University of Rome

EuroCrypt'22

Background on subversion

- Adversary tamper with implementation/spec of crypto
- Started in 80's and 90's – Subliminal channels [Sim84], kleptography [YY97]
- Picked up steam after Snowden's revelations in 2013

Current state of affairs

- Standalone security – no guarantees in larger context
- Every protocol needs to re-prove security from scratch
- Many different models: Reverse Firewall, watchdog, self-guarding, etc…

Our Contributions

- Extension of UC framework to deal with subversions
- Sanitize UC commitments and UC coin toss
- Sanitize GMW compiler to achieve malicious MPC

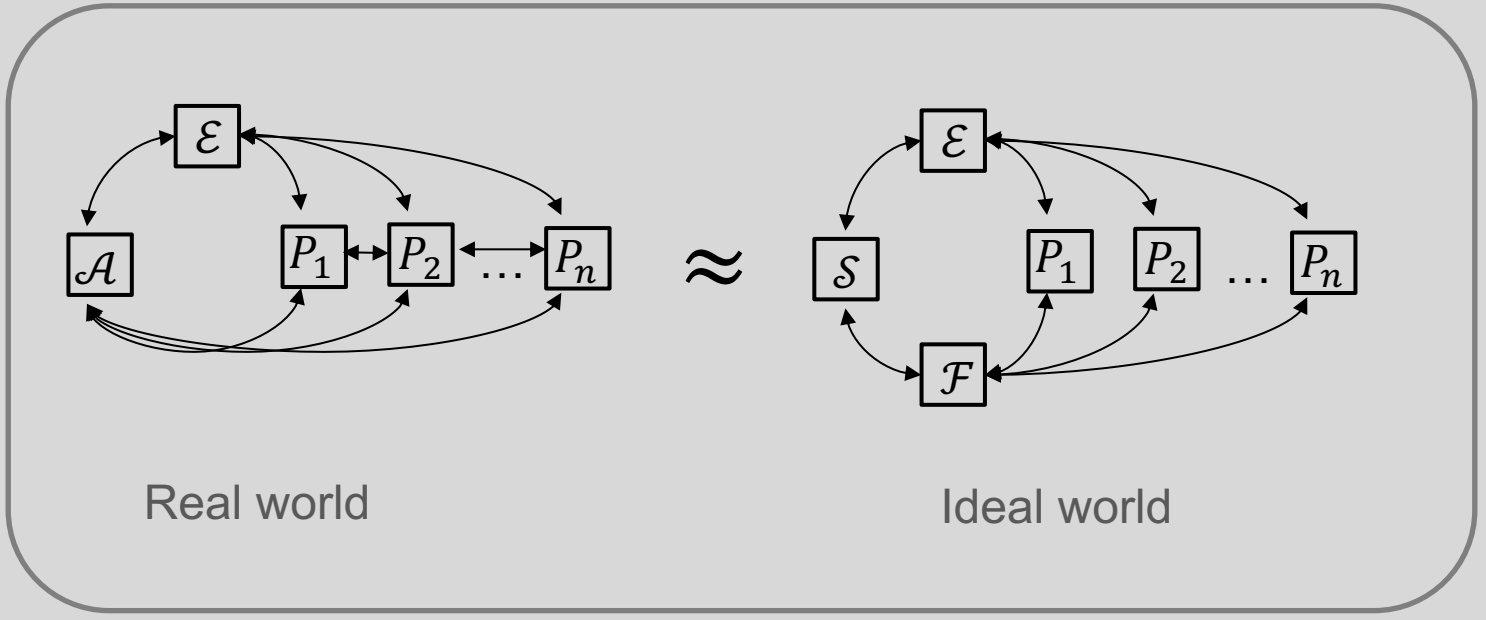
(very) Quick UC Recap

- Define an "ideal functionality" $\mathcal{F}$ for a task
- Design a protocol Π that "implements" $\mathcal{F}$

(very) Quick UC Recap

Π UC-implement $\mathcal{F}$ if:

$$\exists \mathcal{S}\ \forall \mathcal{E}: EXEC_{\Pi,\mathcal{A},\mathcal{E}} \approx EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$$

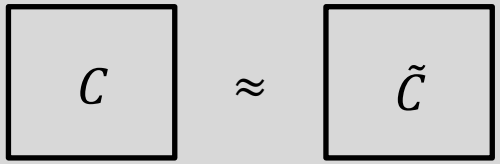


Our Model

- Every UC party P_i is split into 2 parties C_i and F_i :
  - The core C_i is responsible for computing protocol's messages
  - The firewall F_i is responsible for sanitizing C_i's communication

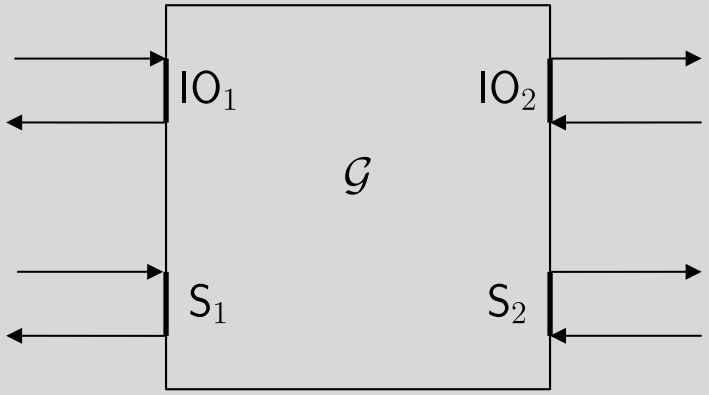
Our Model

- Core and firewall can be independently corrupted
- We allow for "specious" corruptions of the core
  - Specious core is indistinguishable from an honest core, but may leak information via subliminal channel or trigger



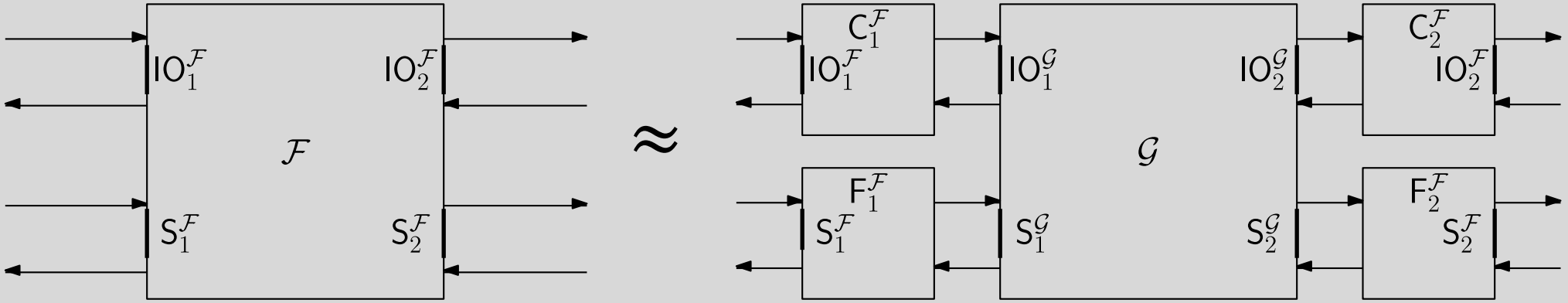
Sanitizable Ideal Functionality

- Dedicated sanitation interface for firewalls (S)



Our Model

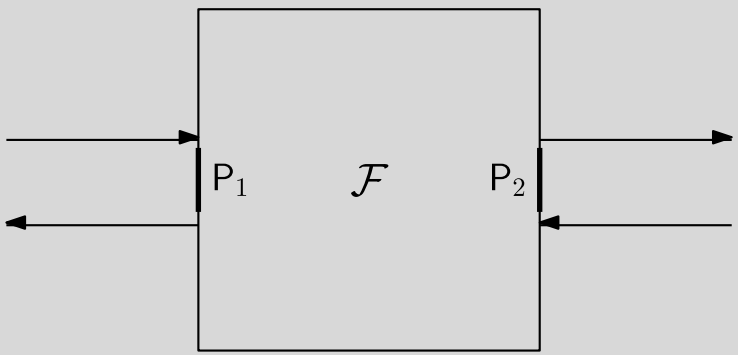
- Implementing a sanitizable ideal functionality



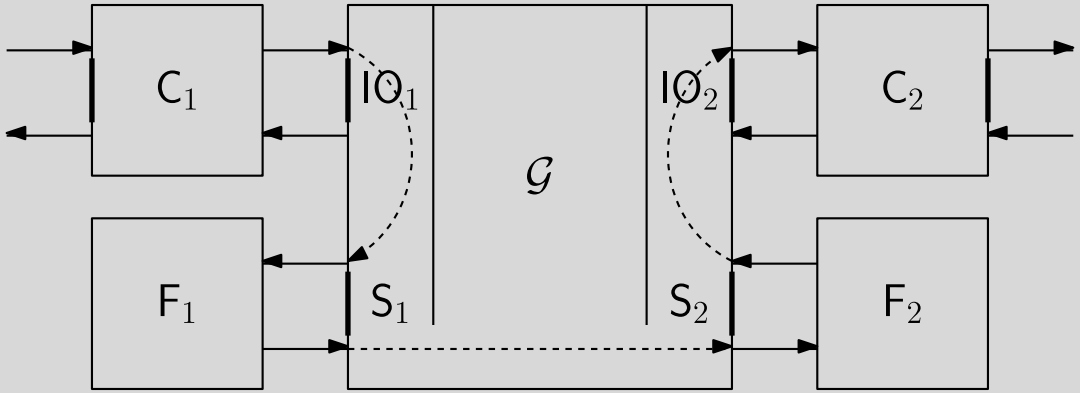
Functionality $\mathcal{F}$

Sanitizable protocol Π implementing $\mathcal{F}$ in $\mathcal{G}$-hybrid model

Sanitizing a regular UC functionality



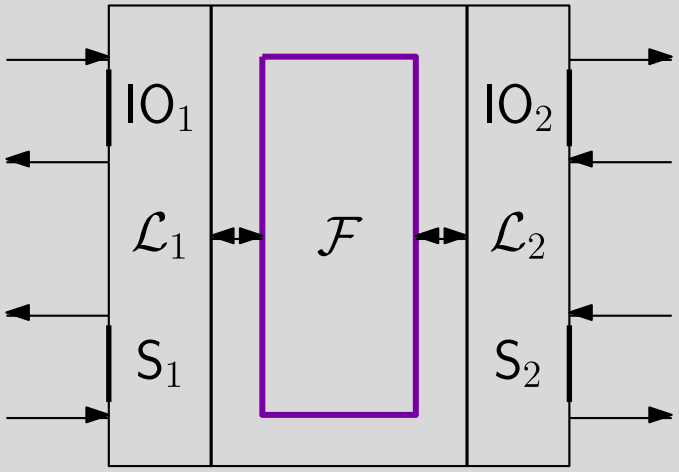
$\approx$



Functionality $\mathcal{F}$

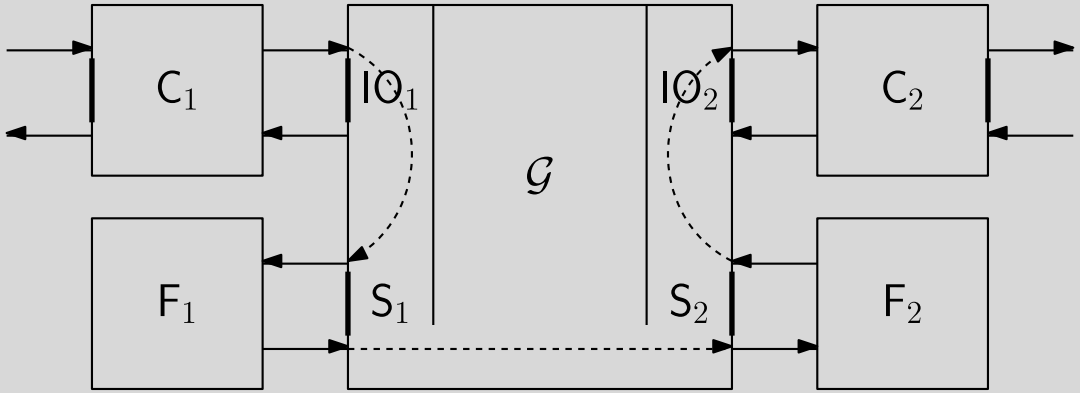
Protocol Π implementing $\mathcal{F}$ in $\mathcal{G}$-hybrid model

Sanitizing a regular UC functionality



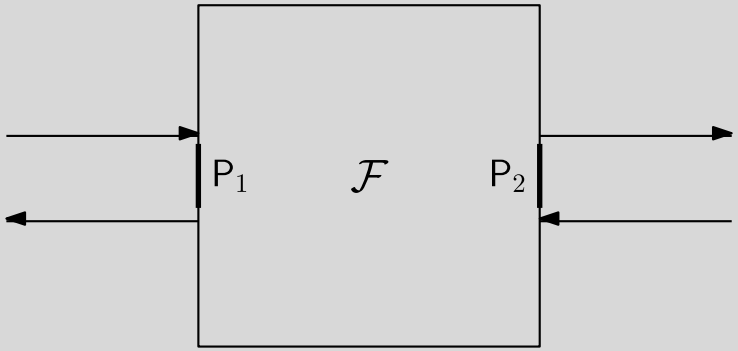
Wrap($\mathcal{F}$)

$\approx$



Protocol Π implementing $\mathcal{F}$ in $\mathcal{G}$-hybrid model

Sanitizing a regular UC functionality



- Transparency
  - Honest core alone is indistinguishable from honest core + firewall

Our Model

- Many more cases to analyze!

Core C	Firewall F
Honest	Honest
Honest	Semi-honest
Honest	Malicious
Specious	Semi-honest
Specious	Honest
Specious	Malicious
Malicious	Honest
Malicious	Semi-honest
Malicious	Malicious

Our Model

- Many more cases to analyze!

Core C	Firewall F	Behaviour in $\mathcal{F}$
Honest	Honest	Honest
Honest	Semi-honest	Honest
Honest	Malicious	Isolated
Specious	Semi-honest	Malicious
Specious	Honest	Specious
Specious	Malicious	Malicious
Malicious	Honest	Malicious
Malicious	Semi-honest	Malicious
Malicious	Malicious	Malicious

Our Model

- Many more cases to analyze!

Core C	Firewall F	Behaviour in $\mathcal{F}$
Honest	Honest	Honest
Honest	Semi-honest	Honest
Honest	Malicious	Isolated
Specious	Semi-honest	Malicious
Specious	Honest	Specious
Specious	Malicious	Malicious
Malicious	Honest	Malicious
Malicious	Semi-honest	Malicious
Malicious	Malicious	Malicious

Our Model

- Many more cases to analyze!

Core C	Firewall F	Behaviour in $\mathcal{F}$
Honest	Semi-honest	Honest
Honest	Malicious	Isolated
Specious	Honest	Specious
Malicious	Malicious	Malicious

Our Model

- Many more cases to analyze!

Core C	Firewall F	Behaviour in $\mathcal{F}$
Honest	Semi-honest	Honest
Honest	Malicious	Isolated
Specious	Honest	Honest
Malicious	Malicious	Malicious

Strong sanitation

Indistinguishability argument!

Our Model

- Many more cases to analyze!

Core C	Firewall F	Behaviour in $\mathcal{F}$
Honest	Semi-honest	Honest
Honest	Malicious	~~Isolated~~ Malicious
Malicious	Malicious	Malicious

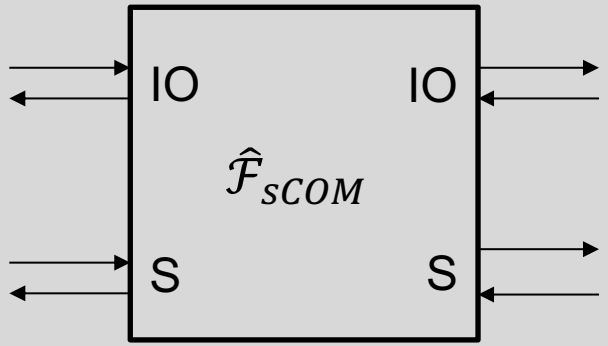
Our Model

- Much better!

Core C	Firewall F	Behaviour in $\mathcal{F}$
Honest	Semi-honest	Honest
Malicious	Malicious	Malicious

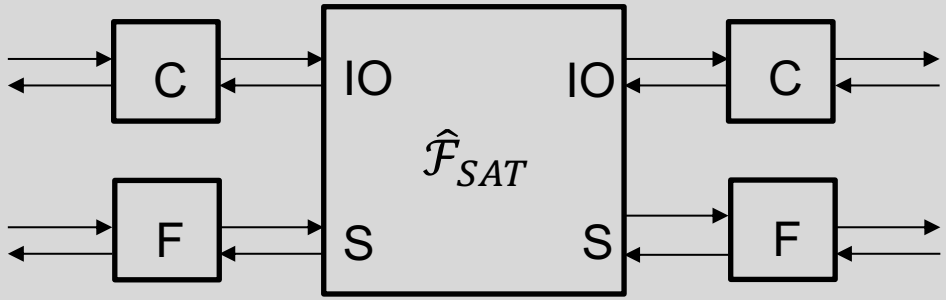
Sanitizable commitment functionality

- Gives the firewall the option to blind the input: $\hat{s}_i = s_i \oplus r_i$
- Upon opening, the receiver gets $\hat{s}_i$



Sanitizable commitment protocol

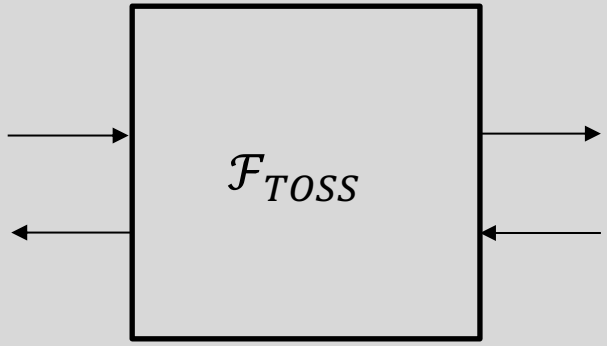
- Inspired on the UC commitment of [Canetti, Sarkar and Wang'20]
- Based on the hardness of DDH
- Allows the firewall to sanitize the input and randomness
- Details on our paper!



Thm: Protocol $\hat{\Pi}$ srUC-realizes the $\hat{\mathcal{F}}_{\mathrm{sCOM}}$ functionality in the $\hat{\mathcal{F}}_{\mathrm{SAT}}$-hybrid model in the presence of up to $\mathrm{n}-1$ malicious static corruptions.

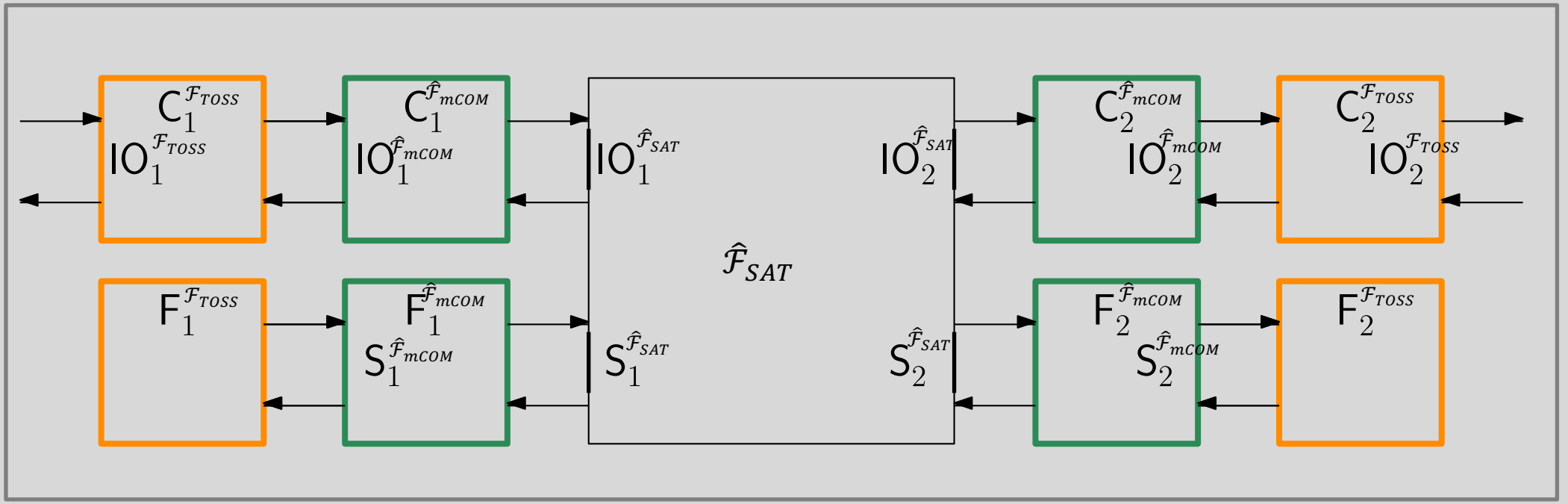
Sanitizing Coin toss (in the $\hat{\mathcal{F}}_{sCOM}$-hybrid model)

- Each core commits to a random s_i with $\hat{\mathcal{F}}_{sCOM}$
- Each firewall samples a random r_i and sends it to $\hat{\mathcal{F}}_{sCOM}$
- Each core output $s = s_i \oplus r_i \oplus (\oplus_{j \neq i} \hat{s_j})$



Thm: Protocol $\hat{\Pi}$ wsrUC-realizes the $\mathcal{F}_{\text{TOSS}}$ functionality in the $\hat{\mathcal{F}}_{\text{sCOM}}$-hybrid model in the presence of up to $\mathbf{n} - 1$ malicious corruptions.

Protocol $\widehat{\Pi}_{TOSS}$



$\widehat{\Pi}_{TOSS}$

GMW Compiler

- Turn semi-honest MPC into malicious MPC [GMW87]
  - Each P_i runs (augmented) coin toss with other parties to get its random tape
  - Each P_i commit to its input and proves in ZK that next message is correct w.r.t its input, current transcript, and random tape

GMW Compiler

- Turn semi-honest MPC into malicious MPC [GMW87]
  - Each P_i runs (augmented) coin toss with other parties to get its random tape
  - Each P_i commit to its input and proves in ZK that next message is correct w.r.t its input, current transcript, and random tape
- Can't prove things about UC commitments!

Sanitizable Commit–and–Prove functionality

- As in [CLOS02] we need a commit-and-prove functionality
- Allows parties to commit to value and prove statements about the committed values
- The firewall has the option to blind the committed values and to verify the proven statements

Sanitizable Commit–and–Prove functionality

- We srUC-realize $\hat{\mathcal{F}}_{C\&P}$ combining the sanitizable commitment construction + re-randomizable NIZKs

Putting it all together: Completeness Theorem

- $\widehat{\Pi}_{GMW}$ is described in $(\hat{\mathcal{F}}_{C\&P}, \mathcal{F}_{\text{TOSS}})$-hybrid model

Putting it all together: Completeness Theorem

- Random tape generation:
  - Core C_i commits to s_i with $\hat{\mathcal{F}}_{C\&P}$
  - Firewall F_i samples random r_i and sends it to $\hat{\mathcal{F}}_{C\&P}$
  - All cores interact with $\mathcal{F}_{TOSS}$ to generate s_i^* for core C_i
  - Core C_i set its random tape to be $\hat{r}_i = s_i^* \oplus (s_i \oplus r_i)$

Putting it all together: Completeness Theorem

- Input commitment:
  - The core sends input x_i to $\hat{\mathcal{F}}_{C\&P}$ that stores it in a list $\bar{x}_i$
  - The firewall choose to **not blind** x_i (the input does not change)

Putting it all together: Completeness Theorem

- Protocol execution
  - The core C_i runs the code of Π on its list $\bar{x}_i$, transcript τ, and random tape $\hat{r}_i$
  - For each message μ sent by P_i in Π, core C_i proves (by asking $\hat{\mathcal{F}}_{C\&P}$) that μ is the correct next message w.r.t list $\bar{x}_i$, transcript τ, and random tape $\hat{r}_i$
  - The firewall now checks that the statement is good, i.e., that s_i^* is the output of $\mathcal{F}_{TOSS}$ and τ is the correct transcript up to now
  - Upon receiving OK from $\hat{\mathcal{F}}_{C\&P}$, core and firewall just append μ to transcript and start over

Conclusions and future work

- New model for handling subversions under composition

- Design firewalls for other functionalities

(e.g. OT, ZK, etc)

- MPC with adaptive corruptions?