Cryptography on the Blockchain

Vassilis Zikas
RPI

IACR Summer School on Blockchain Techs

Bitcoin
What is bitcoin and how does it work?
# Bitcoin

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is bitcoin and how does it work?</td>
<td>✔️</td>
</tr>
<tr>
<td>Is it secure?</td>
<td>✔️ ?</td>
</tr>
</tbody>
</table>

*(in restricted models)*
<table>
<thead>
<tr>
<th><strong>Bitcoin</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>What is bitcoin and how does it work?</strong></td>
<td>✅</td>
</tr>
<tr>
<td><strong>Is it secure?</strong></td>
<td>(in restricted models)</td>
</tr>
<tr>
<td><strong>What do we get from it?</strong></td>
<td>✅ ❓</td>
</tr>
<tr>
<td>What is bitcoin and how does it work?</td>
<td>✓</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>Is it secure?</td>
<td>✓ ?</td>
</tr>
<tr>
<td>What do we get from it?</td>
<td>(in restricted models)</td>
</tr>
</tbody>
</table>
What Crypto can get from Bitcoin?
What Crypto can get from Bitcoin?

In this talk

“Bitcoin = Ledger-based cryptocurrency”
What Crypto can get from Bitcoin?

In this talk
“Bitcoin = Ledger-based cryptocurrency”

A public transaction ledger

Some economic stuff …
What Crypto can get from Bitcoin?

In this talk
“Bitcoin = Ledger-based cryptocurrency”

A public transaction ledger

A bulletin board with a filter on what gets written there

Some economic stuff …
What Crypto can get from Bitcoin?

In this talk
“Bitcoin = Ledger-based cryptocurrency”

A public transaction ledger
A bulletin board with a filter on what gets written there

Some economic stuff …
People (good or bad) want money
“What is exactly the problem that bitcoin solves?”
AK, 2016
The Public Transaction Ledger

“What is exactly the problem that bitcoin solves?”
AK, 2016

The core security goal of Bitcoin is to ensure that all parties establish a common and irreversible view of the sequence of transactions.
The Public Transaction Ledger

“What is exactly the problem that bitcoin solves?”
AK, 2016

“Backbone” [GarayKiayiasLeonardos15]
The core security goal of Bitcoin is to ensure that all parties establish a common and irreversible view of the sequence of transactions
The Public Transaction Ledger

“What is exactly the problem that bitcoin solves?”
AK, 2016

“Backbone” [GarayKiayiasLeonardos15]
The core security goal of Bitcoin is to ensure that all parties establish a common and irreversible view of the sequence of transactions

This goal can be captured as an ideal Transaction-Ledger Functionality
The Public Transaction Ledger

“What is exactly the problem that bitcoin solves?”

AK, 2016

“Backbone” [GarayKiayiasLeonardos15]
The core security goal of Bitcoin is to ensure that all parties establish a common and irreversible view of the sequence of transactions.

This goal can be captured as an ideal Transaction-Ledger Functionality

“If we had a trusted third party instead of the Bitcoin network, how would we expect it to behave?”
Crypto On Blockchain

Outline

• The functionality offered by blockchains

• Leveraging Security Loss with Coins
  ... in Secure Function Evaluation (SFE)

• A formal cryptographic (UC) model for security proofs
Crypto On Blockchain

Outline

• The functionality offered by blockchains

• Leveraging Security Loss with Coins
  ... in Secure Function Evaluation (SFE)

• A formal cryptographic (UC) model for security proofs
The Public Transaction Ledger

State
St

ledger
The Public Transaction Ledger

GetState

"State"

GetState

State

St

Gledger
The Public Transaction Ledger

GetState

"State"

(State, St)

(State, St)

GetState

(Gledger, x)
The Public Transaction Ledger

- GetState
- "State"
- (Submit, x)

The diagram represents a public transaction ledger with a state transition diagram. The process starts with a submission (Submit, x) which updates the state (Stllx). The state is retrieved through the GetState function. The diagram illustrates the flow of transaction submissions and state changes within the ledger.
The Public Transaction Ledger

- In reality: Not a Bulletin Board
- Inputs (transactions) are filtered
The Public Transaction Ledger

- In reality: Not a Bulletin Board
- Inputs (transactions) are filtered
The Public Transaction Ledger

- In reality: Not a Bulletin Board
- Inputs (transactions) are filtered
The Public Transaction Ledger

- In reality: Not a Bulletin Board
- Inputs (transactions) are filtered
The Public Transaction Ledger

- In reality: Not a Bulletin Board
- Inputs (transactions) are filtered
The Public Transaction Ledger

In reality: Not a Bulletin Board
Inputs (transactions) are filtered
The Public Transaction Ledger

- In reality: Not a Bulletin Board
- Inputs (transactions) are filtered
The Public Transaction Ledger

- In reality: Not a Bulletin Board
- Inputs (transactions) are filtered
- The order in which transactions in “State” are inserted might be adversarial … but not too adversarial
The Public Transaction Ledger

GetState

“State”

State $\text{Stllx}$

“State”

Validate(.)

Yes

No

$(\text{Submit, } x)$

Can reorder the recently inserted transactions
Can reorder the recently inserted transactions
The Public Transaction Ledger & Time

Can reorder the recently inserted transactions
The Public Transaction Ledger & Time

GetState

Can reorder the recently inserted transactions
The Public Transaction Ledger & Time

GetState

"State"

State  Buffer  Validate(.)

"State"

(Submit, x)

Can reorder the recently inserted transactions
The Public Transaction Ledger & Time

Can reorder *the recently inserted* transactions
The Public Transaction Ledger & Time

Can reorder the recently inserted transactions
The Public Transaction Ledger & Time

GetState

“State”

State

Buffer $x_1, x_2, \ldots$

Validate(.)

Yes

No

$G_{ledger}$

(Submit, x)

Can reorder **the recently inserted** transactions
The Public Transaction Ledger & Time

\[ \text{State} \rightarrow \text{Buffer} \]

\[ x_1, x_2, \ldots = \pi(x_1, \ldots) \]

\[ \text{Validate}(\cdot) \]

\[ \text{Submit}, x \]

\[ \text{GetState} \rightarrow \text{State} \]

\[ \text{Can reorder the recently inserted transactions} \]
The Public Transaction Ledger & Time

- **State**: $x_1, x_2, \ldots = \pi(x_1, \ldots)$
- **Buffer**: $\pi(x_1, \ldots)$
- **Validate(.)**: Yes, No
- **GetState**
- **time?**
- **(Submit, x)**

Can reorder **the recently inserted** transactions.
The Public Transaction Ledger & Time

Can reorder **the recently inserted** transactions
Can reorder the recently inserted transactions
The Public Transaction Ledger & Time

GetState

Blockify(.)

Buffer

\( x_1, x_2, \ldots = \pi(x_1, \ldots) \)

Validate(.)

\( \mathcal{G}_{\text{ledger}} \)

(Submit, x)

time?

Can reorder the recently inserted transactions
The Public Transaction Ledger & Time

Can reorder **the recently inserted** transactions
The Public Transaction Ledger & Time

Can reorder the recently inserted transactions
<table>
<thead>
<tr>
<th><strong>What Crypto can we get from Bitcoin?</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A public transaction ledger</strong></td>
</tr>
<tr>
<td>A bulletin board with a filter on what gets written there</td>
</tr>
</tbody>
</table>

**The Model**

\[(G_{\text{ledger}}, G_{\text{clock}})\text{-hybrid}\]

\[(G)\text{UC protocols}\]
What Crypto can we get from Bitcoin?

<table>
<thead>
<tr>
<th>A public transaction ledger</th>
<th>Some economic stuff …</th>
</tr>
</thead>
<tbody>
<tr>
<td>A bulletin board with a filter on what gets written there</td>
<td>People (good or bad) want money</td>
</tr>
</tbody>
</table>

The Model

\((\mathcal{G}_{\text{ledger}}, \mathcal{G}_{\text{clock}})\)-hybrid

(G)UC protocols

- Compatibility with standard crypto-protocols (+ composition theorem)
What Crypto can we get from Bitcoin?

A public transaction ledger

A bulletin board with a filter on what gets written there

The Model

\((G_{\text{ledger}}, G_{\text{clock}})\)-hybrid

(G)UC protocols

Some economic stuff …

People (good or bad) want money

- Compatibility with standard crypto-protocols (+ composition theorem)
- Cryptographically as useful as having access to (synchronous) stateful broadcast
What Crypto can we get from Bitcoin?

A public transaction ledger

A bulletin board with a filter on what gets written there

Some economic stuff …

People (good or bad) want money

The Model

$(G_{ledger}, G_{clock})$-hybrid

(G)UC protocols

“This cryptography has been around for a long time” JB 2016

- Compatibility with standard crypto-protocols (+ composition theorem)
- Cryptographically as useful as having access to (synchronous) stateful broadcast
What Crypto can we get from Bitcoin?

A public transaction ledger

- A bulletin board with a filter on what gets written there

The Model

\((G_{\text{ledger}}, G_{\text{clock}})\)-hybrid

(G)UC protocols

Some economic stuff …

- People (good or bad) want money

“\text{This cryptography has been around for a long time}” JB 2016

- Compatibility with standard crypto-protocols (+composition theorem)
- Cryptographically as useful as having access to (synchronous) stateful broadcast
Crypto On Blockchain

Outline

• The functionality offered by blockchains

• Leveraging Security Loss with Coins
  … in Secure Function Evaluation (SFE)

• A formal cryptographic (UC) model for security proofs
Crypto On Blockchain

Outline

• The functionality offered by blockchains

• Leveraging Security Loss with Coins
  ... in Secure Function Evaluation (SFE)

• A formal cryptographic (UC) model for security proofs
Secure Function Evaluation (SFE)

**Goal:** Parties $P_1,\ldots,P_n$ with inputs $x_1,\ldots,x_n$ wish to compute a function $f(x_1,\ldots,x_n)$ securely.
Secure Function Evaluation (SFE)

Ideal World

\[ F^f \]

\[
x_1 \xrightarrow{f(x)} P_1 \quad x_2 \xleftarrow{f(x)} P_2 \quad \cdots \quad x_n \xrightarrow{f(x)=y} P_n
\]
Secure Function Evaluation (SFE)

Ideal World

Real World
Secure Function Evaluation (SFE)

Ideal World

\[ F^f \]

\[ \pi_1(x_1) \quad \pi_2(x_2) \quad \ldots \quad \pi_n(x_n) \]

Real World

\[ \approx \]

\[ \pi_1(x_1) \quad \pi_2(x_2) \quad \ldots \quad \pi_n(x_n) \]
Secure Function Evaluation (SFE)

Ideal World

\[ f(x_1) \leftrightarrow f(\tilde{x}) \]
\[ P_1 \leftrightarrow \cdots \leftrightarrow P_n \]
\[ f(\tilde{x}) = y \]

Real World

\[ \pi_1(x_1) \leftrightarrow \pi_2(x_2) \leftrightarrow \cdots \leftrightarrow \pi_n(x_n) \]
Secure Function Evaluation (SFE)

Ideal World

Real World

\[ f(x) = y \]
Secure Function Evaluation (SFE)

Protocol \( \pi \) is secure if for every adversary:

- (privacy) Whatever the adversary learns he could compute by himself
- (correctness) Honest (uncorrupted) parties learn their correct outputs
**Fair SFE**

In fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output.
Fair SFE

**In fair SFE:** If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output

\[ F^f \]

\[ \perp \quad \perp \quad \perp \]

\[ P_1 \quad P_2 \]
**Fair SFE**

**In fair SFE:** If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output.
**Fair SFE**

**In fair SFE:** If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output.

Fair SFE is impossible against corrupted majorities [Cleve86]
Fair SFE

In fair SFE: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output.

Fair SFE is impossible against corrupted majorities [Cleve86]

Security against corrupted majorities = Security with abort
Fair SFE

**In fair SFE:** If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output.

Fair SFE is impossible against corrupted majorities [Cleve86]

Security against corrupted majorities = Security with abort

Discounted security
SFE with Fair(ness) Compensation

Idea [AndrychowiczDziembowskiMalinowskiMazurek14]:
We can leverage unfairness with $$$

SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output or get compensated.
SFE with Fair(ness) Compensation

Idea [AndrychowiczDziembowskiMalinowskiMazurek14]: We can leverage unfairness with $$$

SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output or get compensated.

\[ F_f \]

\[ \perp \] \[ \perp \] \[ \perp \] 
\[ P_1 \] \[ P_2 \] 

[Unfair]
SFE with Fair(ness) Compensation

**Idea** [AndrychowiczDziembowskiMalinowskiMazurek14]: We can leverage unfairness with $$$

**SFE with fair compensation**: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output or get compensated.

\[ F^f \]

\[
\begin{align*}
\uparrow \downarrow & \quad \downarrow \downarrow \quad \uparrow \downarrow \\
\text{P}_1 & \quad \text{P}_2 \\
+ & \quad + \\
\text{⚠️} & \quad \text{⚠️}
\end{align*}
\]

\[
\begin{align*}
& \quad \downarrow \\
\text{⚠️} & \quad \text{⚠️} \quad \text{⚠️}
\end{align*}
\]

(Unfair)
SFE with Fair(ness) Compensation

Idea [AndrychowiczDziembowskiMalinowskiMazurek14]: We can leverage unfairness with $$$

SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output or get compensated.

\[ F^f \]

\[ P_1 \downarrow \quad \downarrow \downarrow \downarrow \quad P_2 \]

\[ + \quad + \quad - \]

\[ + \quad + \quad - \]

\[ \checkmark \quad (Unfair) \quad \checkmark \quad ("fair") \]
SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Tools 1/2: Authenticated Additive Secret Sharing

\[ x = x_1 \oplus ... \oplus x_n, (sk, vk) \leftarrow \text{KeyGen} \]

\[ [x]_1 = x_1, \text{Sig}_{sk}(x_1), vk \]

\[ [x]_n = x_n, \text{Sig}_{sk}(x_n), vk \]
SFE with Fair(ness) Comp.: Construction

[ BentovKumaresan14,15 ]

Tools 1/2: Authenticated Additive Secret Sharing

\[ x = x_1 \oplus \ldots \oplus x_n, (sk, vk) \leftarrow \text{KeyGen} \]

- No \( n-1 \) parties have info on \( x \)
- Together all \( n \) parties can recover \( x \)
- No party can lie about its share
  - Only \( x \) might be reconstructed!
S transfers $q$ coins to $R$ such that
Tools 2/2: Claim and Refund Transactions

S transfers q coins to R such that

- Time restriction $\tau$
S transfers q coins to R such that

- Time restriction $\tau$

\[
\text{time} \quad \frac{1}{2}
\]
Tools 2/2: Claim and Refund Transactions

S transfers q coins to R such that

- Time restriction $\tau$

Diagram:

- Time
- $\tau$
- R can claim coins
- S can claim coins

References:
[BentovKumaresan14,15]
S transfers q coins to R such that

- Time restriction $\tau$

\[
\begin{array}{c|c}
\text{time} & \tau \\
\hline
\text{R can claim coins} & \text{S can claim coins}
\end{array}
\]

- A predicate (relation) $R(\text{state,buffer,tx})$:
  - In order to spend the coins the receiver needs to submit a tx satisfying $R$ (at the point of validation).
SFE with Fair(ness) Comp.: Construction

[BentovKumaresan14,15]

Tools 2/2 : Claim and Refund Transactions

S transfers q coins to R such that

- Time restriction $\tau$

<table>
<thead>
<tr>
<th>time</th>
<th>$\tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>R can claim coins</td>
<td>S can claim coins</td>
</tr>
</tbody>
</table>

- A predicate (relation) $R(state, buffer, tx)$:
  - In order to spend the coins the receiver needs to submit a tx satisfying $R$ (at the point of validation).

- Supported by Bitcoin scripting language
- Captured by $Validate(.)$
Protocol Idea for computing $y = f(x_1, \ldots, x_n)$

1. Run SFE with unfair abort to compute $n$-out-of-$n$ authenticated sharing $[y]$ of $y = f(x_1, \ldots, x_n)$
   - E.g., Every $P_i$ receives share $[y]_i$ such that $y = [y]_1 + \ldots + [y]_n$ and public signature on $[y]_i$
SFE with Fair(ness) Comp.: Construction

[Untexted reference: BentovKumaresan14,15]

Protocol Idea for computing $y = f(x_1, \ldots, x_n)$

1. Run SFE with unfair abort to compute \textit{n-out-of-n} authenticated sharing $[y]$ of $y = f(x_1, \ldots, x_n)$

   - E.g., Every $P_i$ receives share $[y]_i$ such that $y = [y]_1 + \ldots + [y]_n$ and public signature on $[y]_i$
Protocol Idea for computing \( y = f(x_1, \ldots, x_n) \)

1. Run SFE with unfair abort to compute \( n \)-out-of-\( n \) authenticated sharing \([y]\) of \( y = f(x_1, \ldots, x_n)\)
   
   - E.g., Every \( P_i \) receives share \([y]_i\) such that \( y = [y]_1 + \ldots + [y]_n\) and public signature on \([y]_i\)

\[ f \]

Abort at this point is fair
Protocol Idea for computing $y=f(x_1,\ldots,x_n)$

2. Use the following reconstruction idea:

2.1. Every $P_i$ transfers 1 bitcoin to every $P_j$ with the restriction:

- $P_j$ can claim (spend) this coin in round $\rho_{ij}$ if it submits to the ledger his valid share (and signature) by round $\rho_{ij}$
- if $P_j$ has not claimed this coin by the end of round $\rho_{ij}$, then the coin is “refunded” to $P_i$ (i.e., after round $\rho_{ij}$, $P_i$ can spend this coin himself).
Protocol Idea for computing $y = f(x_1, \ldots, x_n)$

2. Use the following reconstruction idea:

2.1. Every $P_i$ transfers 1 bitcoin to every $P_j$ with the restriction:

- $P_j$ can claim (spend) this coin in round $\rho_{ij}$ if it submits to the ledger his valid share (and signature) by round $\rho_{ij}$
- if $P_j$ has not claimed this coin by the end of round $\rho_{ij}$, then the coin is “refunded” to $P_i$ (i.e., after round $\rho_{ij}$, $P_i$ can spend this coin himself).

2.2. Proceed in rounds in which the parties claim the coins from other parties by announcing their shares (and signatures)
SFE with Fair(ness) Comp.: Construction

[ BentovKumaresan14,15 ]

Protocol Idea for computing \( y = f(x_1, \ldots, x_n) \)

Security (SFE with fair compensation): Follow the money …

- If the adversary announces all his shares then every party:
  - Sends \( n \) coins in phase two (one to each party)
  - Claims back \( n \) coins in phase three (one from each party)

- If a corrupted party \( P_j \) does not announce his share then every party
  - Sends \( n \) coins in phase two (one to each party)
  - Claims back
    - \( n \) coins in phase three for announcing his shares
    - the coin that it had sent to \( P_j \)
Rethinking SFE w Fair(ness) Compensation

[ BentovKumaresan14,15 ]

Time
Rethinking SFE w Fair(ness) Compensation

[ BentovKumaresan14,15 ]

Time

------------------ Protocol Starts
Rethinking SFE w Fair(ness) Compensation

Protocol Starts
Sharing is Output, Committed transactions

[BentovKumaresan14,15]
Rethinking SFE w Fair(ness) Compensation

[BentovKumaresan14,15]

Time

- Seconds: Protocol Starts, Sharing is Output, Committed transactions
- 1 hour: Start reclaiming transactions
Rethinking SFE w Fair(ness) Compensation

[BentovKumaresan14,15]

Time

- Seconds: Protocol Starts, Sharing is Output, Committed transactions
- 1 hour: Start reclaiming transactions
- several hours: Output or compensation is settled
Rethinking SFE w Fair(ness) Compensation

Time

- Seconds
  - Protocol Starts
  - Sharing is Output, Committed transactions

- 1 hour
  - Start reclaiming transactions

- Several hours
  - Output or compensation is settled

“several” =
  - [BentovKumaresan14] linear in players (n)
  - [BentovKumaresan15] constant
Rethinking SFE w Fair(ness) Compensation

What if the adversary aborts before making the committed transactions?

Time

---

Seconds Protocol Starts

---

Sharing is Output, Committed transactions

---

1 hour Start reclaiming transactions

---

several hours output or compensation is settled

"several" =

- [BentovKumaresan14] linear in players (n)
- [BentovKumaresan15] constant
Rethinking SFE w Fair(ness) Compensation

What if the adversary aborts before making the committed transactions?

Protocol Starts

Seconds

Sharing is Output, Committed transactions

1 hour

Start reclaiming transactions

This can be confirmed here …

several hours

output or compensation is settled

“several” =
- [BentovKumaresan14] linear in players (n)
- [BentovKumaresan15] constant
Rethinking SFE w Fair(ness) Compensation

What if the adversary aborts before making the committed transactions?

Protocol Starts

Seconds

Sharing is Output, Committed transactions

1 hour

Start reclaiming transactions

Several hours

output or compensation is settled

“Several” =
- [BentovKumaresan14] linear in players (n)
- [BentovKumaresan15] constant
Rethinking SFE w Fair(ness) Compensation

Time

- Seconds
- 1 hour
- Several hours

Protocol Starts

Sharing is Output, Committed transactions

Start reclaiming transactions

What if the adversary aborts before making the committed transactions?

This can be confirmed here …

… and reclaimed here …

Output or compensation is settled

“several” =

- [BentovKumaresan14] linear in players (n)
- [BentovKumaresan15] constant
Rethinking SFE w Fair(ness) Compensation

What if the adversary aborts before making the committed transactions?

Protocol Starts

Seconds
Sharing is Output, Committed transactions

1 hour
Start reclaiming transactions

This can be confirmed here …

… and reclaimed here …

output or compensation is settled

“several” =
• [BentovKumaresan14] linear in players (n)
• [BentovKumaresan15] constant

Time

O(n) times

= O(n) hours till output

several hours
Rethinking SFE w Fair(ness) Compensation

**SFE with fair compensation:** If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output or get compensated.

![Diagram of fair and unfair compensation scenarios.](image)
**Rethinking SFE w Fair(ness) Compensation**

**SFE with fair compensation:** If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output **or** get compensated.

![Diagram of SFE with fair compensation](image)
SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output or get compensated.
SFE with fair compensation: If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output or get compensated.
SFE with Robust(ness) Compensation
SFE with Robust(ness) Compensation

**Fair SFE:** If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output.
**SFE with Robust(ness) Compensation**

**Fair SFE:** If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output.

**SFE with fair compensation:** If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output or get compensated.
If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output or get compensated (fast …)
SFE with Robust(ness) Compensation

**Fair SFE:** If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output.

**SFE with fair compensation:** If the adversary learns any information beyond (what is derived by) its inputs then every honest party should learn the output or get compensated (fast ...)

**How can we get robustness?**
S transfers $q$ coins to $R$ such that
SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- Time restriction \((\tau_-, \tau_+)\)
SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- Time restriction \((\tau_-, \tau_+)\)

\[\text{time} \quad \underline{\text{---------------------------}}\]
S transfers $q$ coins to $R$ such that

- **Time restriction** $(\tau_-, \tau_+)$

![Diagram showing time restriction and transactions]

- Coins are blocked
- $R$ can claim coins
- $S$ can claim coins
SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- **Time restriction** \((\tau_-, \tau_+)\)

- **Link**: A reference `ref` such that only a transaction with the same reference can spend the q coins
SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- **Time restriction**: $(\tau_-, \tau_+)$

  - time
  
  - coins are blocked
  
  - R can claim coins
  
  - S can claim coins

- **Link**: A reference $\text{ref}$ such that only a transaction with the same reference can spend the q coins

- **A predicate (relation)** $\mathcal{R}(\text{state}, \text{buffer}, \text{tx})$:
  - In order to spend the coins the receiver needs to submit a tx satisfying $\mathcal{R}$ (at the point of validation).
SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- Time restriction $(\tau_-, \tau_+)$

- Link: A reference $\text{ref}$ such that only a transaction with the same reference can spend the q coins

- A predicate (relation) $\mathcal{R}(\text{state}, \text{buffer}, \text{tx})$:
  - In order to spend the coins the receiver needs to submit a tx satisfying $\mathcal{R}$ (at the point of validation).

$$B_v, address_i, address_j, \Sigma, \text{aux}, \sigma_i, \tau$$
SFE with Robust Compen. : Construction

Tools 1/3 : Special Transaction

S transfers q coins to R such that

- **Time restriction** \((\tau_-, \tau_+)\)

- **Link**: A reference \(\text{ref}\) such that only a transaction with the same reference can spend the q coins.

- A predicate (relation) \(R(\text{state}, \text{buffer}, \text{tx})\):
  - In order to spend the coins the receiver needs to submit a tx satisfying \(R\) (at the point of validation).

\[
B_{v, \text{address}_i, \text{address}_j, \Sigma, \text{aux}, \sigma_i, \tau}
\]
SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE
An SFE protocol which is secure when parties follow their instructions
SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE
An SFE protocol which is secure when parties follow their instructions

**Example:** A Summation protocol

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>( x_1 )</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>( x_2 )</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>( P_n )</td>
<td>( x_n )</td>
</tr>
</tbody>
</table>

\[ y_1 + y_2 + \ldots + y_n = \sum_{i=1}^{n} y_i \]
SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE

An SFE protocol which is secure when parties follow their instructions

**Example:** A Summation protocol

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$x_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_2$</td>
<td>$x_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_n$</td>
<td>$x_n$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$x_1 = \bigoplus_{j=1}^{n} x_{1j}$
SFE with Robust Compen.: Construction

Tools 2/3: Semi-honest SFE

An SFE protocol which is secure when parties follow their instructions

**Example:** A Summation protocol

<table>
<thead>
<tr>
<th></th>
<th>( P_1 )</th>
<th>( P_2 )</th>
<th>( P_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>( x_1 )</td>
<td>( x_{11} )</td>
<td>( x_{11} )</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>( x_2 )</td>
<td>( x_{21} )</td>
<td>( x_{22} )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( P_n )</td>
<td>( x_n )</td>
<td>( x_{n1} )</td>
<td>( x_{n2} )</td>
</tr>
<tr>
<td>( y_1 )</td>
<td>( y_2 )</td>
<td>( \cdots )</td>
<td>( y_n )</td>
</tr>
</tbody>
</table>
SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE

An SFE protocol which is secure when parties follow their instructions

Example: A Summation protocol

\[
\begin{array}{c|ccc|c}
P_1 & P_2 & P_n \\
\hline
P_1 & x_1 & x_{11} & x_{12} & \cdots & x_{1n} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
P_n & x_n & x_{n1} & x_{n2} & \cdots & x_{nn} \\
\hline
y_1 & y_2 & \cdots & y_n \\
\end{array}
\]

\[
x_1 = \bigoplus_{j=1}^{n} x_{1j}
\]

\[
x_2 = \bigoplus_{j=1}^{n} x_{2j}
\]

\[
x_n = \bigoplus_{j=1}^{n} x_{nj}
\]

\[
y = \bigoplus_{i=1}^{n} y_i
\]
SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE

An SFE protocol which is secure when parties follow their instructions

**Example:** A Summation protocol

<table>
<thead>
<tr>
<th></th>
<th>P₁</th>
<th>P₂</th>
<th>Pₙ</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>x₁</td>
<td>x₁₁</td>
<td>x₁₁</td>
</tr>
<tr>
<td>P₂</td>
<td>x₂</td>
<td>x₂₁</td>
<td>x₂₂</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Pₙ</td>
<td>xₙ</td>
<td>xₙ₁</td>
<td>xₙ₂</td>
</tr>
</tbody>
</table>

\[ y = \bigoplus_{i=1}^{n} y_i \]

\[ x_1 = \bigoplus_{j=1}^{n} x_{1j} \]

\[ x_2 = \bigoplus_{j=1}^{n} x_{2j} \]

\[ x_n = \bigoplus_{j=1}^{n} x_{nj} \]
SFE with Robust Compen. : Construction

Tools 2/3 : Semi-honest SFE
An SFE protocol which is secure when parties follow their instructions

Assuming a public key infrastructure (commitments/encryption/signatures) there exists a semi-honest SFE protocol $\pi$ for every function which

- Uses only public communication
- Tolerates arbitrary many semi-honest parties
- Terminates in constant rounds
SFE with Robust Compen. : Construction

Tools 3/3 : The GMW Compiler

Compile a semi-honest SFE protocol $\pi$ into (malicious) secure
SFE with Robust Compen.: Construction

Tools 3/3: The GMW Compiler

Compile a semi-honest SFE protocol $\pi$ into (malicious) secure

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Every $P_i$ commits to its input

Rounds 2 ... $\rho_\pi + 1$:
Execute $\pi$ round-by-round so that in each round every party proves (in ZK) that he follows $\pi$
Compile a semi-honest SFE protocol $\pi$ into (malicious) secure SFE with Robust Compen.

**Tools 3/3: The GMW Compiler**

Compile a semi-honest SFE protocol $\pi$ into (malicious) secure SFE with Robust Compen.

**Round 0:**
Setup generation (+ commitments to randomness)

**Round 1:**
Every $P_i$ commits to its input

**Rounds 2 … $\rho_\pi + 1$:**
Execute $\pi$ round-by-round so that in each round every party proves (in ZK) that he follows $\pi$

**Security (with abort):**

- **Privacy:** The parties see the following:
  - Setup
  - Commitments
  - Messages from $\pi$

- **Correctness:**
  - If ZKPs succeed then the parties are indeed following $\pi$
  - Else abort
SFE with Robust Compen. : Construction

Idea: Use “GMW”-like compiler on the Ledger
Idea: Use “GMW”-like compiler on the Ledger

GMW

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Every $P_i$ commits to its input

Rounds 2 ... $\rho_\pi + 1$:
Execute $\pi$ round-by-round so that in each round every party proves (in ZK) that he follows $\pi$
**SFE with Robust Compen.: Construction**

**Idea:** Use “GMW”-like compiler on the Ledger

| GMW | GMW’:
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Round 0:</strong> Setup generation (+ commitments to randomness)</td>
<td><strong>Round 0:</strong> Setup generation (+ commitments to randomness)</td>
</tr>
<tr>
<td><strong>Round 1:</strong> Every $P_i$ commits to its input</td>
<td><strong>Round 1:</strong> Do nothing</td>
</tr>
<tr>
<td><strong>Rounds 2 \ldots \rho_{\pi} + 1:</strong> Execute $\pi$ round-by-round so that in each round every party proves (in ZK) that he follows $\pi$</td>
<td><strong>Rounds 3 \ldots \rho_{\pi} + 2:</strong> Execute $\pi$ round-by-round so that in each round every party proves (in NIZK) that he follows $\pi$</td>
</tr>
</tbody>
</table>
SFE with Robust Compen. : Construction

Idea: Use “GMW”-like compiler on the Ledger

GMW’:

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every $P_i$ commits to its input and broadcasts his view of the public setup.

Rounds 3 ... $\rho_{\pi} + 2$: Execute $\pi$ round-by-round so that in each round every party proves (in NIZK) that the follows $\pi$
Idea: Use “GMW”-like compiler on the Ledger

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every $P_i$ commits to its input and broadcasts his view of the public setup.

Rounds 3 … $\rho_{\pi} + 2$:
Execute $\pi$ round-by-round so that in each round every party proves (in NIZK) that the follows $\pi$
SFE with Robust Compen. : Construction

Idea: Use “GMW”-like compiler on the Ledger

**GMW’:**

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every $P_i$ commits to its input and broadcasts his view of the public setup.

Rounds 3 ... $\rho_\pi + 2$:
Execute $\pi$ round-by-round so that in each round every party proves (in NIZK) that the follows $\pi$
SFE with Robust Compen. : Construction

Idea: Use “GMW”-like compiler on the Ledger

GMW’:

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every $P_i$ commits to its input and broadcasts his view of the public setup.

Rounds 3 ... $\rho_\pi + 2$:
Execute $\pi$ round-by-round so that in each round every party proves (in NIZK) that the follows $\pi$

SFE with Robust Compensation

Round 0:
Setup generation (+ commitments to randomness)
SFE with Robust Compen. : Construction

Idea: Use “GMW”-like compiler on the Ledger

**GMW’**:  
Round 0: Setup generation (+ commitments to randomness)

Round 1: Do nothing

Round 2: Every $P_i$ commits to its input and broadcasts his view of the public setup.

Rounds 3 ... $\rho_{\pi} + 2$: Execute $\pi$ round-by-round so that in each round every party proves (in NIZK) that the follows $\pi$

**SFE with Robust Compensation**

Round 0: Setup generation (+ commitments to randomness)

Round 1: Every party $P_i$ makes $n \cdot \rho_{\pi} + 1$ special 1-coin transactions $B_{(i,j,r)}$:  
- $P_j$ can spend coin in round $r$
- ref needs to have the protocol ID
- $R$ is true if the transaction which spends the coin includes a valid $r$-round message for $P_j$
Idea: Use “GMW”-like compiler on the Ledger

**GMW’:**

**Round 0:**
Setup generation (+ commitments to randomness)

**Round 1:**
Do nothing

**Round 2:**
Every $P_i$ commits to its input and broadcasts his view of the public setup.

**Rounds 3 ... $\rho_\pi + 2$:**
Execute $\pi$ round-by-round so that in each round every party proves (in NIZK) that the follows $\pi$

**SFE with Robust Compensation**

**Round 0:**
Setup generation (+ commitments to randomness)

**Round 1:**
Every party $P_i$ makes $n \cdot \rho_\pi + 1$ special 1-coin transactions $B_{(i,j,r)}$:
- $P_j$ can spend coin in round $r$
- ref needs to have the protocol ID
- $R$ is true if the transaction which spends the coin includes a valid $r$-round message for $P_j$

**Rounds 3 ... $\rho_\pi + 2$:** Execute $\text{GMW}(\pi)$ round-by-round so that in each round $r$ every party spends all its round $r$ referenced coins by a transaction which includes the round $r$ message in $\text{GMW}(\pi)$. 
SFE with Robust Compensation: Construction

**Idea:** Use “GMW”-like compiler on the Ledger

**GMW’:**

**Round 0:**
Setup generation (+ commitments to randomness)

**Round 1:**
Do nothing

**Round 2:**
Every $P_i$ commits to its input and broadcasts his view of the public setup.

**Rounds 3 … $\rho_{\pi} + 2$:**
Execute $\pi$ round-by-round so that in each round $r$ every party proves (in NIZK) that the follows:
- $P_j$ can spend coin in round $r$
- ref needs to have the protocol ID
- $R$ is true if the transaction which spends the coin includes a valid $r$-round message for $P_j$

**Round 0:**
Setup generation (+ commitments to randomness)

**Round 1:** Every party $P_i$ makes $n \cdot \rho_{\pi} + 1$ special 1-coin transactions $B_{i,j,r}$:
- $P_j$ can spend coin in round $r$
- ref needs to have the protocol ID
- $R$ is true if the transaction which spends the coin includes a valid $r$-round message for $P_j$

**Rounds 3 … $\rho_{\pi} + 2$:** Execute $GMW(\pi)$ round-by-round so that in each round $r$ every party spends all its round $r$ referenced coins by a transaction which includes the round $r$ message in $GMW(\pi)$.

*Validate(.) executes the code of an extra party without inputs in GMW and rejects if abort.*
Security with Robust Compensation.

• Case 1: The adversary correctly makes all the “committing” transactions in Round 1

  • If no party cheats then every party claims from each of the other parties as many coins as he deposited by simply executing his protocol.

  • If some party $P_j$ cheats, then every party still claims all his coins as above + all the committed coins that $P_j$ cannot spend as he did not execute his protocol.
SFE with Robust Compensation. : Construction

Security with Robust Compensation.

- **Case 2**: Some corrupted party does not make (consistent) transactions in Round 1
  - e.g. aborts or commits to a different setup.
Security with Robust Compensation.

- **Case 2:** Some corrupted party does not make (consistent) transactions in Round 1
  - e.g. aborts or commits to a different setup.

  ... seems to have similar issue as before ...
Security with Robust Compensation.

• **Case 2:** Some corrupted party does not make (consistent) transactions in Round 1
  • e.g. aborts or commits to a different setup.

  ... seems to have similar issue as before ...

• **Solution:** The validation predicate can be changed as:
  • Separates the parties into “islands” of consistent setups (depending on their Round-1 transactions).
  • For each island $I \subseteq [n]$: Compute the function among parties in $I$ (with all other parties’ input being 0)
Idea: Use “GMW”-like compiler on the Ledger

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every $P_i$ commits to its input and broadcasts his view of the public setup.

Rounds $3 \ldots \rho_{\pi} + 2$:
Execute $\pi$ round-by-round so that in each round every party proves (in NIZK) that the follows $\pi$

Round 1: Every party $P_i$ makes $n \cdot \rho_{\pi} + 1$ special 1-coin transactions $B_{(i,j,r)}$:
- $P_j$ can spend coin in round $r$
- ref needs to have the protocol ID
- $R$ is true if the transaction which spends the coin includes a valid $r$-round message for $P_j$

Rounds $2 \ldots \rho_{\pi} + 2$: Execute $\text{GMW}(\pi)$ round-by-round so that in each round $r$ every party spends all its round $r$ referenced coins by a transaction which includes the round $r$ message in $\text{GMW}(\pi)$.
SFE with Robust Compen.: Construction

Idea: Use “GMW”-like compiler on the Ledger

GMW':

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Do nothing

Round 2:
Every $P_i$ commits to its input and broadcasts his view of the public setup.

Rounds 3 ... $\rho_\pi + 2$:
Execute $\pi$ round-by-round so that in each round every party proves (in NIZK) that the follows $\pi$

SFE with Robust Compensation

Round 0:
Setup generation (+ commitments to randomness)

Round 1:
Every party $P_i$ makes $n \cdot \rho_\pi + 1$ special 1-coin transactions $B_{(i,j,r)}$:
- $P_j$ can spend coin in round $r$
- ref needs to have the protocol ID
- $R$ is true if the transaction which spends the coin includes a valid $r$-round message for $P_j$

Rounds 2 ... $\rho_\pi + 2$: Execute GMW($\pi$) round-by-round so that in each round $r$ every party spends all its round $r$ referenced coins by a transaction which includes the round $r$ message in GMW($\pi$).
SFE with Robust Compens.: Construction

Security with Robust Compensation.

• **Case 2:** Some corrupted party does not make (consistent) transaction in Round 1
  • e.g. aborts or commits to a different setup.

  … seems to have similar issue as before …

• **Solution:** The validation predicate can be changed as:
  • Separates the parties into “islands” of consistent setups (depending on their Round-1 transactions).
  • For each island $I \subseteq [n]$: Compute the function among parties in $I$ (with all other parties’ input being 0)

• All honest parties are on the same island
• Corrupted parties can choose to play with the honest parties or participate in a computation independent of honest inputs.
Crypto On Blockchain

Outline

• The functionality offered by blockchains

• Leveraging Security Loss with Coins
  … in Secure Function Evaluation (SFE)

• A formal cryptographic (UC) model for security proofs
A Formal Model: GUC

Ideal World

\[ F^f \]

\[ P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_n \]

Real World

\[ \approx \]

\[ \pi_1(x_1) \rightarrow \pi_2(x_2) \rightarrow \ldots \rightarrow \pi_n(x_n) \]
A Formal Model: GUC

Ideal World

Real World

\[ F^f \]

\[ \approx \]

\[ \prod_{i=1}^{n} P_i \]

\[ \prod_{i=1}^{n} \pi_i(x_i) \]
A Formal Model: GUC

Ideal World

Real World

\[ \pi_1(x_1) \rightarrow P_1 \rightarrow \pi_2(x_2) \rightarrow P_2 \rightarrow \cdots \rightarrow \pi_n(x_n) \rightarrow P_n \]

\[ \mathcal{F}^f \]

\[ G_{\text{Ledger}} \]
A Formal Model: GUC

Ideal World

???

Real World

\( \mathcal{F}^f \)

\[ \pi_1(x_1) \quad \pi_2(x_2) \quad \ldots \quad \pi_n(x_n) \]

\( \approx \)

\[ \pi_1(x_1) \quad \pi_2(x_2) \quad \ldots \quad \pi_n(x_n) \]

\( G_{\text{Ledger}} \)

Should capture all properties we want from \( \pi \)
A Formal Model: GUC

Ideal World

\[ W(F^f) \]

\[ F^f \]

\[ P_1 \]

\[ P_2 \]

\[ \ldots \]

\[ P_n \]

Real World

\[ \pi_1(x_1) \]

\[ \pi_2(x_2) \]

\[ \ldots \]

\[ \pi_n(x_n) \]

\[ G_{\text{Ledger}} \]

Should capture all properties we want from \( \pi \)
Benefits of this Modeling
Benefits of this Modeling

- A single abstraction of the functionality offered by cryptocurrencies
  - Advanced transactions correspond to an advanced validation predicate

- A definition of *fair compensation* as a (UC) functionality-wrapper forces us to be precise
  - An explicit formation of synchrony with a single global clock (capturing what protocols assume in reality).

- Compatibility with standard (formal) analysis of crypto protocols

- A (universal) composition theorem
A Formal Model: GUC

W(F^f) \rightarrow G_{Ledger}

P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_n

π_1(x_1) \rightarrow \pi_2(x_2) \rightarrow \ldots \rightarrow \pi_n(x_n)

Ideal World

Real World

\approx \pi_1(x_1) \pi_2(x_2) \pi_n(x_n)
A Formal Model: GUC

Ideal World

Real World

$$W(F^f)$$

$$\mathcal{F}^f$$

$$\approx$$

$$\pi_1(x_1) \quad \pi_2(x_2) \quad \cdots \quad \pi_n(x_n)$$
A wrapper functionality $W(F_f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})$
SFE with Robust Compen. : Functionality

A wrapper functionality $W(F_f)$ with three predicates:
- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})$

**Idea**: The predicates are used to filter the adversarial influence

- $Q^{\text{Init}}(\text{State}, \text{Wallet}_i) = True$ iff the $\text{Wallet}_i$ has enough funds
- $Q^{\text{Dlvr}}(\text{State}, \text{Wallet}_i) = True$ iff it is OK to deliver to $P_i$
  - E.g., if $P_i$ does not “owe” money
- $Q^{\text{Abrt}}(\text{State}, \text{Wallet}_i) = True$ iff it is OK for $P_i$ to abort
  - E.g., if $P_i$ has an increase of funds
A wrapper functionality $W(\mathcal{F}_f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})$

**Phase 1: Resource Allocation**
SFE with Robust Compen. : Functionality

A wrapper functionality $W(F_f)$ with three predicates:
- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})$

Phase 1: Resource Allocation

- $\text{ledger}$
- $\text{allocate}$
SFE with Robust Compen. : Functionality

A wrapper functionality $W(F^f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abt}})$

Phase 1: Resource Allocation

$P_i$ → “allocate $P_i$”

$G_{\text{ledger}}$ → $W(F^f)$
SFE with Robust Compen. : Functionality

A wrapper functionality \( W(\mathcal{F}_f) \) with three predicates:

- \((Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abt}})\)

\[ W(\mathcal{F}_f) \]

**Phase 1: Resource Allocation**

\[ \mathcal{F}_f \]
A wrapper functionality $W(\mathcal{F}_f)$ with three predicates:

- $(Q_{\text{Init}}, Q_{\text{Dlvr}}, Q_{\text{Abt}})$

**Phase 1: Resource Allocation**

Create $(PK_i, SK_i) = \text{Gen}(r,1^k)$
A wrapper functionality $W(F_f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlv}}, Q^{\text{Abt}})$

**Phase 1: Resource Allocation**

Create $(PK_i, SK_i) = \text{Gen}(r, 1^k)$
SFE with Robust Compen. : Functionality

A wrapper functionality $W(F_f)$ with three predicates:

- $(Q_{\text{Init}}, Q_{\text{Dlvr}}, Q_{\text{Abrt}})$

### Phase 1: Resource Allocation

**Create** $(PK_i, SK_i) = \text{Gen}(r, 1^k)$
A wrapper functionality $W(F_f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})$

**Phase 1: Resource Allocation**

Create $(PK_i, SK_i) = \text{Gen}(r, 1^k)$
SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}_f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})$

**Phase 2: Input**

\[ \mathcal{G}_{\text{ledger}} \]

\[ W(\mathcal{F}_f) \]

\[ P_i \]

\[ m, \mathcal{F} \rightarrow m \]

\[ m \rightarrow m, \mathcal{F} \]
SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}_f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})$

Phase 2: Input

- $P_i \xrightarrow{\text{"Input, x"}} m, \mathcal{F}$
- $\mathcal{F} \xrightarrow{m} m, \mathcal{F}$
- $m \xrightarrow{m \text{ for Sim}} \mathcal{F}_f$
A wrapper functionality $W(\mathcal{F}^f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})$

**Phase 2: Input**

- $P_i$ \rightarrow \text{"Input, x"} \leftarrow \text{getState}
- $m, \mathcal{F}$ \rightarrow \text{m, } \mathcal{F}^f$
- $m$ \rightarrow \text{m for Sim} \rightarrow \mathcal{F}^f
SFE with Robust Compen. : Functionality

A wrapper functionality $W(\mathcal{F}_f)$ with three predicates:

- $(Q^{Init}, Q^{Dlvr}, Q^{Abrt})$

Phase 2: Input

$W(\mathcal{F}_f)$

"Input, x"

$\text{getState}$

$\text{Gledger}$
SFE with Robust Compen. : Functionality

A wrapper functionality $W(F_f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})$

```
Gledger
```

```
getState
```

```
State
```

```
W(F_f)
```

Phase 2: Input

```
P_i
```

```
"Input, x"
```

```
m, F
```

```
m
```

```
m
```

```
m for Sim
```

```
F_f
```

```
Q^{\text{Init}}(\text{State, PK}_i)
```

```
No
```

```
Yes
```

```
\text{m for Sim}
```
A wrapper functionality \( W(\mathcal{F}_f) \) with three predicates:

- \((Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})\)

\[\text{Phases:} \quad \begin{align*}
P_i &\xrightarrow{\text{"Input, x"}} m, \mathcal{F} \\
m, \mathcal{F} &\xrightarrow{m} m, \mathcal{F} \\
m &\xrightarrow{m \text{ for Sim}} \mathcal{F}_f
\end{align*}\]
A wrapper functionality $W(\mathcal{F}_f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})$
A wrapper functionality \( W_{P1,\ldots,Pn}(F_f) \) with three predicates:

- \((Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abt}})\)

**Phase 3: Output**

\[ \mathcal{G} \text{ledger} \]

\[ W(F_f) \]

\[ \mathcal{F}_f \]
SFE with Robust Compen. : Functionality

A wrapper functionality $W_{P_1,\ldots,P_n}(F_f)$ with three predicates:

- $(Q_{\text{Init}}, Q_{\text{Dlvr}}, Q_{\text{Abrt}})$

Phase 3: Output

Deliver, $(f(x_1),\ldots,f(x_n))$
SFE with Robust Compen.: Functionality

A wrapper functionality $W_{P_1,...,P_n}(F^f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abt}})$

Phase 3: Output

$P_i$

Ready for FairDeliver + Corrupt outputs

$W(F^f)$

Deliver, $(f(x_1),...,f(x_n))$

$G_{\text{ledger}}$

$m, F$

$m$

$m$

$m$ for Sim

$F^f$
SFE with Robust Compen. : Functionality

A wrapper functionality $W_{P_1,\ldots,P_n}(F_f)$ with three predicates:
- $(Q^{\text{Init}}, Q^{\text{Delivr}}, Q^{\text{Abrt}})$

\[ W(F_f) \]

Phase 3: Output

Ready for FairDeliver

+ Corrupt outputs

Deliver, $(f(x_1),\ldots,f(x_n))$

\[ F_f \]

\[ m \]

\[ m \]

\[ m \text{ for Sim} \]

Deliver/Abort $P_i$
A wrapper functionality $\mathcal{W}_{P_1,\ldots,P_n}(\mathcal{F}^f)$ with three predicates:

- $(Q_{\text{Init}}, Q_{\text{Dlvr}}, Q_{\text{Abrt}})$

**SFE with Robust Compen. : Functionality**

**Phase 3: Output**

- Ready for FairDeliver
- Deliver, $(f(x_1),\ldots,f(x_n))$

**GetState**

- Corrupt outputs

**State**

- Deliver/Abort $P_i$

- m, $\mathcal{F}$

- m

- m for Sim

- $\mathcal{F}^f$
A wrapper functionality $W_{P1,...,Pn}(\mathcal{F}^f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abtr}})$

The adversary can deliver to $P_i$ only if $Q^{\text{Dlvr}}(\text{State}, P_i) = \text{True}$

The adversary can make $P_i$ abort only if $Q^{\text{Abtr}}(\text{State}, P_i) = \text{True}$

**Phase 3: Output**

- Deliver, $(f(x_1),...,f(x_n))$
- Deliver/Abort $P_i$
- $m, F$
- $m$ for Sim

Prepared functionality $\mathcal{G}^{\text{ledger}}$
A wrapper functionality $W_{P_1,...,P_n}(\mathcal{F}_f)$ with three predicates:

- $(Q^{\text{Init}}, Q^{\text{Dlvr}}, Q^{\text{Abrt}})$

The adversary can deliver to $P_i$ only if $Q^{\text{Dlvr}}(\text{State, } P_i) = \text{True}$

The adversary can make $P_i$ abort only if $Q^{\text{Abrt}}(\text{State, } P_i) = \text{True}$

Phase 3: Output

Deliver, $(f(x_1),...,f(x_n))$

Deliver/Abort $P_i$

Ready for FairDeliver

Corrupt outputs

$G_{\text{ledger}}$
A Formal Model: GUC

Ideal World

Real World

$\mathcal{W}(\mathcal{F}^f)$

$\mathcal{F}^f$

$\mathcal{G}_{\text{Ledger}}$

$\approx \pi_1(x_1)$

$\pi_1(x_1)$

$\pi_2(x_2)$

$\pi_2(x_2)$

$\pi_n(x_n)$

$\pi_n(x_n)$
Take Away Message and Open Directions

• Bitcoin opens new directions for cryptographic models
  - Adding a reward/punishment mechanism restricts the set of likely attacks
  - Limitations of crypto should be reconsidered (Impossibilities/Efficiencies)
• The choice of the model makes a difference when suggesting a solution
  - Safe strategy: Rectify the cryptographic model (Bonus: compatibility)
Take Away Message and Open Directions

• Bitcoin opens new directions for cryptographic models
  - Adding a reward/punishment mechanism restricts the set of likely attacks
  - Limitations of crypto should be reconsidered (Impossibilities/Efficiencies)
• The choice of the model makes a difference when suggesting a solution
  - Safe strategy: Rectify the cryptographic model (Bonus: compatibility)

Future directions

• A game theoretic analysis might allow us to improve existing results
• What more can we get from Bitcoin?
• The right model for exploring its rational aspects?