Very-Efficient Simulatable Flipping of Many Coins into-a-Well

(and a New Universally-Composable Commitment Scheme)

Luís Brandão[1,2,*]

[1]University of Lisbon (Portugal)

[2]Carnegie Mellon University (USA)

Presented at ***Public Key Cryptography***

March 09, 2016 @ Taipei, Taiwan

Supported as a Ph.D. student at FCUL-DI and CMU-ECE by the Fundação para a Ciência e a Tecnologia (FCT) (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program, under Grant SFRH/BD/33770/2009.

Ciências ULisboa

Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL
AN INTERNATIONAL PARTNERSHIP

Electrical & Computer ENGINEERING
Carnegie Mellon

FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA
PORTUGAL

Roadmap

1. **Simulatable coin-flipping and commitments**

2. **Protocol #1: coin-flipping (simulatable with rewinding)**

3. **Protocol #2: UC Commitment Scheme**

4. **Open questions / research directions**

Roadmap

1. **Simulatable coin-flipping and commitments**

2. **Protocol #1: coin-flipping (simulatable with rewinding)**

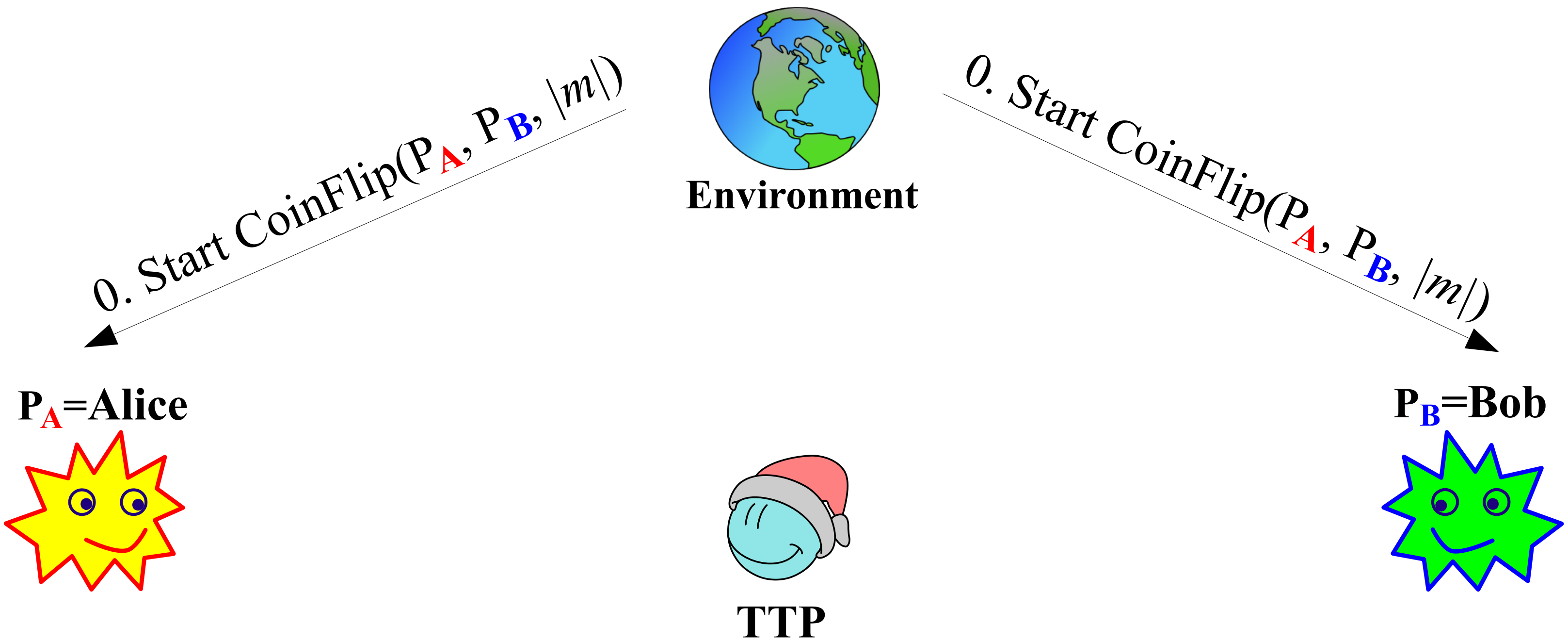
3. **Protocol #2: UC Commitment Scheme**

4. **Open questions / research directions**

An ideal coin-flipping

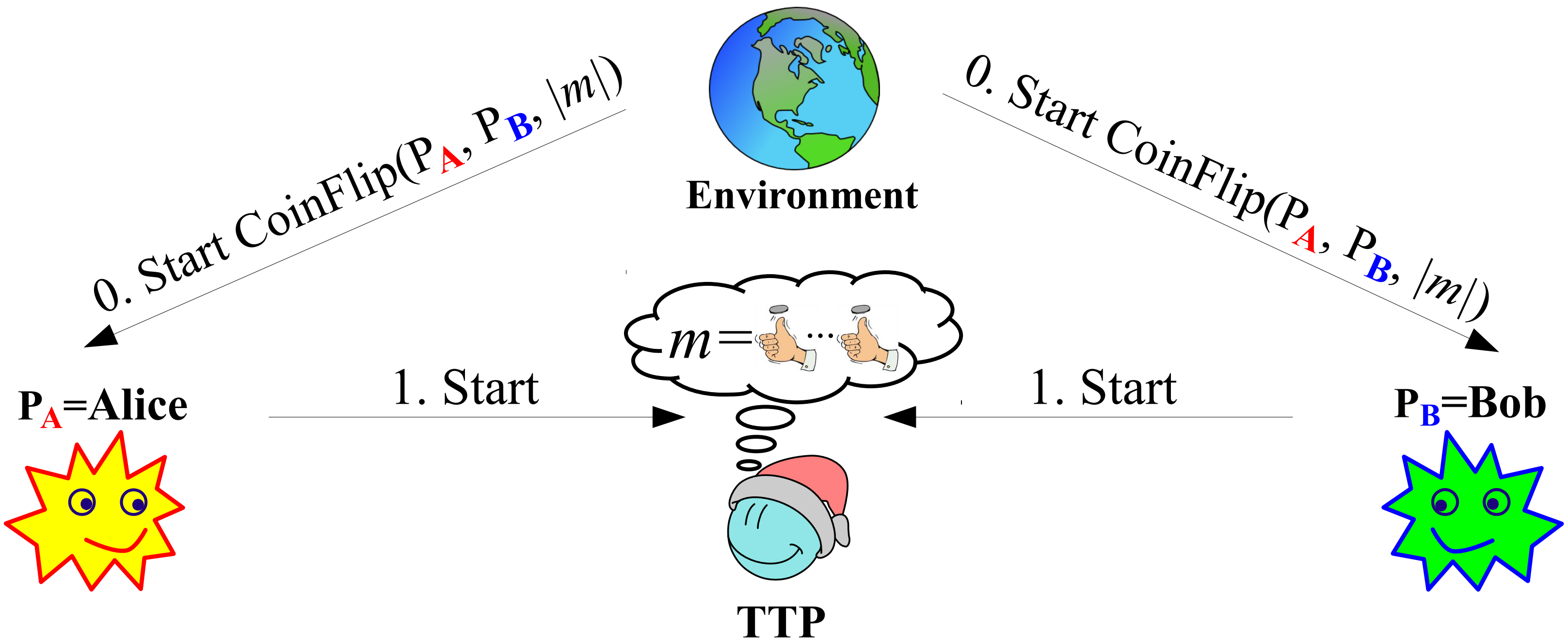
An ideal coin-flipping

Legend:
TTP = trusted third party



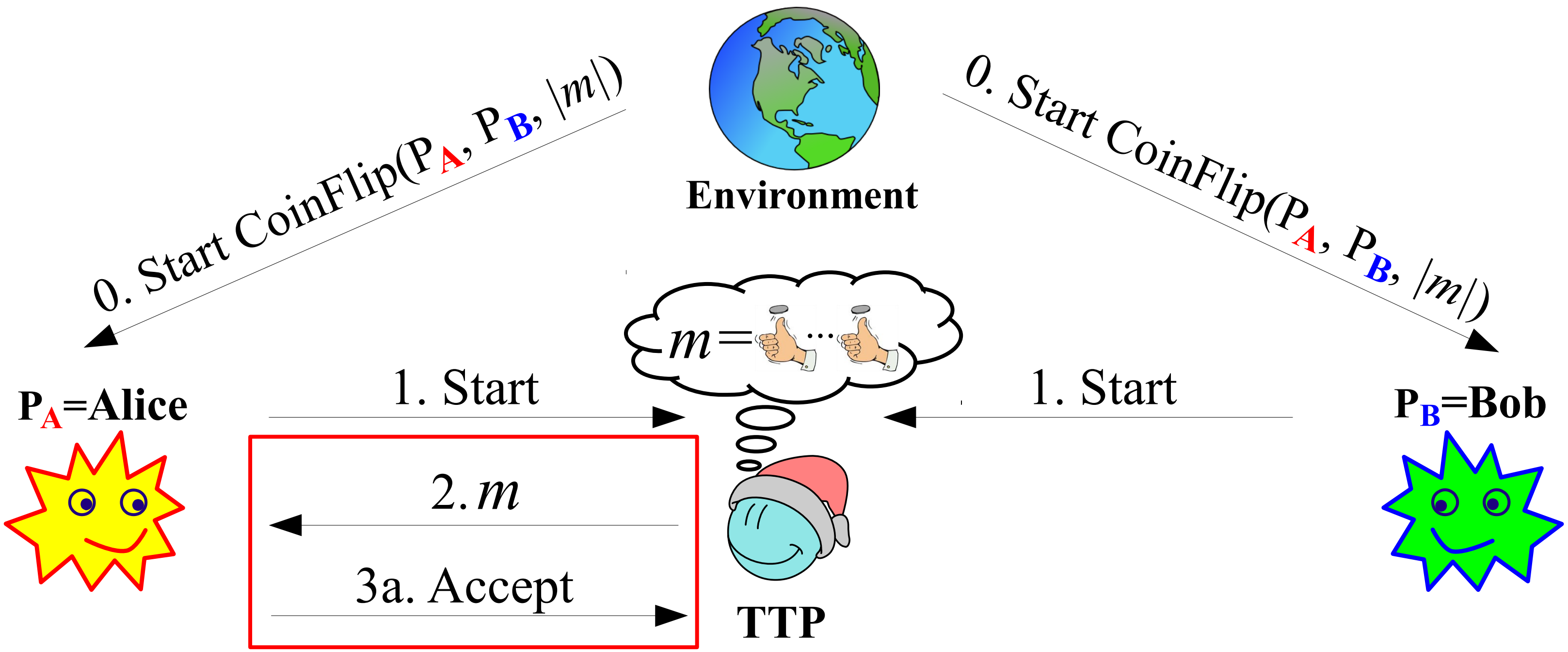
An ideal coin-flipping

Legend:
TTP = trusted third party



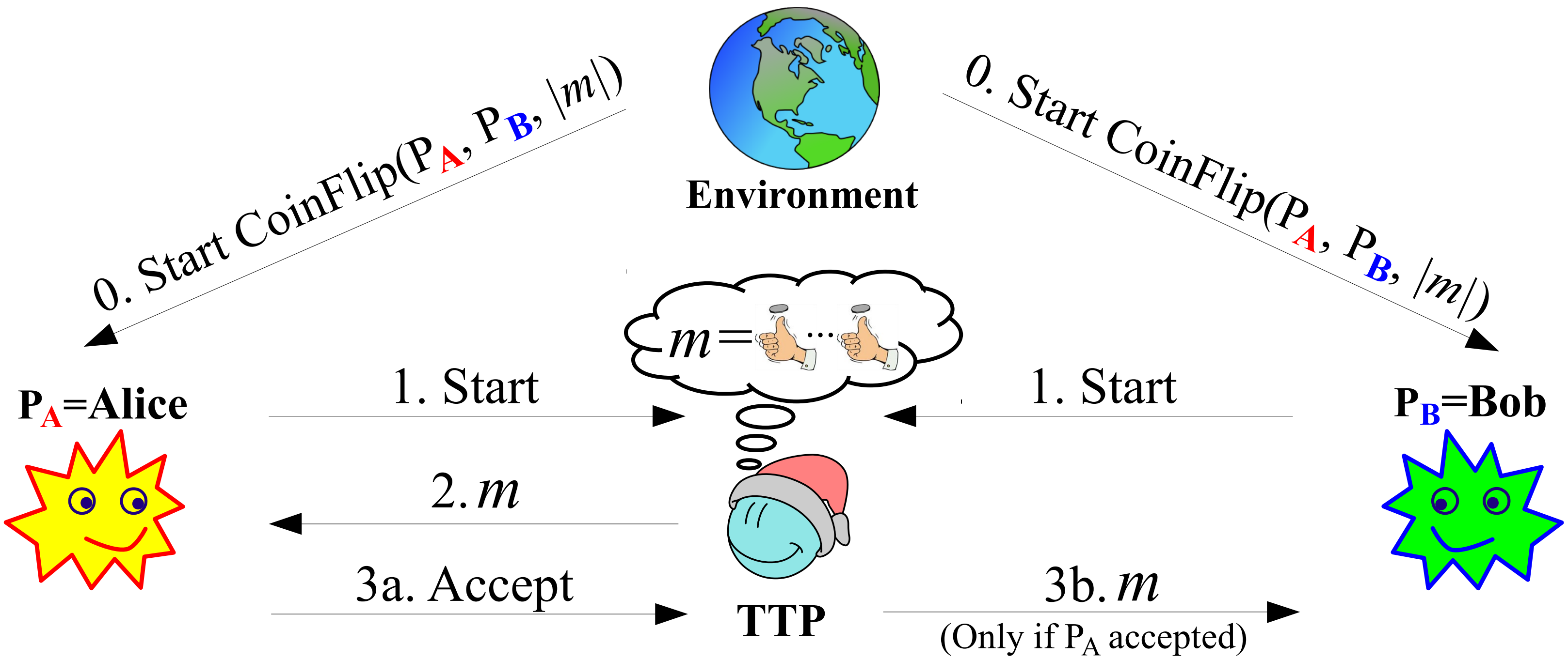
An ideal coin-flipping into-a-well

Legend:
TTP = trusted third party



An ideal coin-flipping into-a-well

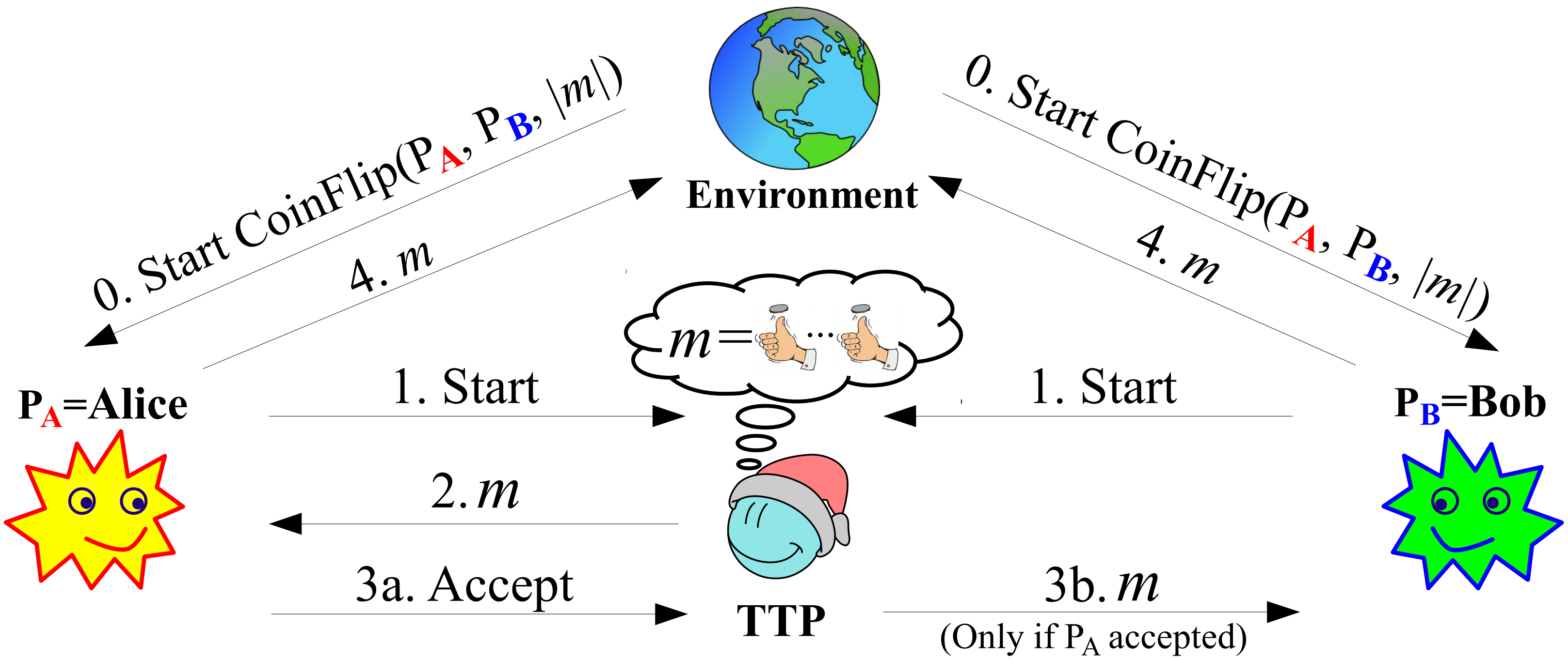
Legend:
TTP = trusted third party



Part 1
Ideal CF
TradTemp
Ext-Equiv
Intuition

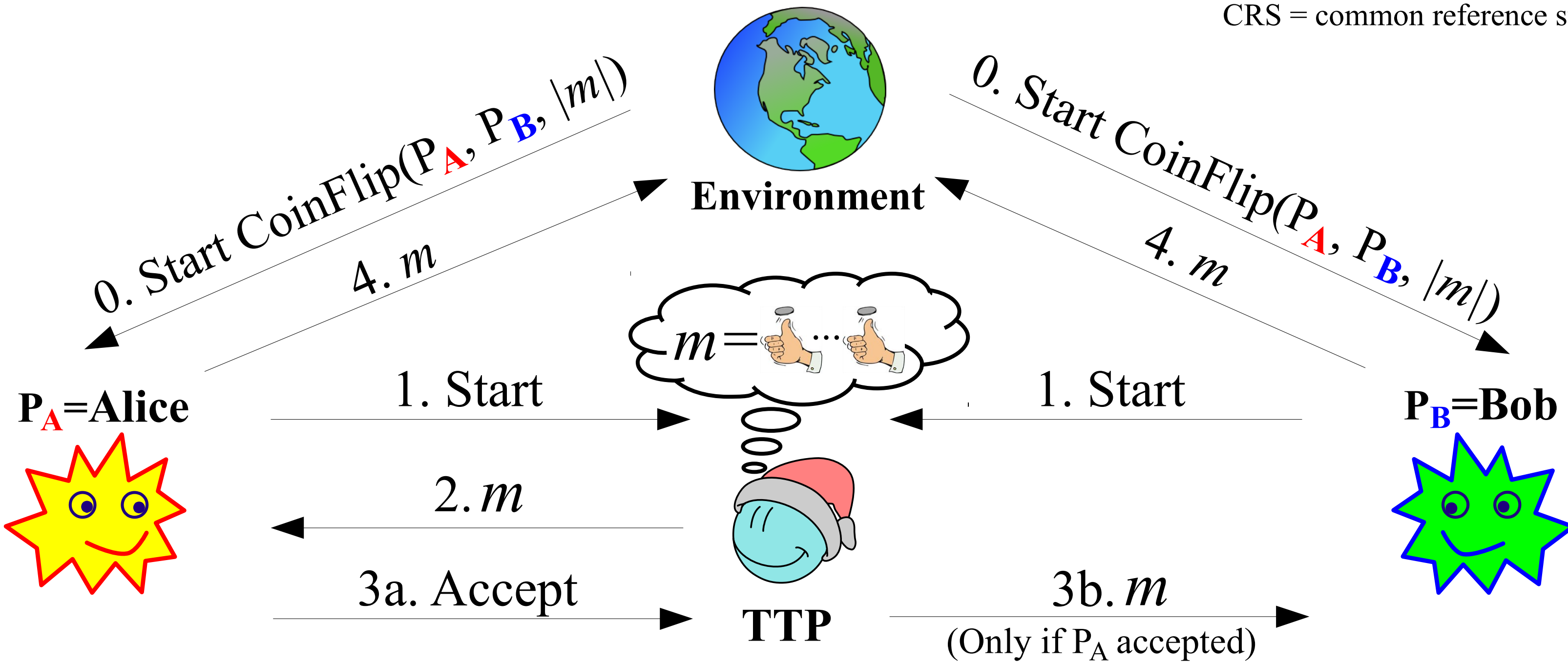
An ideal coin-flipping into-a-well

Legend:
TTP = trusted third party



An ideal coin-flipping into-a-well

Legend:
TTP = trusted third party
CRS = common reference string

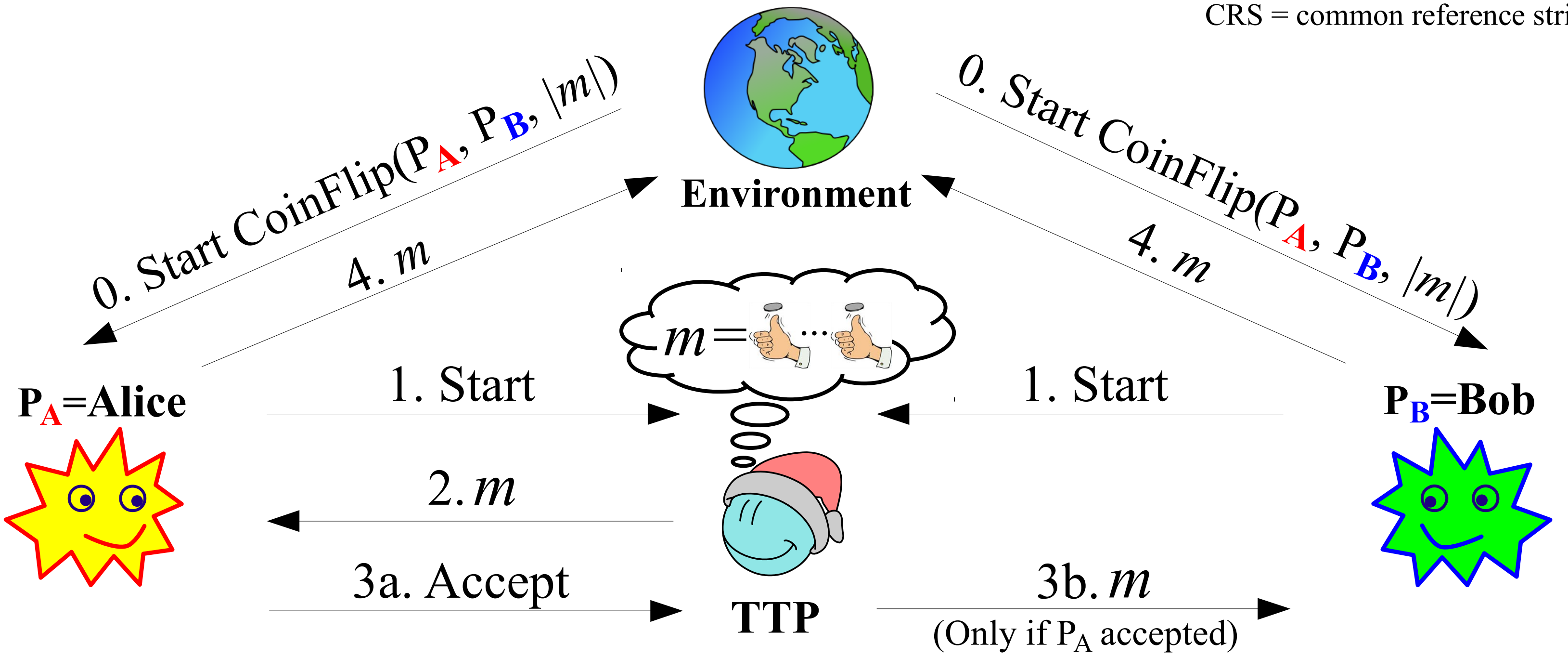


Example motivations

- Real world decisions (e.g., who gets the car? [Blum83])
- Enable probabilistic output of external two-party protocol
- Random string (e.g., CRS) for another simulatable protocol

An ideal coin-flipping into-a-well

Legend:
TTP = trusted third party
CRS = common reference string



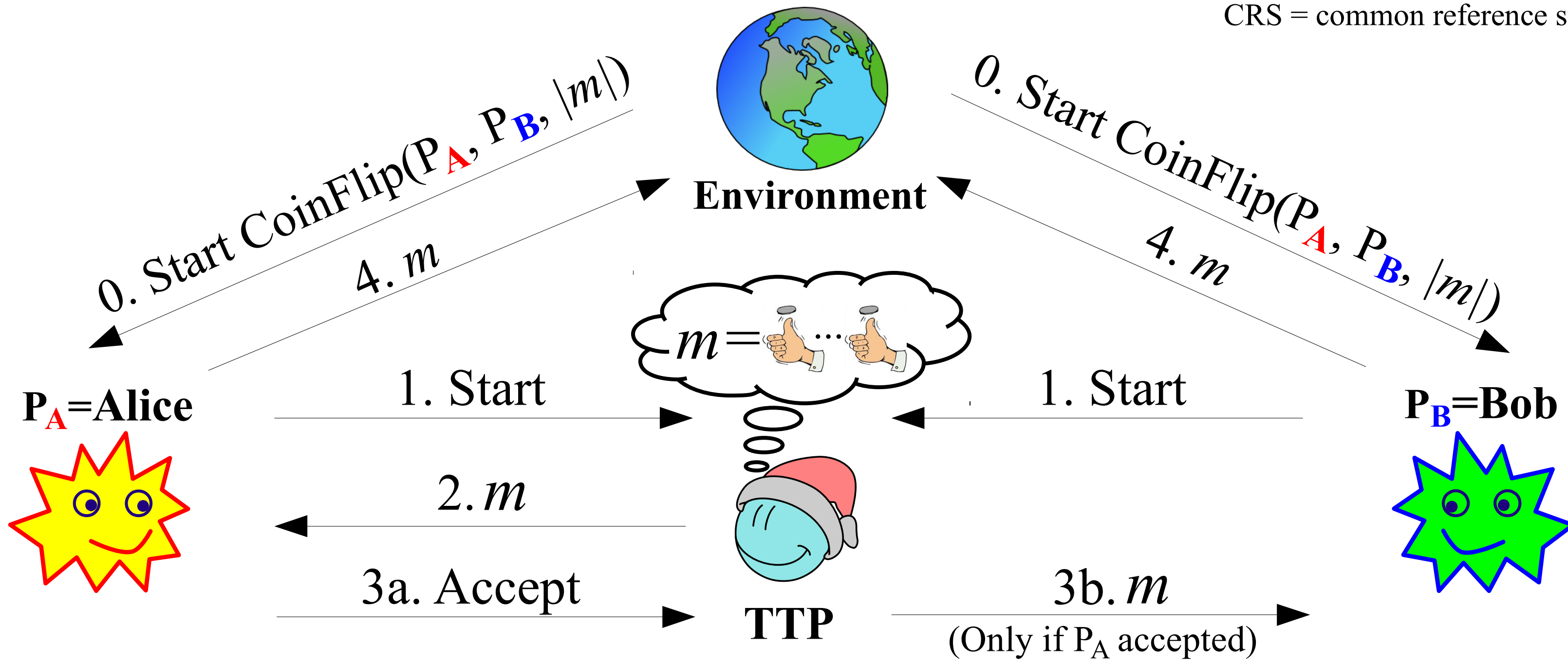
Example motivations

- Real world decisions (e.g., who gets the car? [Blum83])
- Enable probabilistic output of external two-party protocol
- Random string (e.g., CRS) for another simulatable protocol

Research question: How to perform two-party coin-flipping, i.e., without TTP, efficiently for many coins in parallel, within the ideal/real **simulation** paradigm?

An ideal coin-flipping into-a-well

Legend:
TTP = trusted third party
CRS = common reference string



Example motivations

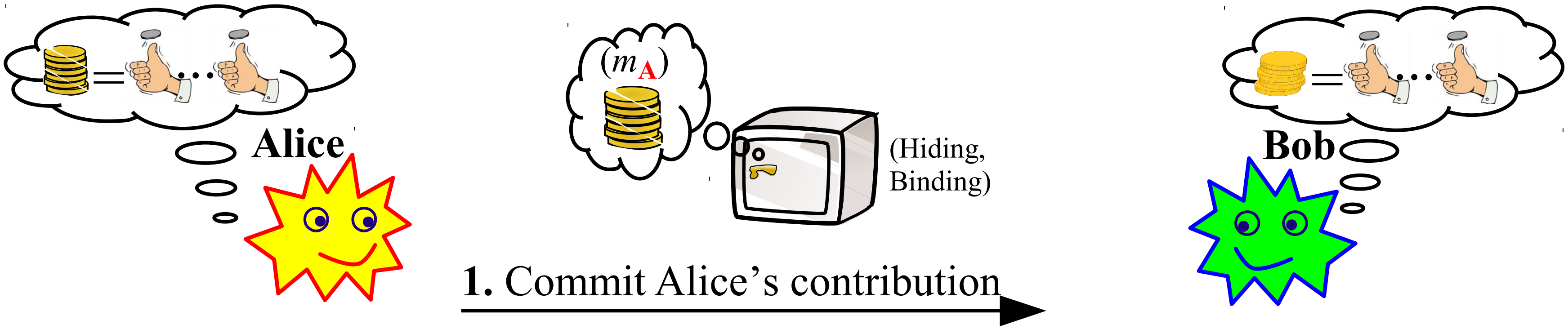
- Real world decisions (e.g., who gets the car? [Blum83])
- Enable probabilistic output of external two-party protocol
- Random string (e.g., CRS) for another simulatable protocol

Research question: How to perform two-party coin-flipping, i.e., without TTP, efficiently for many coins in parallel, within the ideal/real **simulation** paradigm?

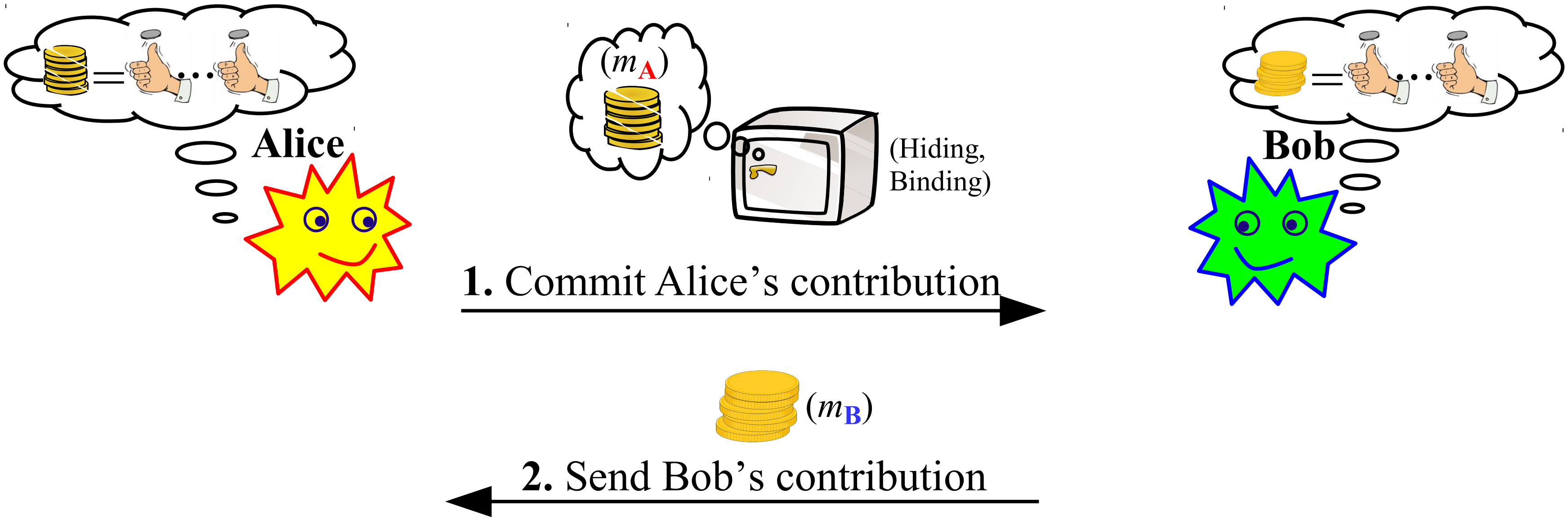
(Adversarial model: static, malicious, computational)

An early two-party coin-flipping protocol [Blum81-83]

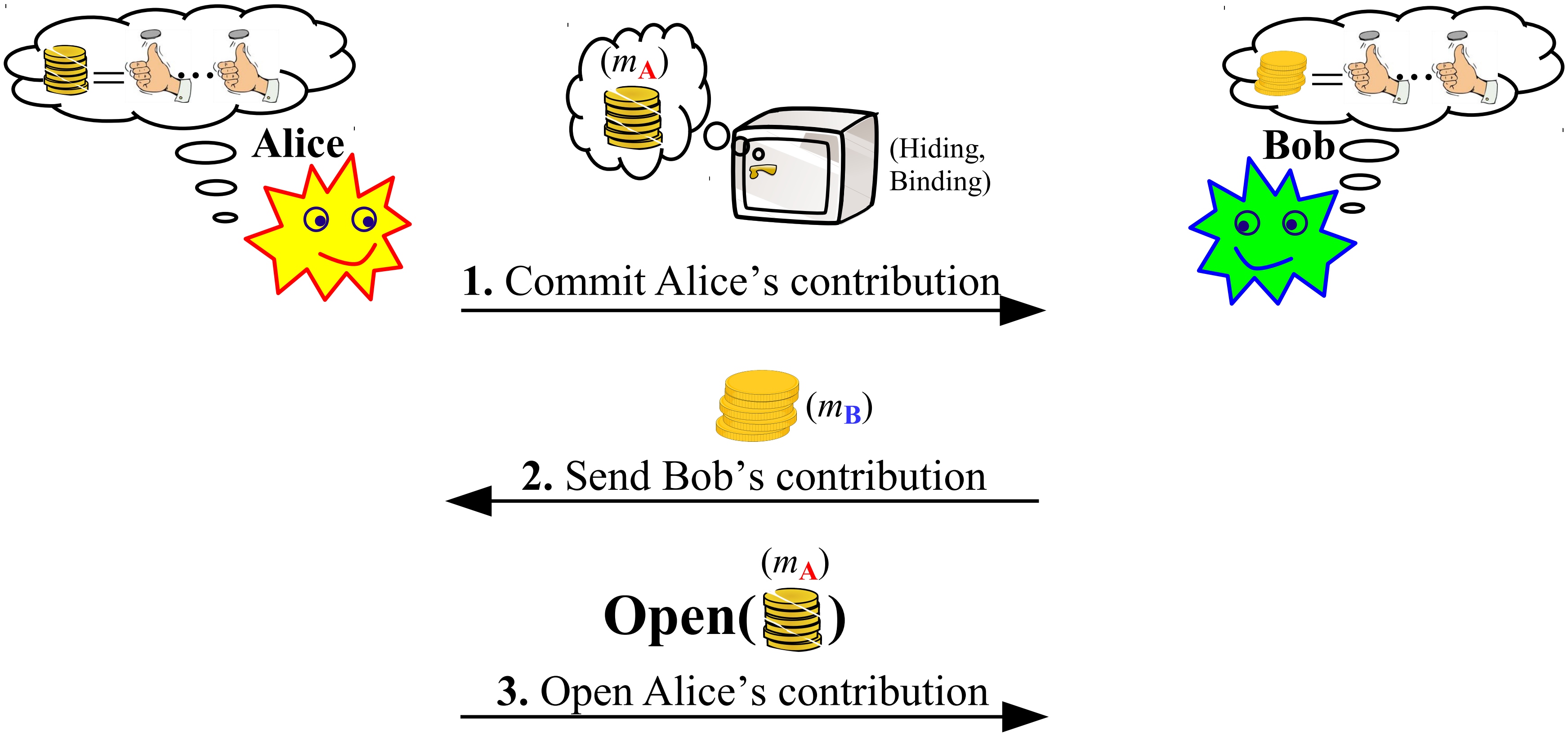
An early two-party coin-flipping protocol [Blum81-83]



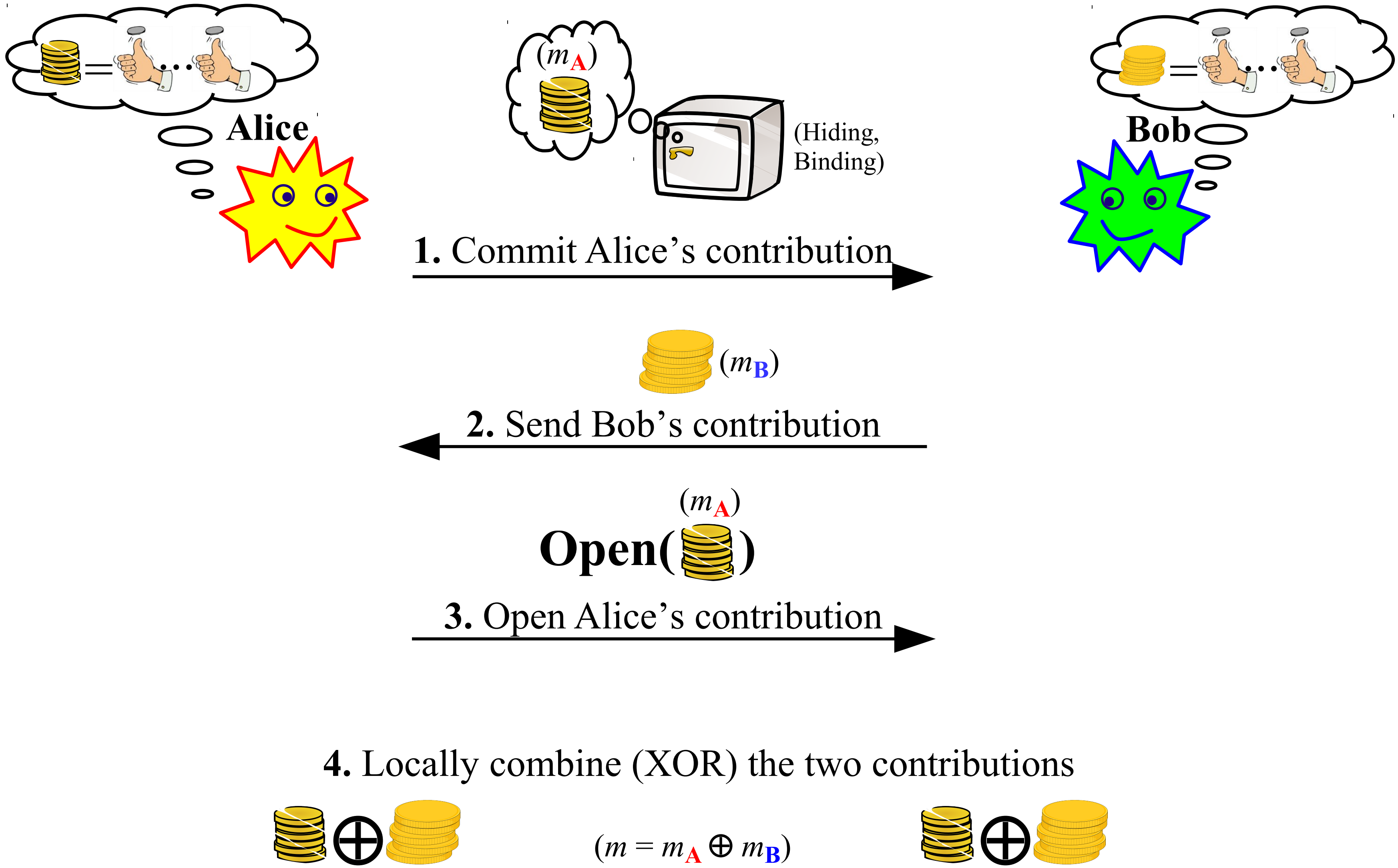
An early two-party coin-flipping protocol [Blum81-83]



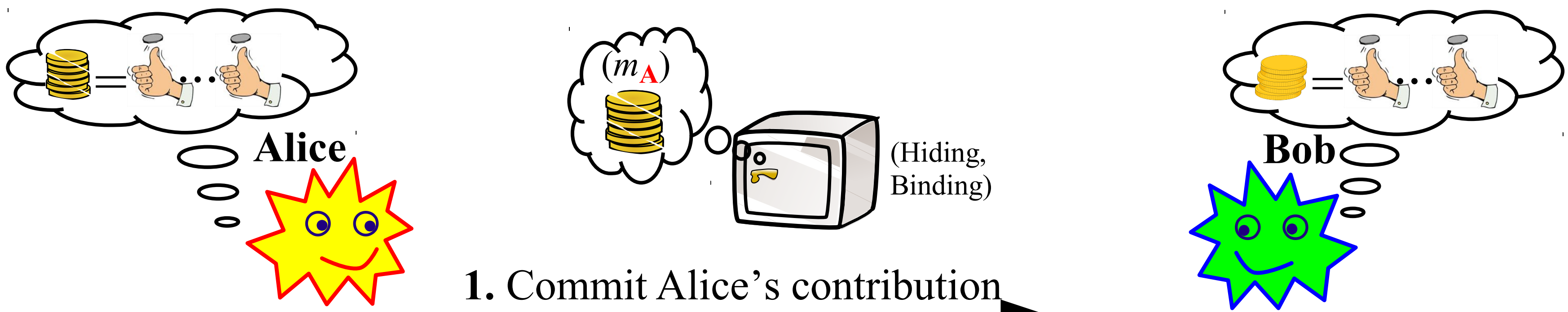
An early two-party coin-flipping protocol [Blum81-83]



An early two-party coin-flipping protocol [Blum81-83]



An early two-party coin-flipping protocol [Blum81-83]



1. Commit Alice's contribution →

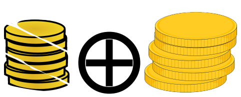
← **2.** Send Bob's contribution

(m_A)

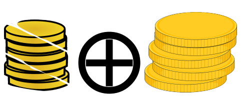
Open(**)**

3. Open Alice's contribution →

4. Locally combine (XOR) the two contributions

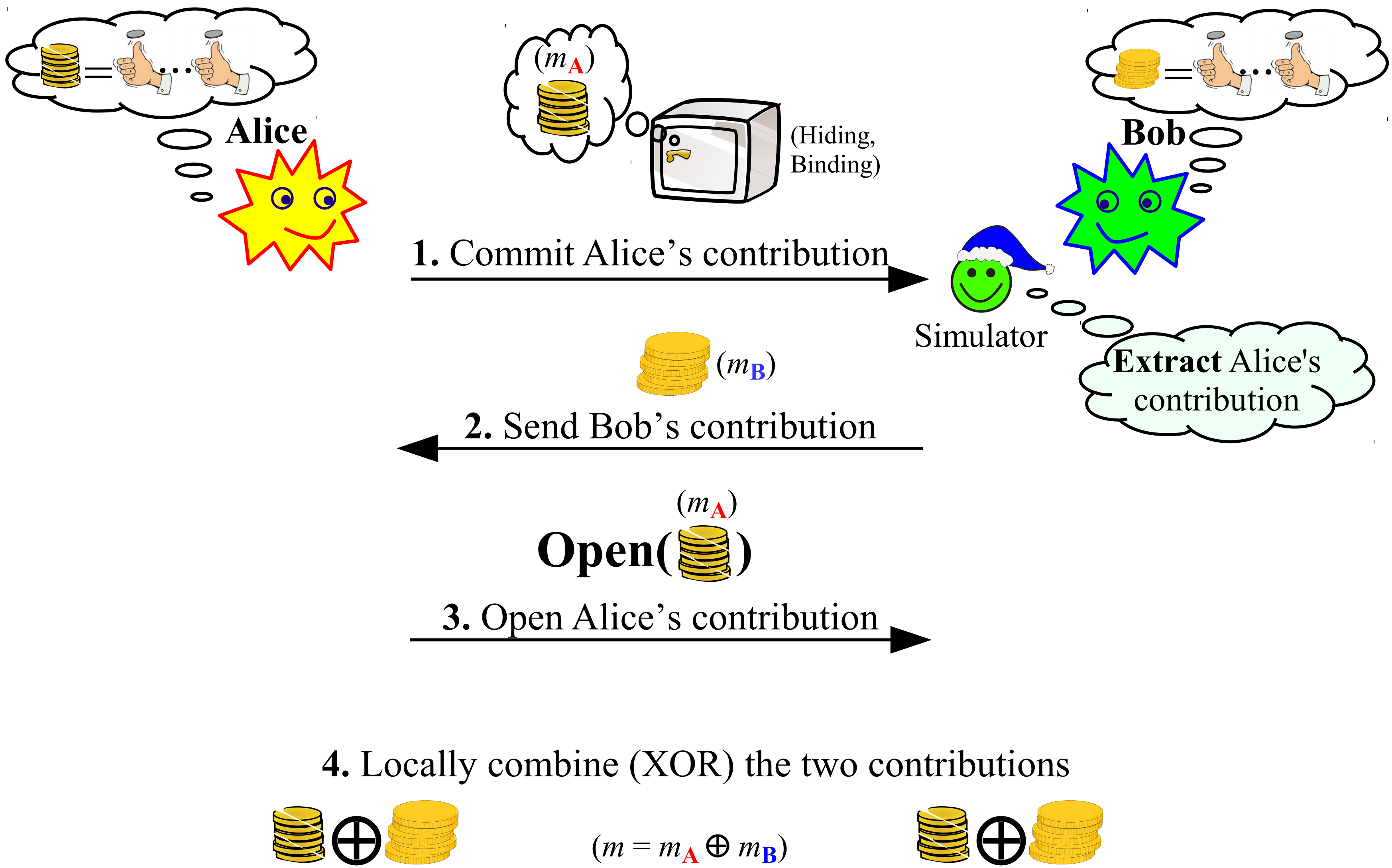


$(m = m_A \oplus m_B)$



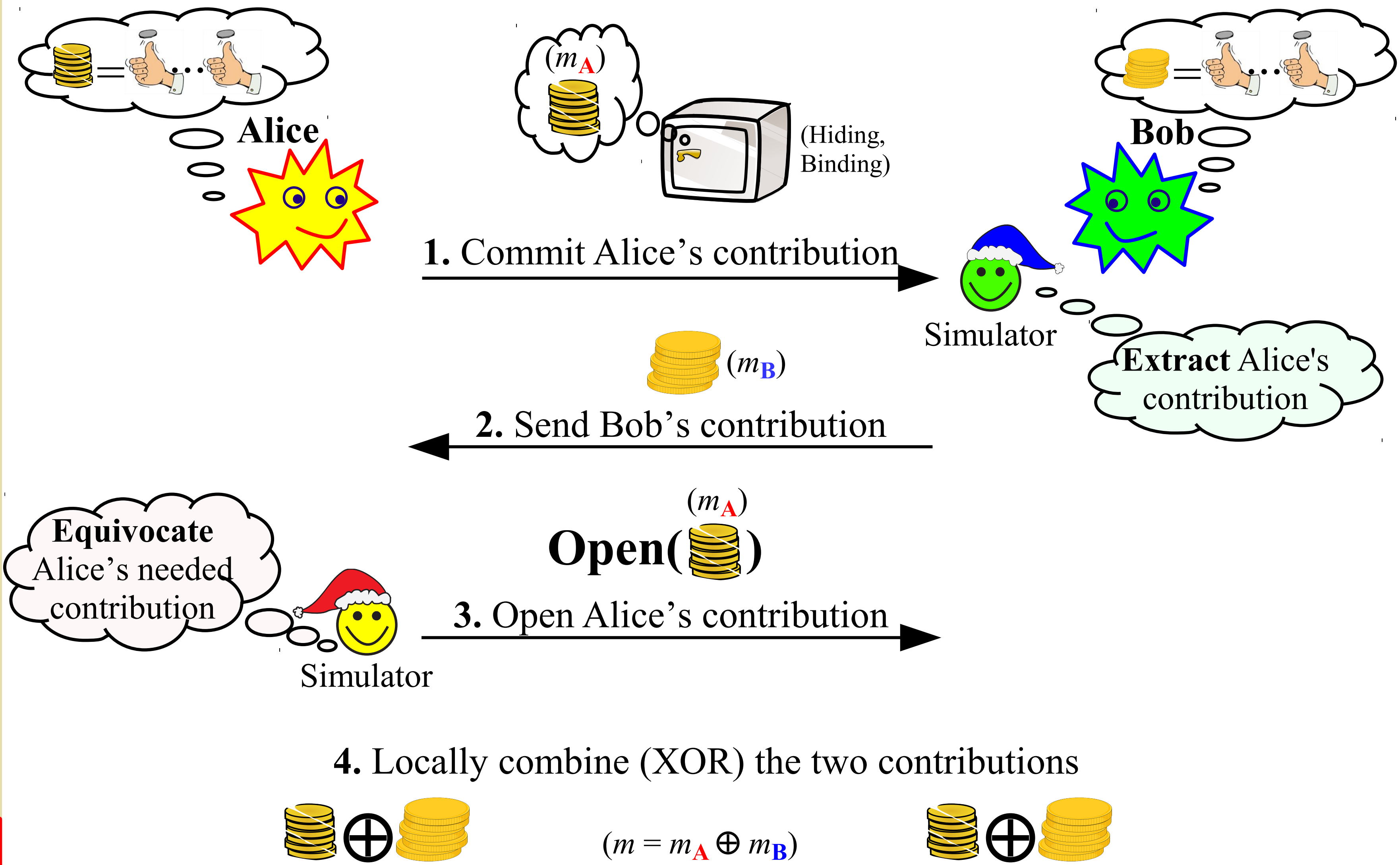
Simulatability ⇒ In a simulation, the Simulator (Sim) can induce any desired outcome (the one decided by TTP in ideal world).

An early two-party coin-flipping protocol [Blum81-83]



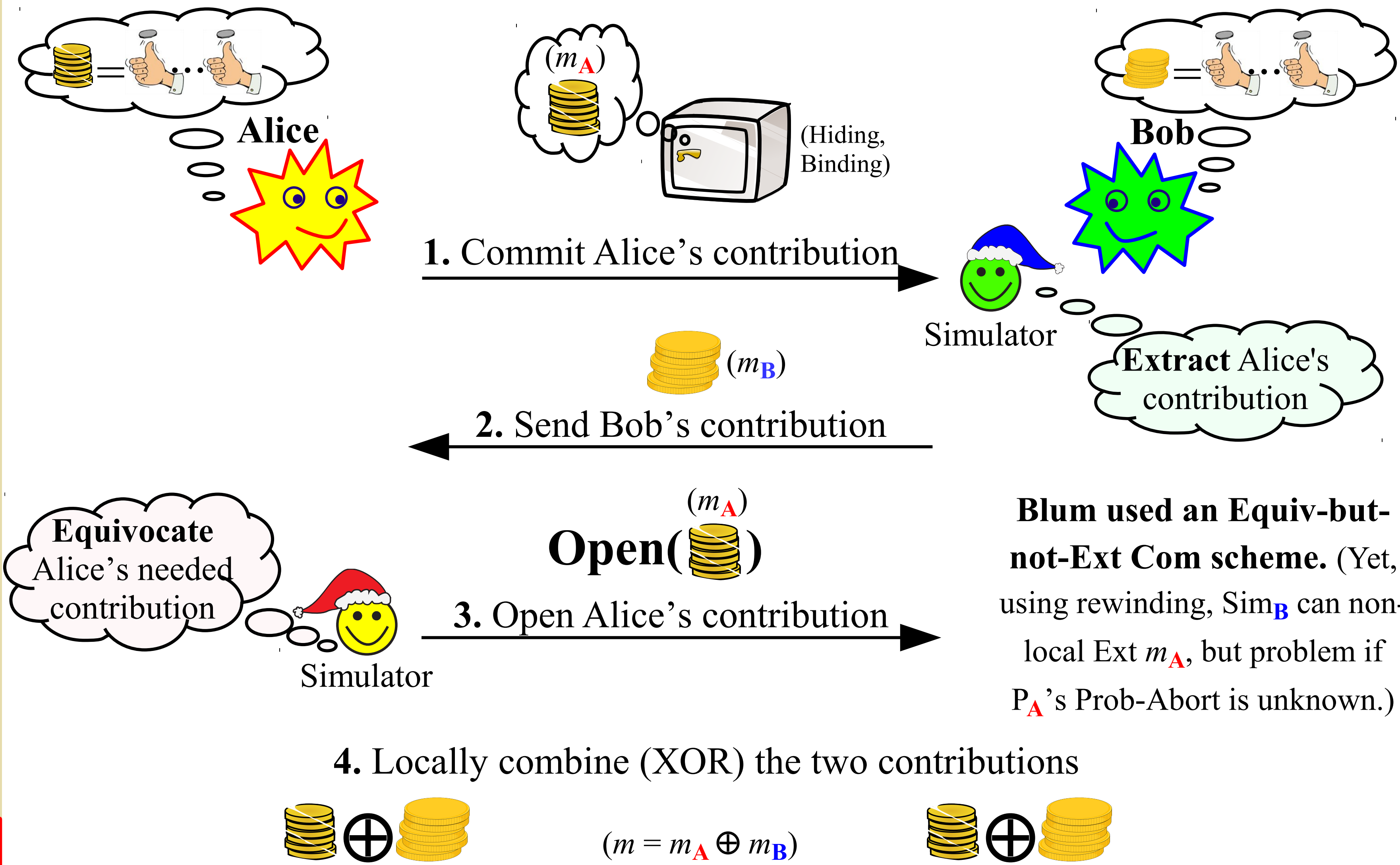
Simulatability ⇒ In a simulation, the Simulator (Sim) can induce any desired outcome (the one decided by TTP in ideal world).

An early two-party coin-flipping protocol [Blum81-83]



Simulatability ⇒ In a simulation, the Simulator (Sim) can induce any desired outcome (the one decided by TTP in ideal world).

An early two-party coin-flipping protocol [Blum81-83]



Blum used an Equiv-but-not-Ext Com scheme. (Yet, using rewinding, Sim_B can non-local Ext m_A, but problem if P_A's Prob-Abort is unknown.)

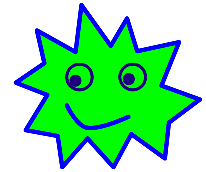
Simulatability ⇒ In a simulation, the Simulator (Sim) can induce any desired outcome (the one decided by TTP in ideal world).

Example of an Ext-and-Equiv Com Scheme [Lin03]

Example of an Ext-and-Equiv Com Scheme [Lin03]

Sender

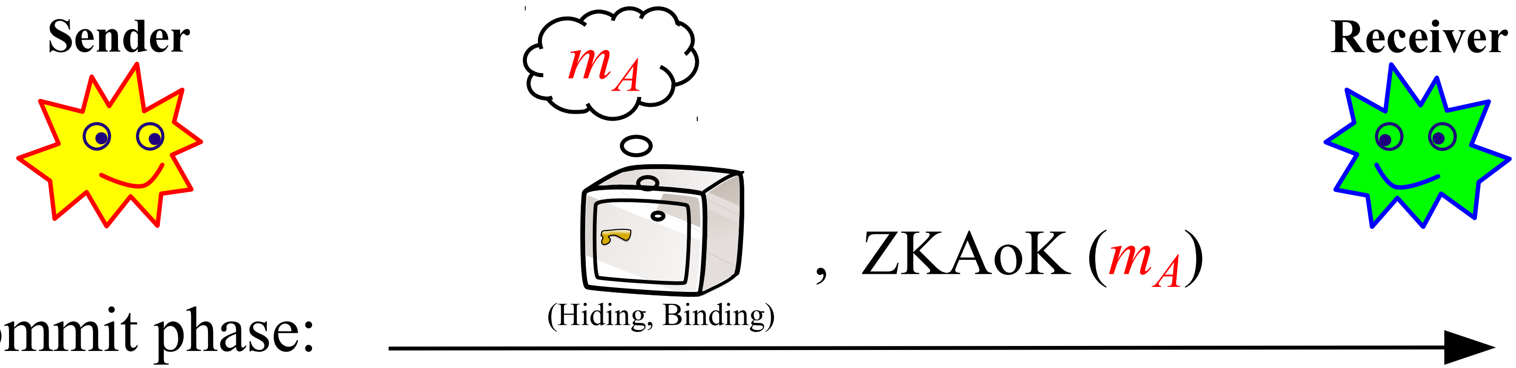
Receiver



Commit phase:

Open phase:

Example of an Ext-and-Equiv Com Scheme [Lin03]

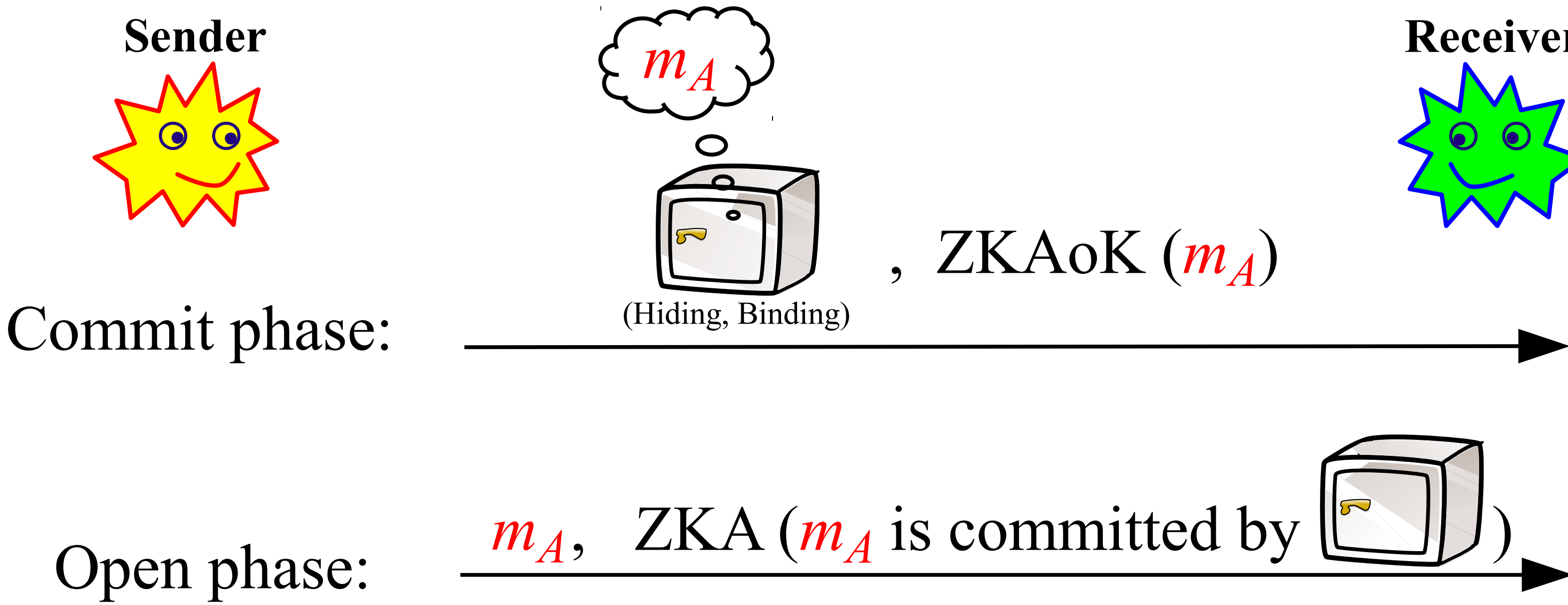


Commit phase:

Open phase:

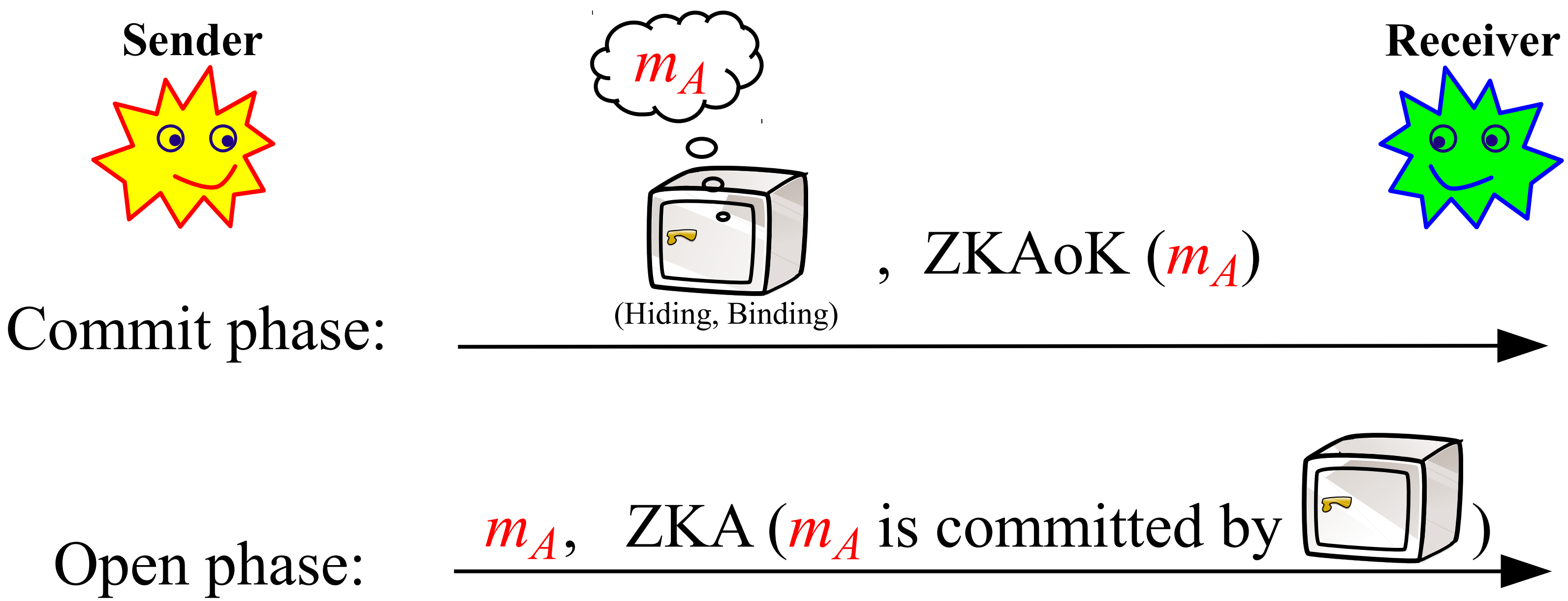
Legend:
ZKA = Zero-Knowledge Argument
ZKAoK = ZKA of knowledge

Example of an Ext-and-Equiv Com Scheme [Lin03]



Legend:
ZKA = Zero-Knowledge Argument
ZKAoK = ZKA of knowledge

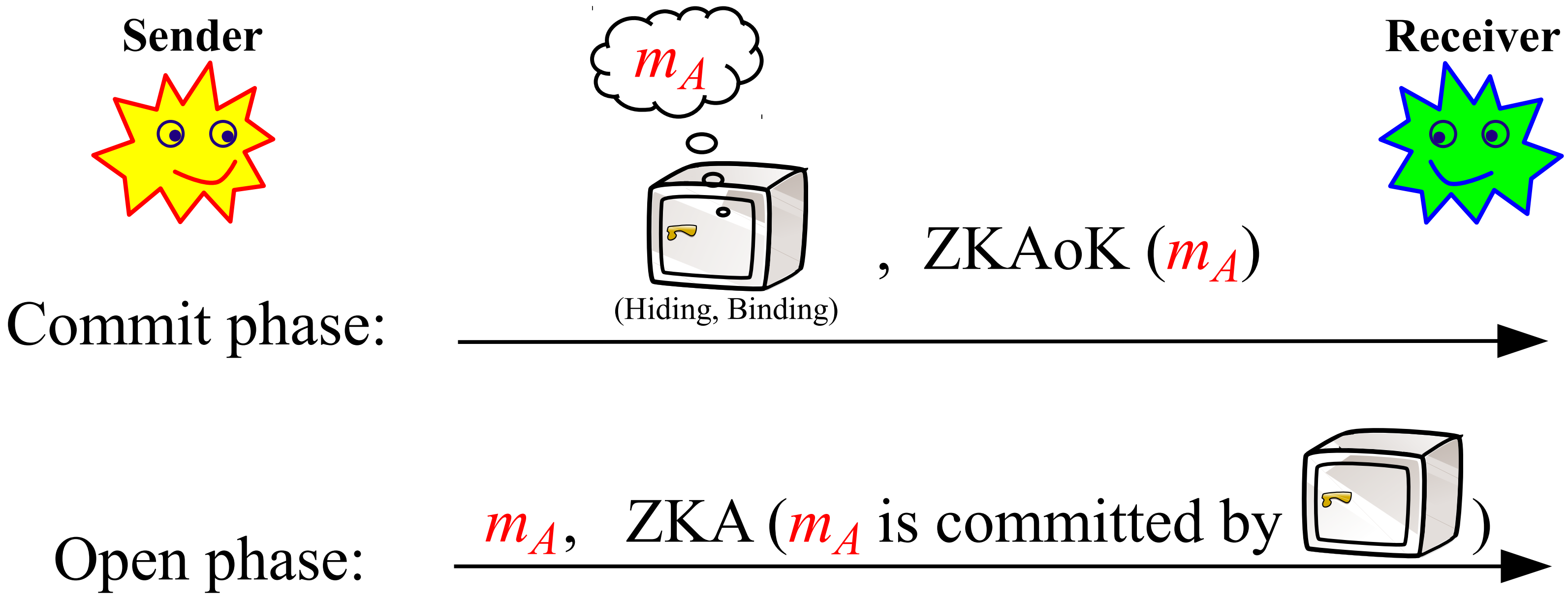
Example of an Ext-and-Equiv Com Scheme [Lin03]



Another example: [PW09] achieve Ext&Equiv via cut-and-choose methods.

Legend:
ZKA = Zero-Knowledge Argument
ZKAoK = ZKA of knowledge

Example of an Ext-and-Equiv Com Scheme [Lin03]

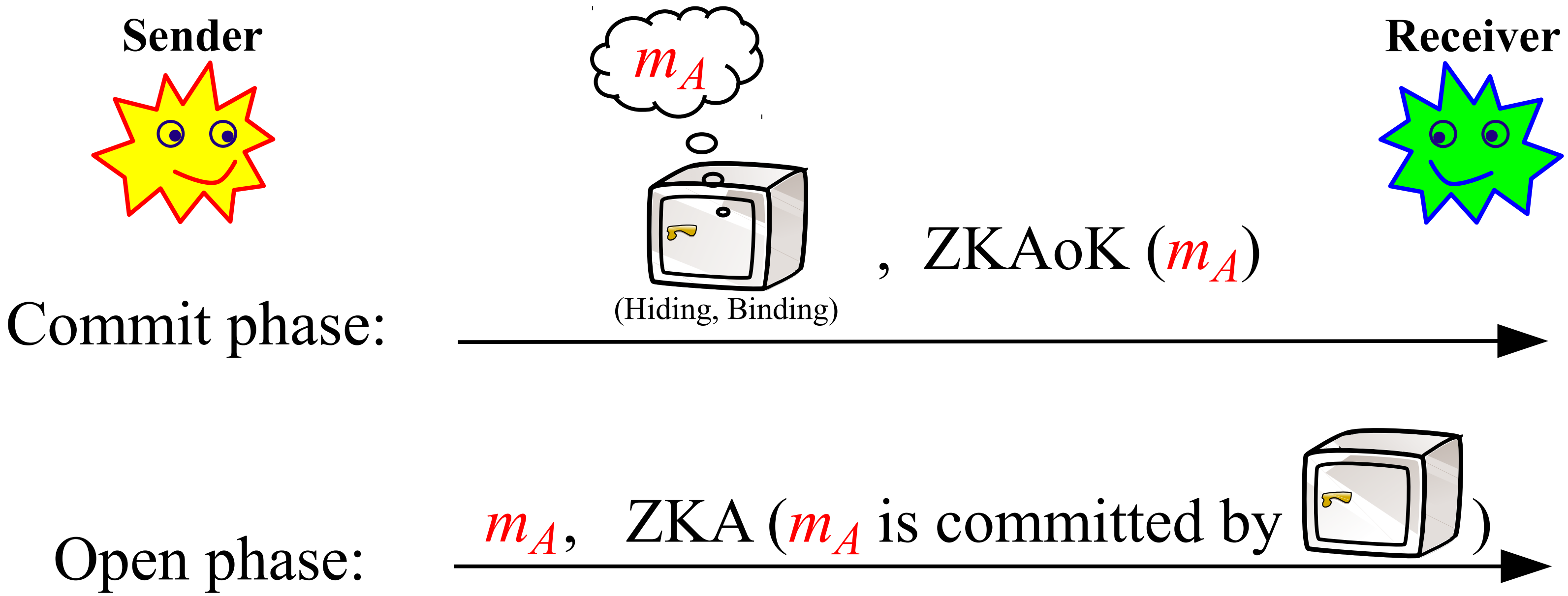


Another example: [PW09] achieve Ext&Equiv via cut-and-choose methods.

Problem: expensive in computational and/or communication terms

Legend:
ZKA = Zero-Knowledge Argument
ZKAoK = ZKA of knowledge

Example of an Ext-and-Equiv Com Scheme [Lin03]



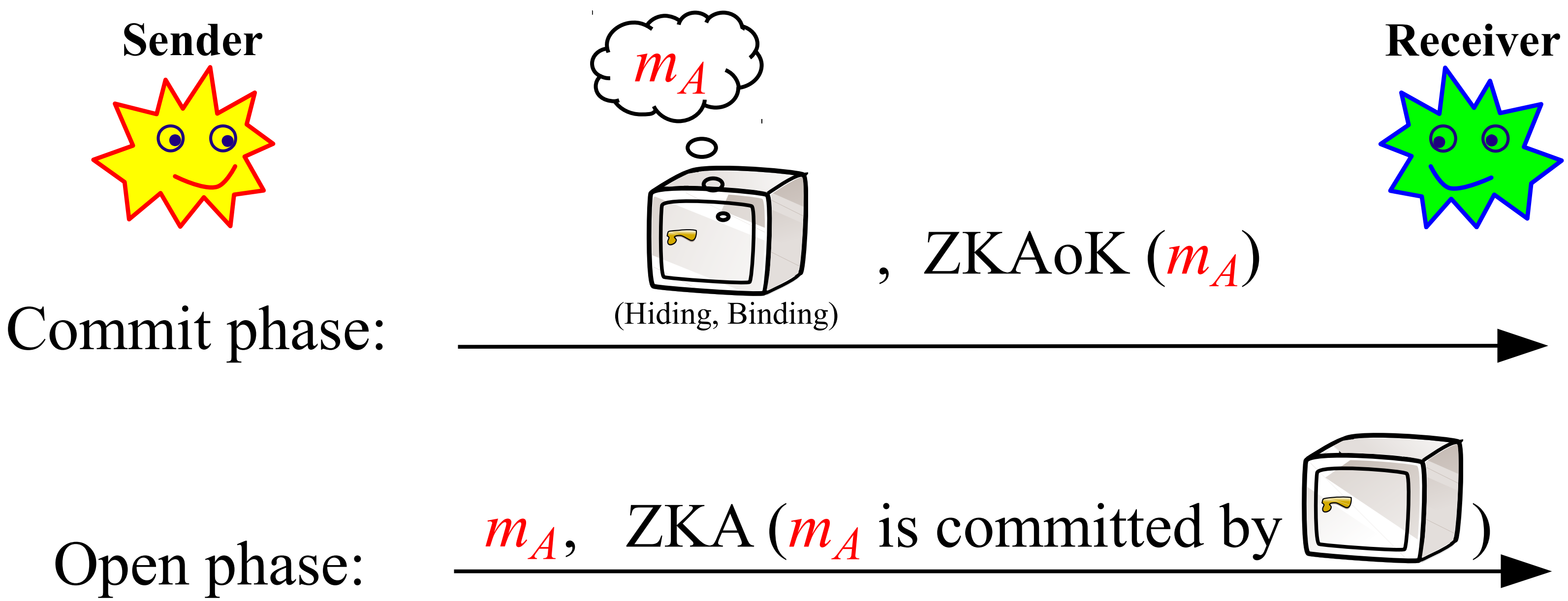
Another example: [PW09] achieve Ext&Equiv via cut-and-choose methods.

Problem: expensive in computational and/or communication terms

Can we make it more efficient?

Legend:
ZKA = Zero-Knowledge Argument
ZKAoK = ZKA of knowledge

Example of an Ext-and-Equiv Com Scheme [Lin03]



Another example: [PW09] achieve Ext&Equiv via cut-and-choose methods.

Problem: expensive in computational and/or communication terms

Can we make it more efficient?

Note: [Lin03] actually uses this construction in the scope of a more general coin-flipping into a well, where P_A only learns $f(m_A \oplus m_B)$.

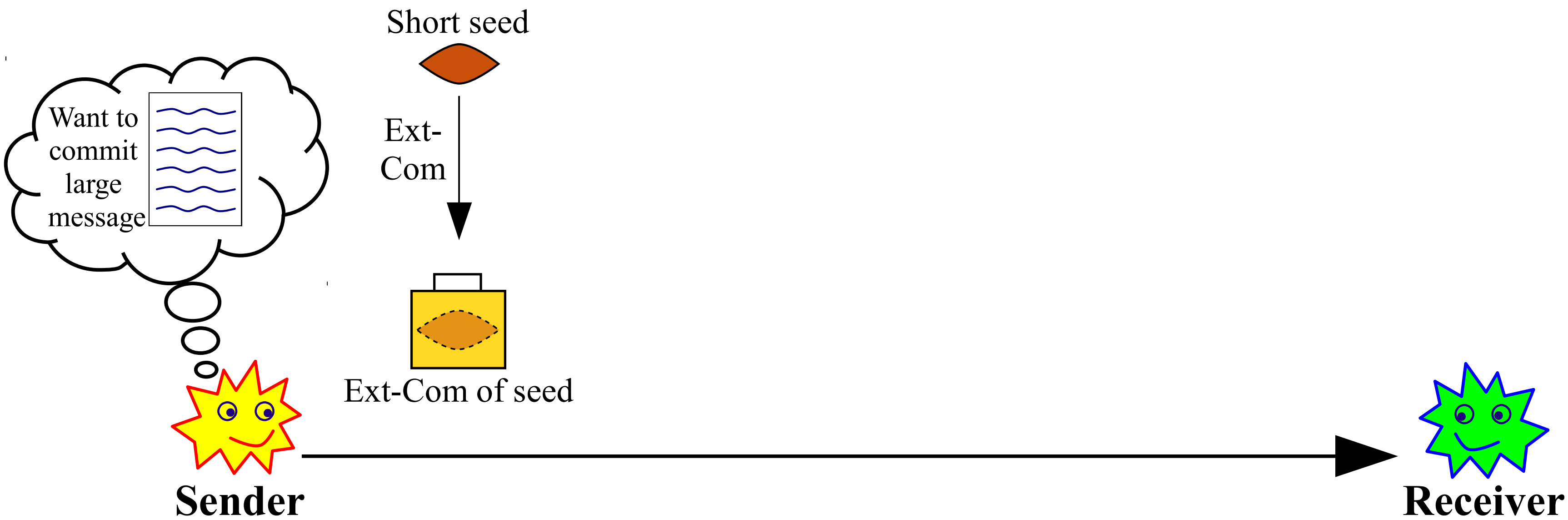
Legend:
ZKA = Zero-Knowledge Argument
ZKAoK = ZKA of knowledge

Initial intuition (insufficient)

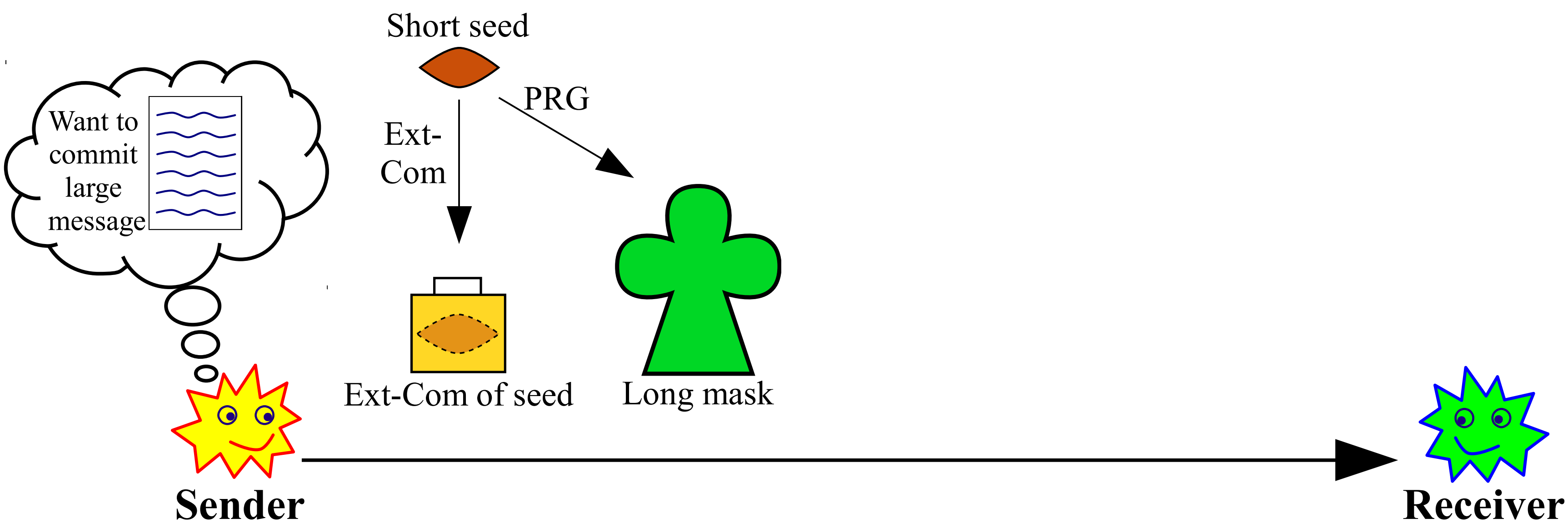
Initial intuition (insufficient)

Initial intuition (insufficient)

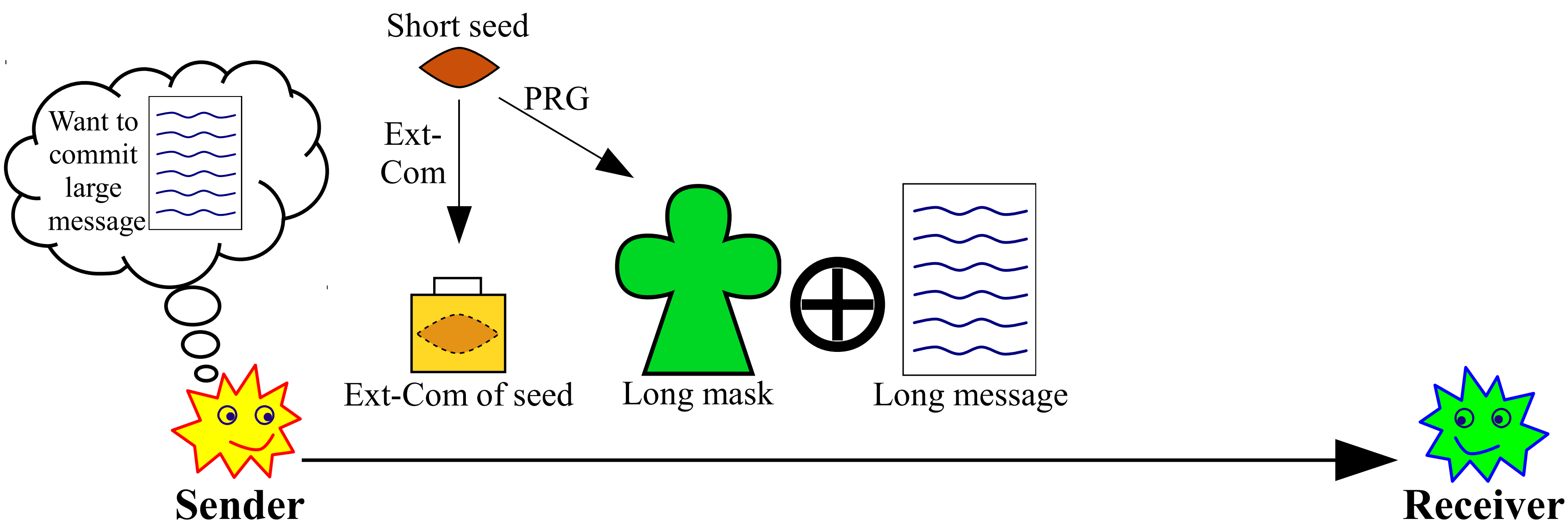
Initial intuition (insufficient)



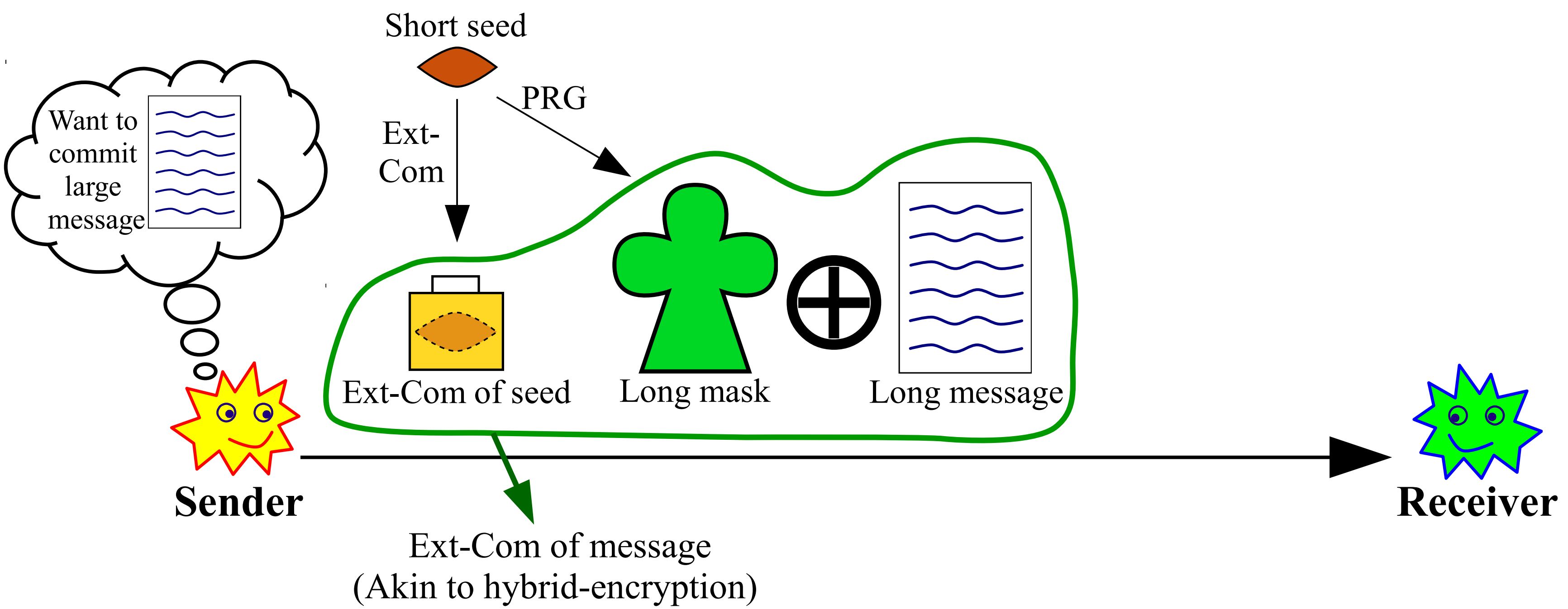
Initial intuition (insufficient)



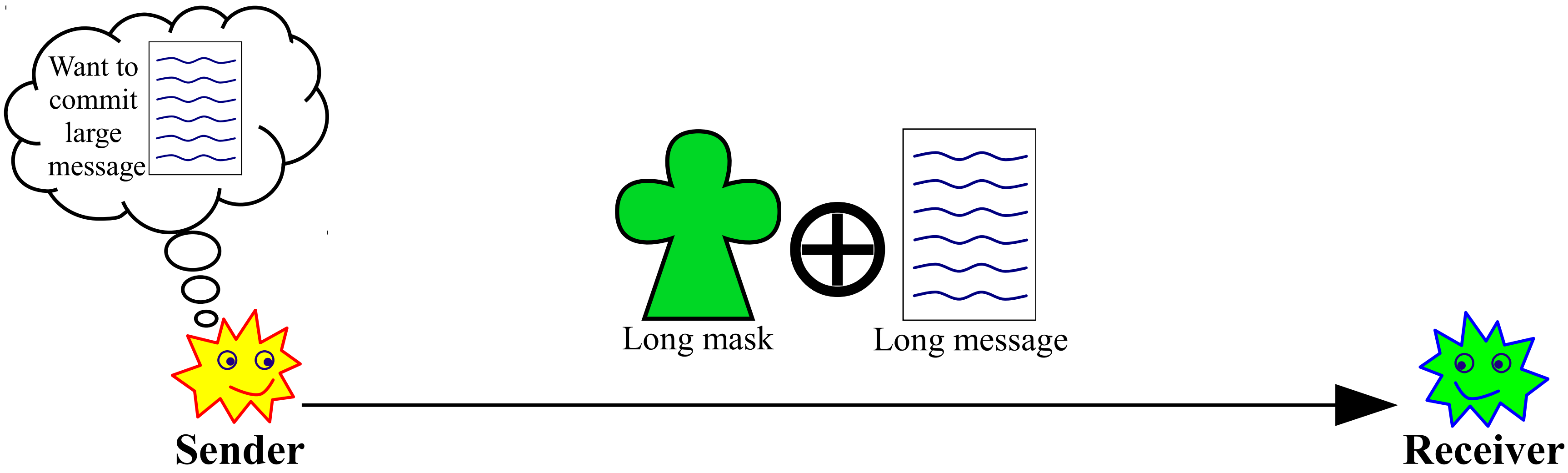
Initial intuition (insufficient)



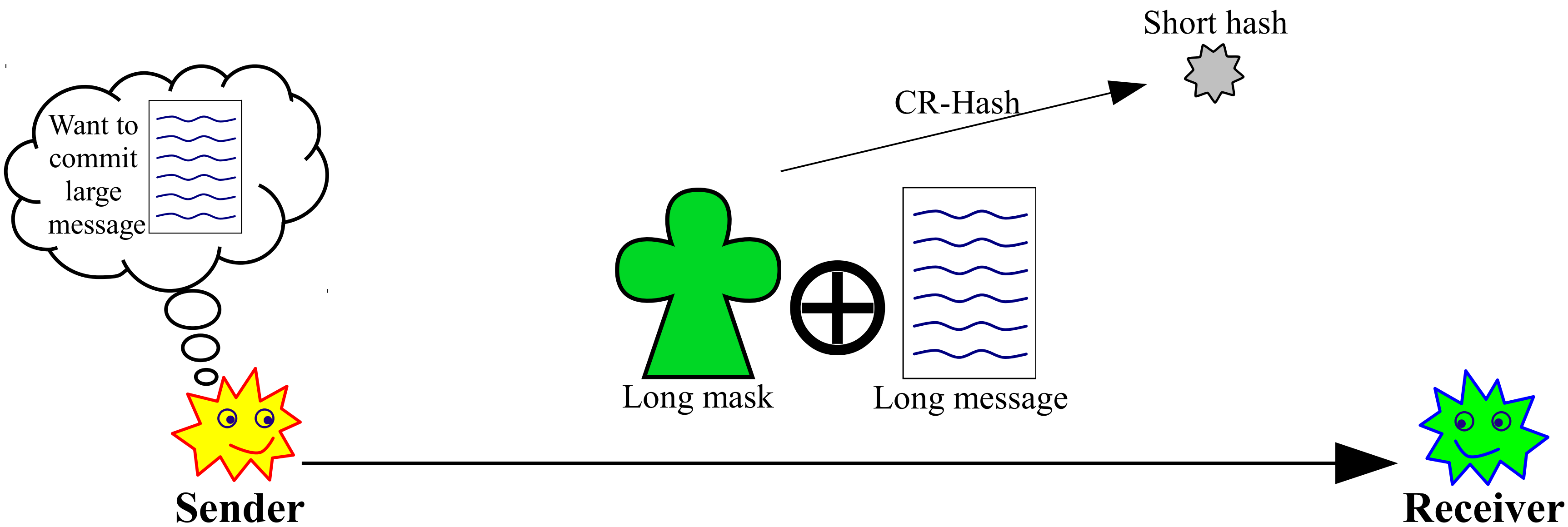
Initial intuition (insufficient)



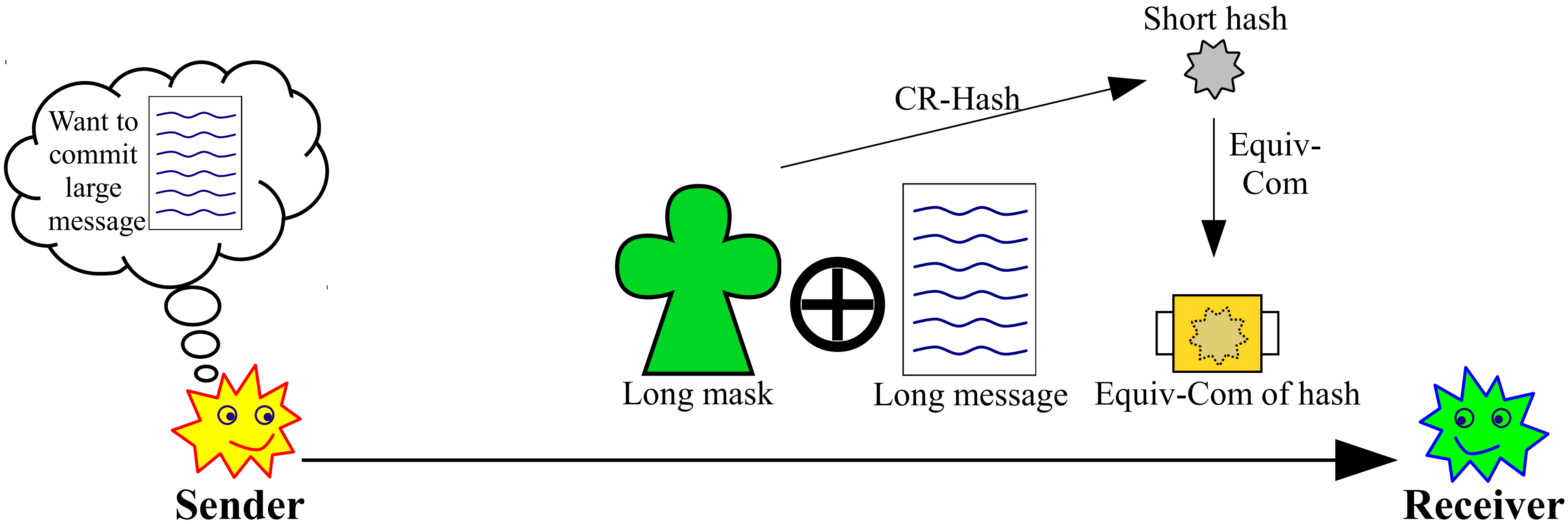
Initial intuition (insufficient)



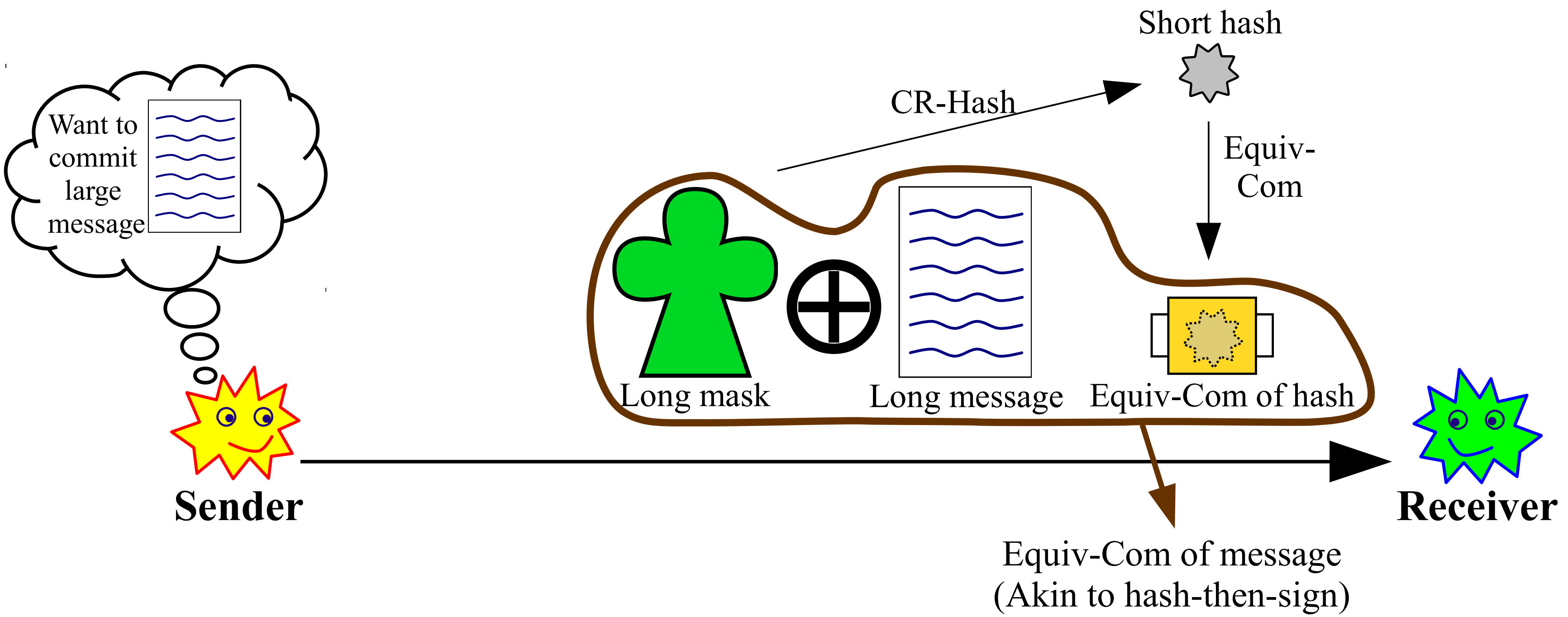
Initial intuition (insufficient)



Initial intuition (insufficient)

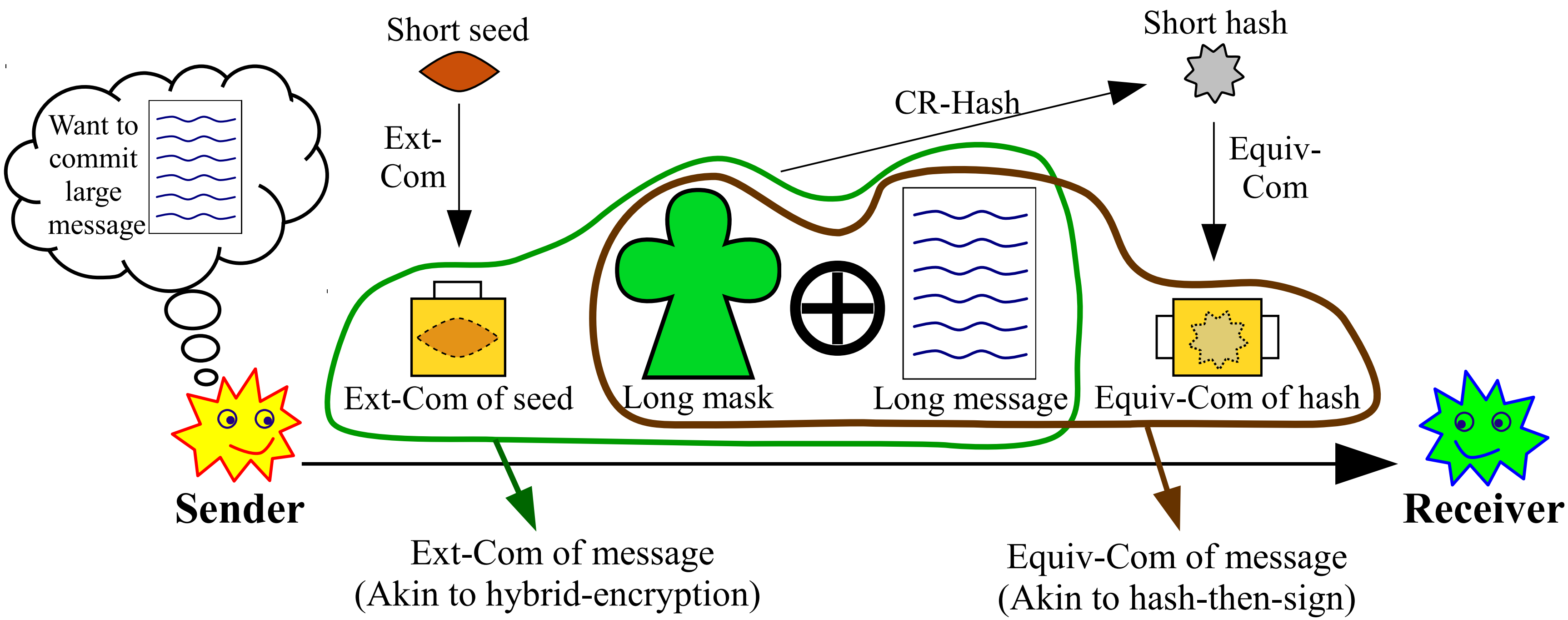


Initial intuition (insufficient)

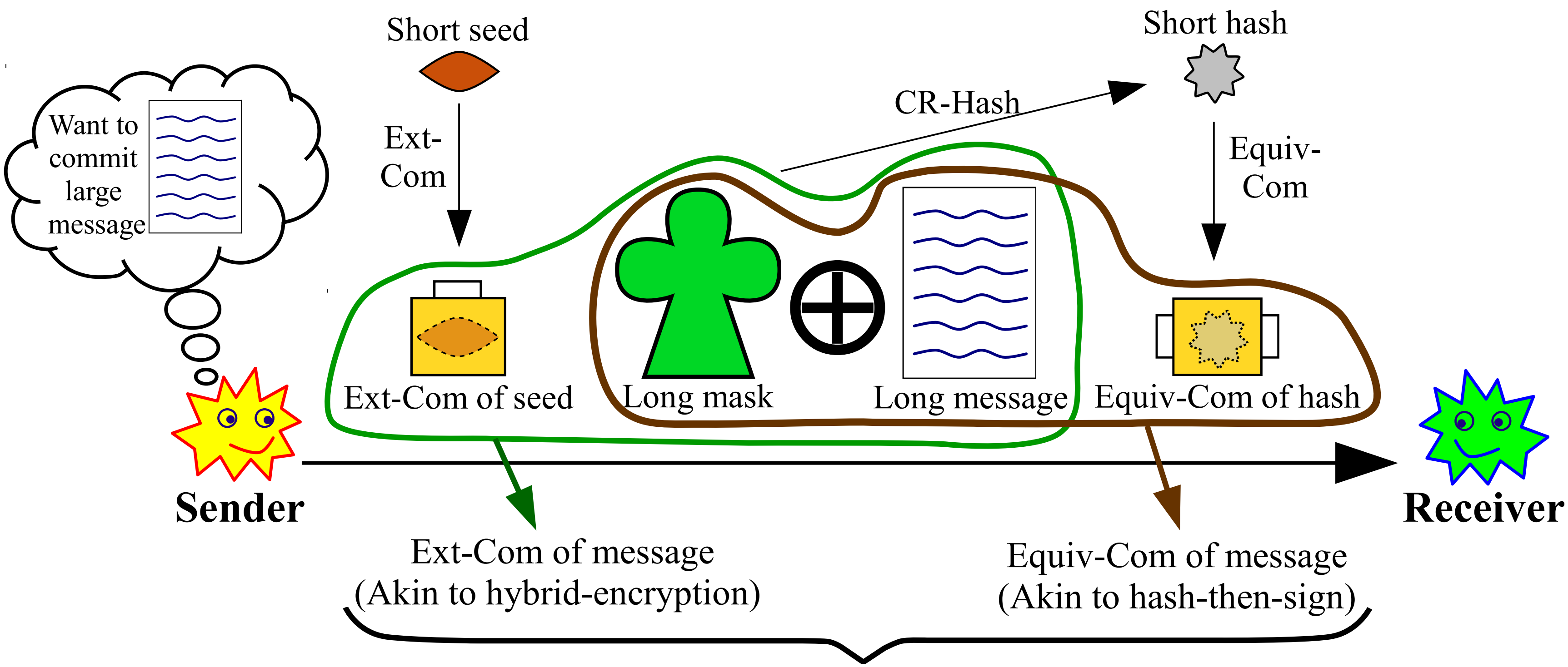


Part 1
Ideal CF
TradTemp
Ext-Equiv
Intuition

Initial intuition (insufficient)



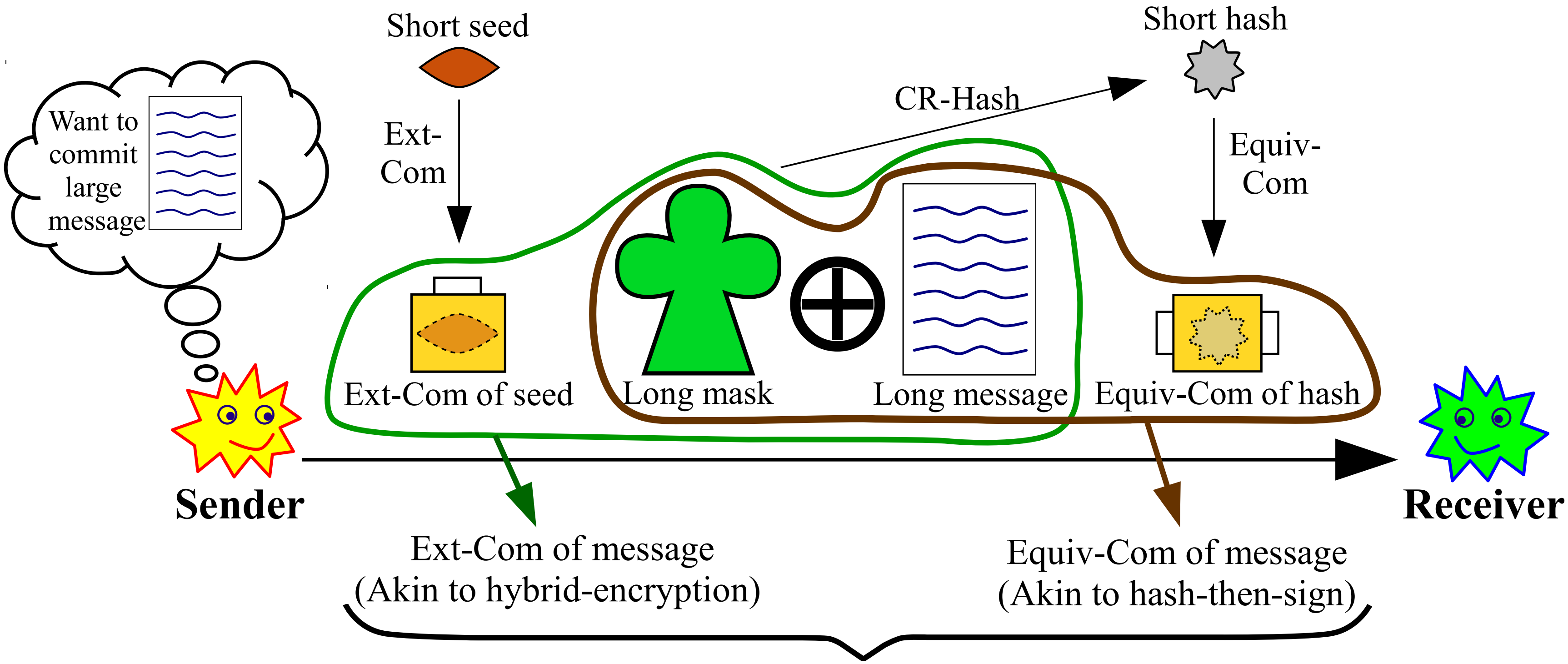
Initial intuition (insufficient)



NOT an Ext-&-Equiv Com of the message:

- Opening the **Ext-Com** of seed does not allow **equivocation**
- Removing the **Ext-Com** of seed would not allow **extraction**

Initial intuition (insufficient)

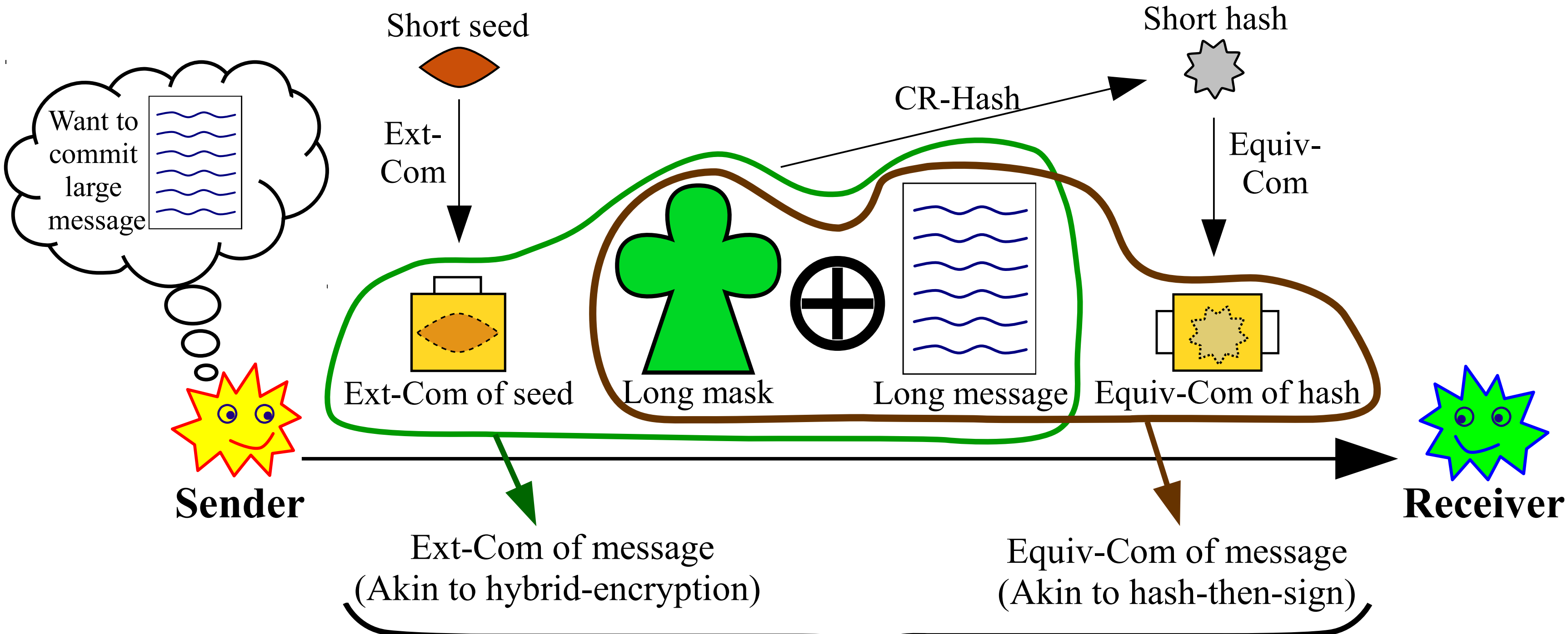


NOT an Ext-&-Equiv Com of the message:

- Opening the **Ext-Com** of seed does not allow **equivocation**
- Removing the **Ext-Com** of seed would not allow **extraction**

This presentation – how to efficiently combine Ext and Equiv (for many bits)?

Initial intuition (insufficient)



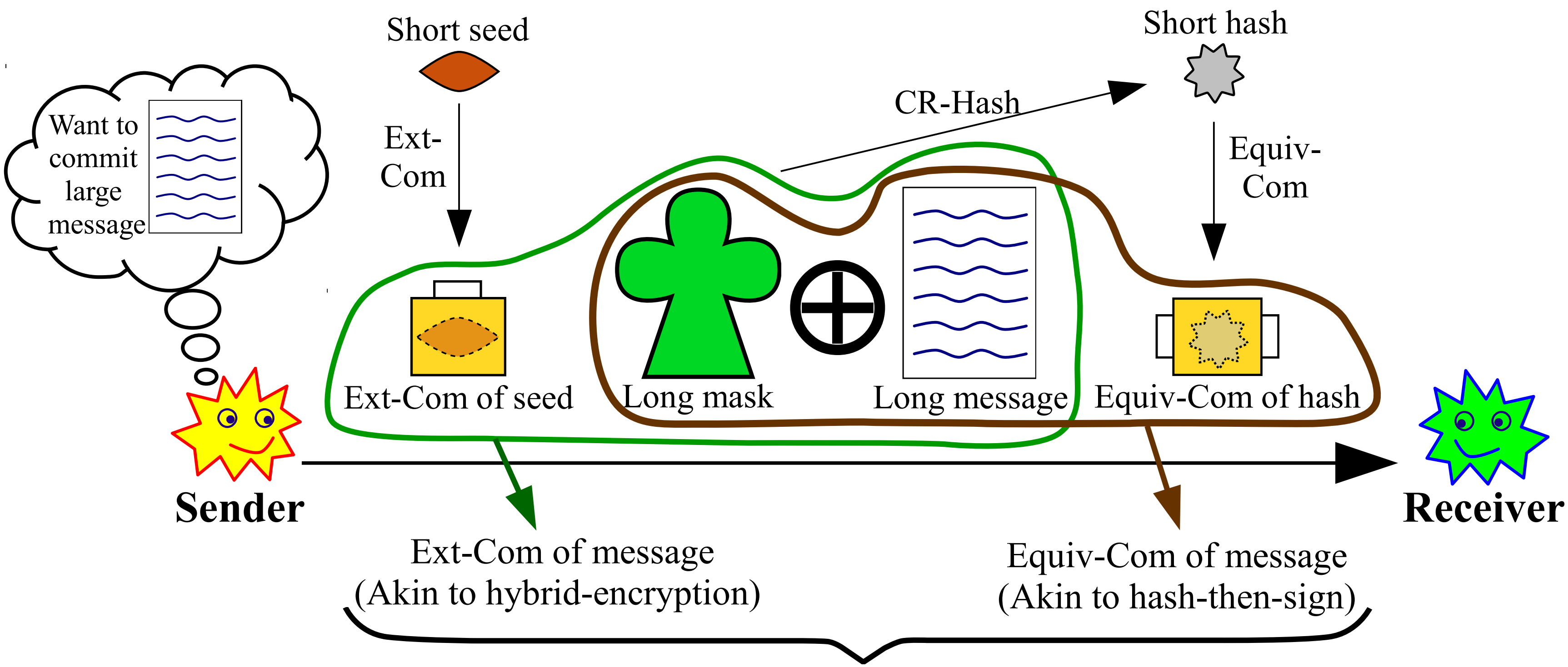
NOT an Ext-&-Equiv Com of the message:

- Opening the **Ext-Com** of seed does not allow **equivocation**
- Removing the **Ext-Com** of seed would not allow **extraction**

This presentation – how to efficiently combine Ext and Equiv (for many bits)?

- Prot #1: **Coin-flipping** simulatable-with-rewinding

Initial intuition (insufficient)



NOT an Ext-&-Equiv Com of the message:

- Opening the **Ext-Com** of seed does not allow **equivocation**
- Removing the **Ext-Com** of seed would not allow **extraction**

This presentation – how to efficiently combine Ext and Equiv (for many bits)?

- Prot #1: **Coin-flipping** simulatable-with-rewinding
- Prot #2: **UC-Com scheme** (namely without rewinding)

Roadmap

1. Simulatable coin-flipping and commitments

2. Protocol #1: coin-flipping (simulatable with rewinding)

3. Protocol #2: UC Commitment Scheme

4. Open questions / research directions

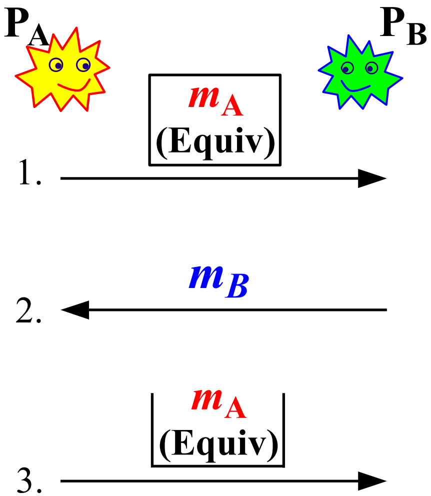
Different constructions (high level)

Different constructions (high level)

[Blum81-83] **[Lin03], [PW09]** **This paper**

Different constructions (high level)

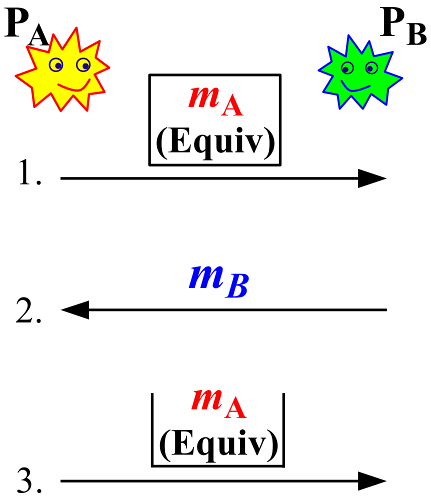
[Blum81-83] | **[Lin03], [PW09]** | **This paper**



$$m = m_A \oplus m_B$$

Different constructions (high level)

[Blum81-83] | **[Lin03], [PW09]** | **This paper**



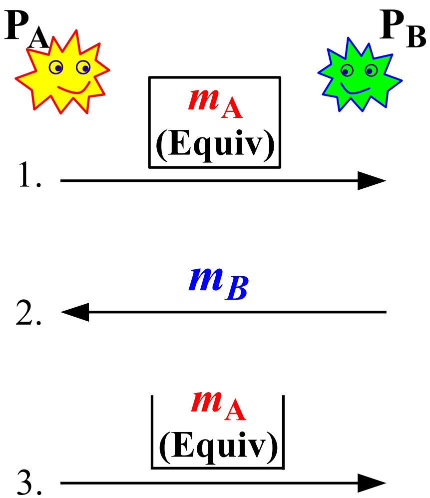
$$m = m_A \oplus m_B$$

Problem: Can't ensure ≈ Prob(⊥) in ideal vs. real world. In step 3, P_A-Prob(⊥) before Sim_B RW may (pathologically) differ from P_A-Prob(⊥) after RW.

Legend: RW = rewind; Prob(⊥) = probability of abort.

Different constructions (high level)

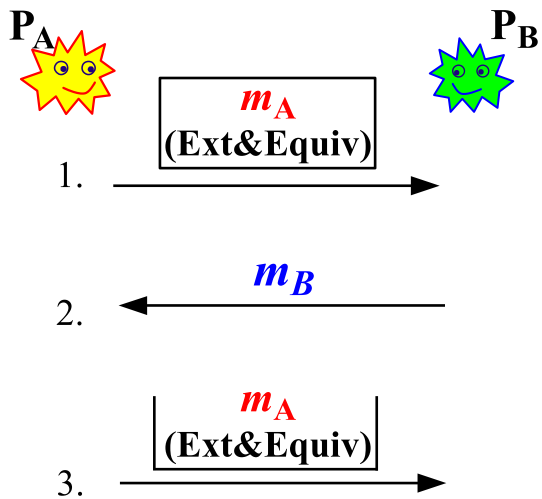
[Blum81-83]



$$m = m_A \oplus m_B$$

<u>Problem</u>: Can't ensure ≈ Prob(⊥) in ideal vs. real world. In step 3, P_A-Prob(⊥) before Sim_B RW may (pathologically) differ from P_A-Prob(⊥) after RW.

[Lin03], [PW09]



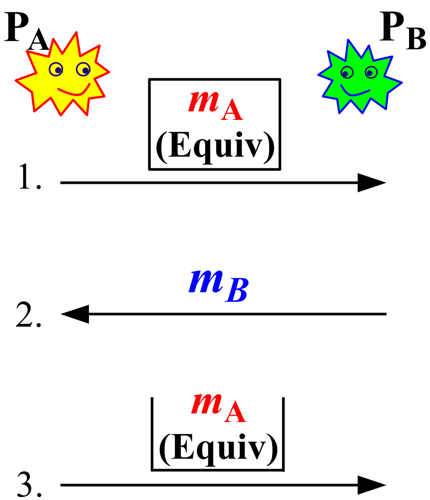
$$m = m_A \oplus m_B$$

This paper

<u>Legend</u>: RW = rewind; Prob(⊥) = probability of abort.

Different constructions (high level)

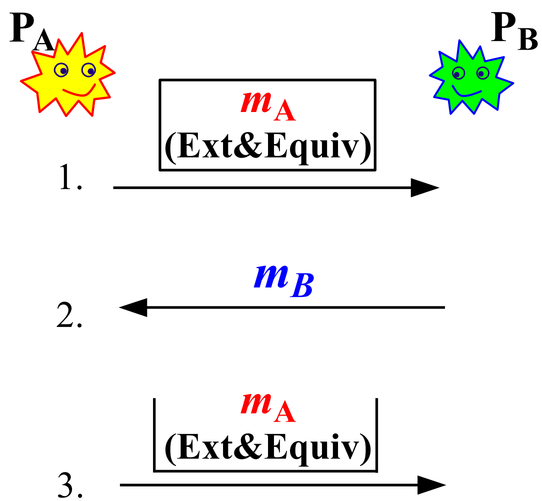
[Blum81-83]



$$m = m_A \oplus m_B$$

Problem: Can't ensure ≈ Prob(⊥) in ideal vs. real world. In step 3, P_A-Prob(⊥) before Sim_B RW may (pathologically) differ from P_A-Prob(⊥) after RW.

[Lin03], [PW09]



$$m = m_A \oplus m_B$$

- Lin03: ZK-based
- PW09: Cut&Choose based

Simulatable, but inefficient for large $|m|$.

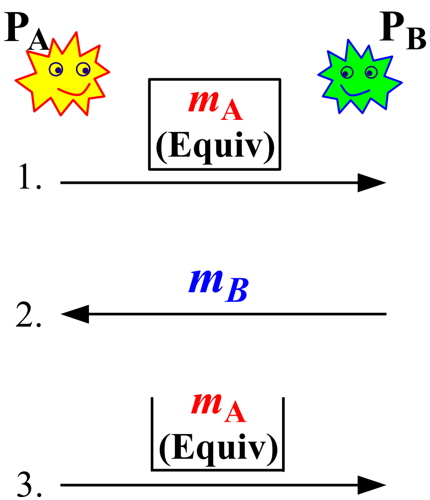
This paper

Part 2
Compare
Analyze
Complex

Legend: RW = rewind; Prob(⊥) = probability of abort.

Different constructions (high level)

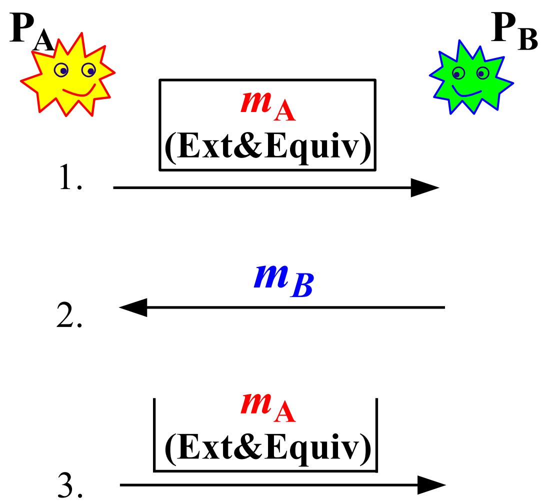
[Blum81-83]



$$m = m_A \oplus m_B$$

Problem: Can't ensure ≈ Prob(⊥) in ideal vs. real world. In step 3, P_A-Prob(⊥) before Sim_B RW may (pathologically) differ from P_A-Prob(⊥) after RW.

[Lin03], [PW09]

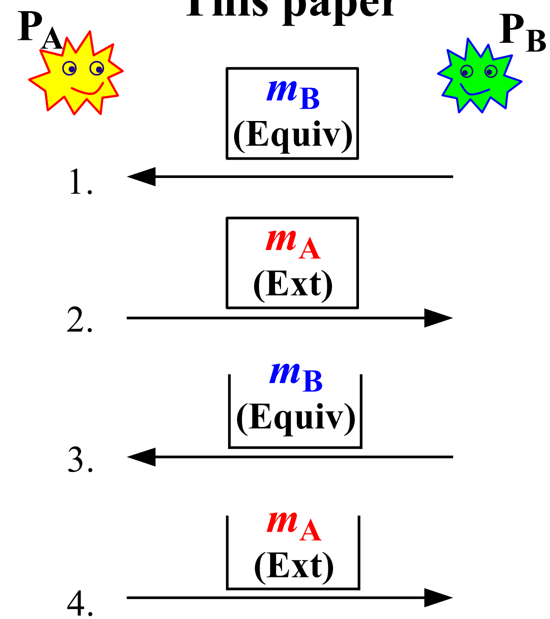


$$m = m_A \oplus m_B$$

- Lin03: ZK-based
- PW09: Cut&Choose based

Simulatable, but inefficient for large $|m|$.

This paper

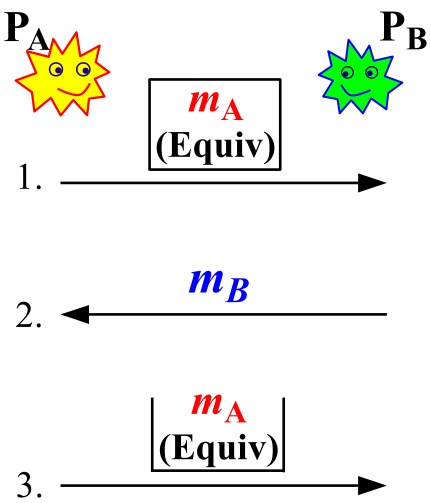


$$m = m_A \oplus m_B$$

Legend: RW = rewind; Prob(⊥) = probability of abort.

Different constructions (high level)

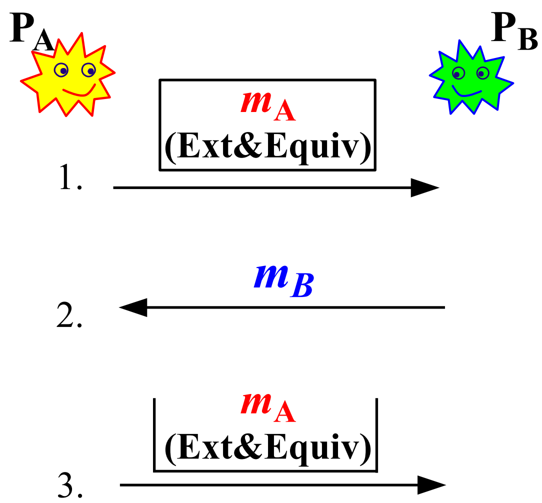
[Blum81-83]



$$m = m_A \oplus m_B$$

Problem: Can't ensure ≈ Prob(⊥) in ideal vs. real world. In step 3, P_A-Prob(⊥) before Sim_B RW may (pathologically) differ from P_A-Prob(⊥) after RW.

[Lin03], [PW09]

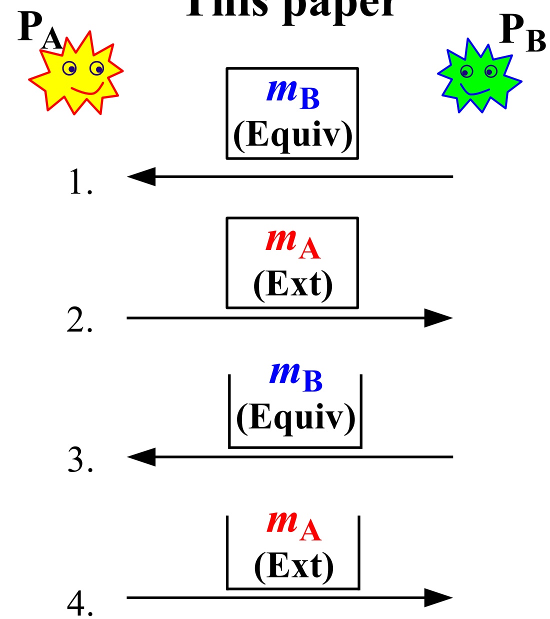


$$m = m_A \oplus m_B$$

- Lin03: ZK-based
- PW09: Cut&Choose based

Simulatable, but inefficient for large $|m|$.

This paper



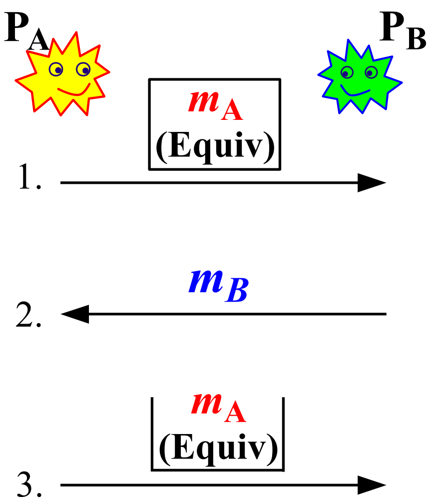
$$m = m_A \oplus m_B$$

Ext-Com and Equiv-Com are efficient

Legend: RW = rewind; Prob(⊥) = probability of abort.

Different constructions (high level)

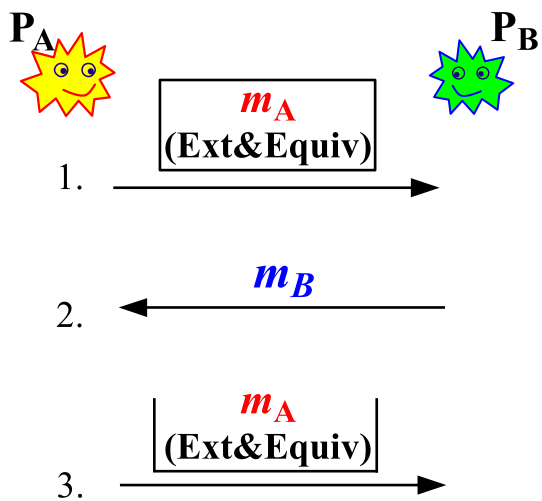
[Blum81-83]



$$m = m_A \oplus m_B$$

Problem: Can't ensure ≈ Prob(⊥) in ideal vs. real world. In step 3, P_A-Prob(⊥) before Sim_B RW may (pathologically) differ from P_A-Prob(⊥) after RW.

[Lin03], [PW09]

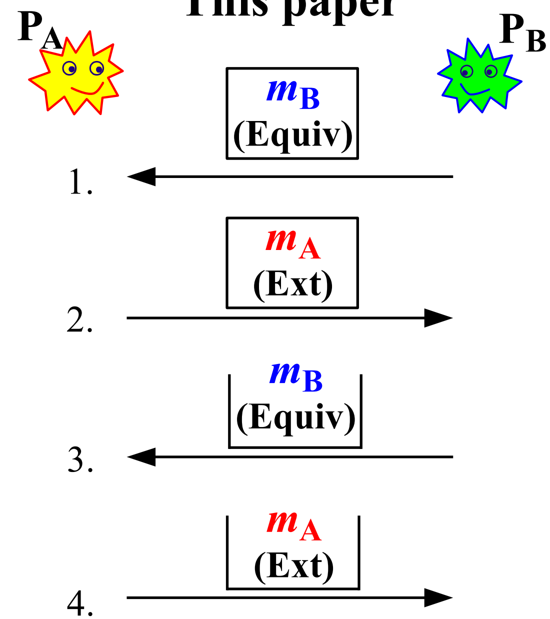


$$m = m_A \oplus m_B$$

- Lin03: ZK-based
- PW09: Cut&Choose based

Simulatable, but inefficient for large $|m|$.

This paper



$$m = m_A \oplus m_B$$

Ext-Com and Equiv-Com are efficient

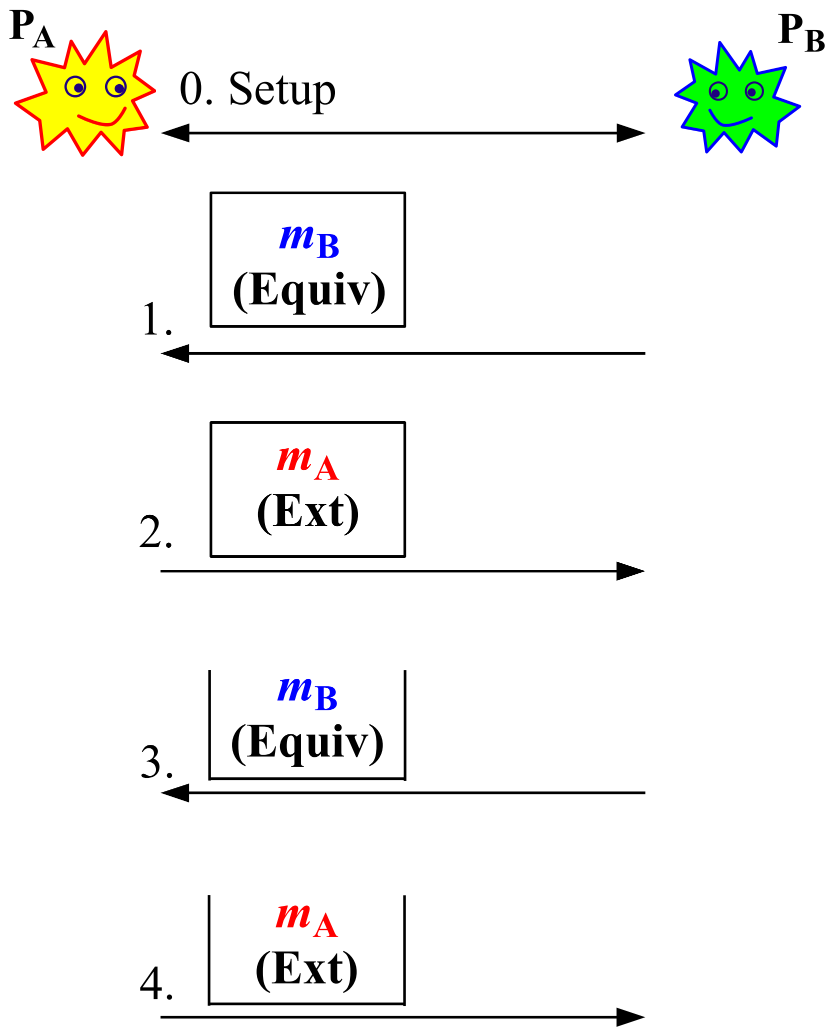
Simulatability: In the difficult side, Prob(⊥) by P_B (step 3) may depend on Com(m_A), but not on clear m_A. Can be simulated in Expected-Poly # RWs.

Legend: RW = rewind; Prob(⊥) = probability of abort.

Closer look: possible instantiation and simulation (high level)

Legend: Ped (Pedersen); ElgCom (ElGamal)

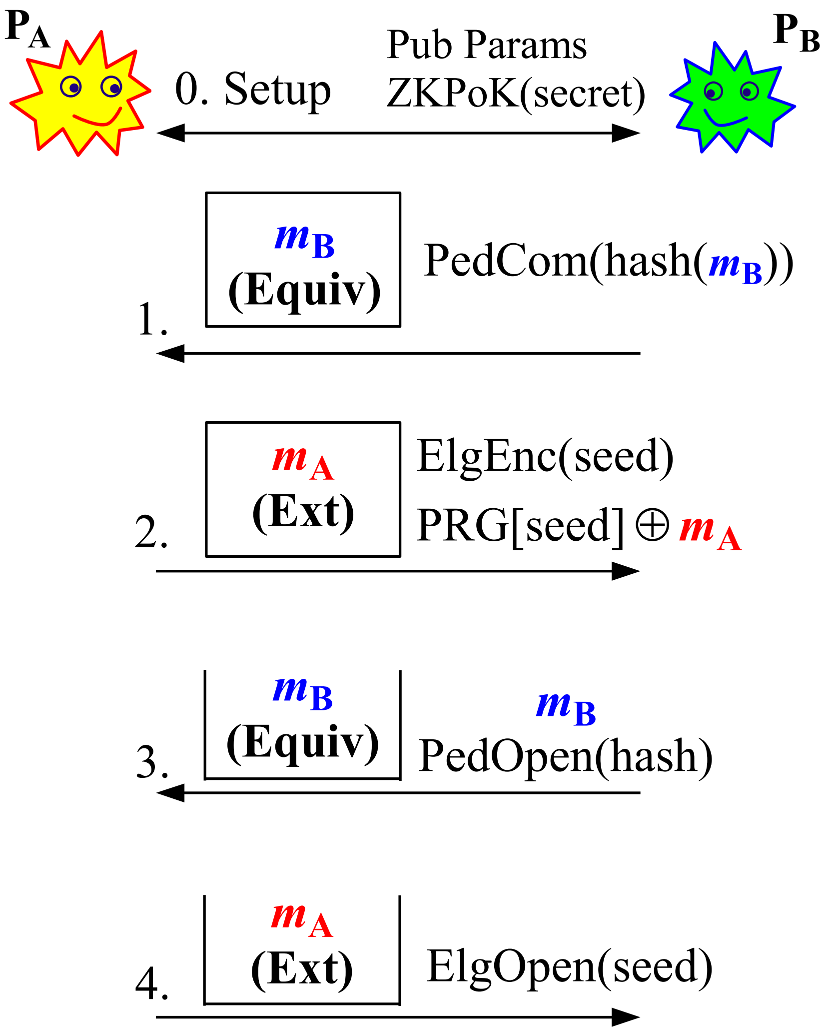
Closer look: possible instantiation and simulation (high level)



$$m = m_A \oplus m_B$$

Legend: Ped (Pedersen); ElgCom (ElGamal)

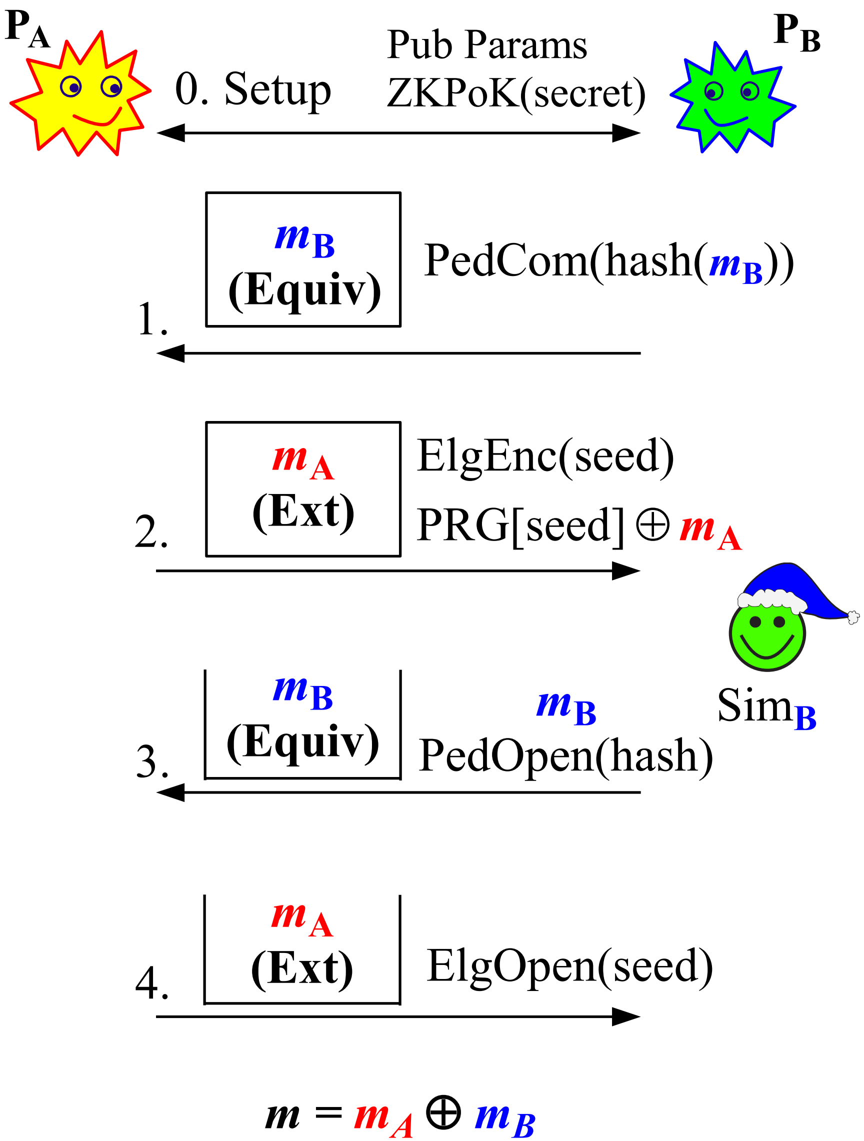
Closer look: possible instantiation and simulation (high level)



$$m = m_A \oplus m_B$$

Legend: Ped (Pedersen); ElgCom (ElGamal)

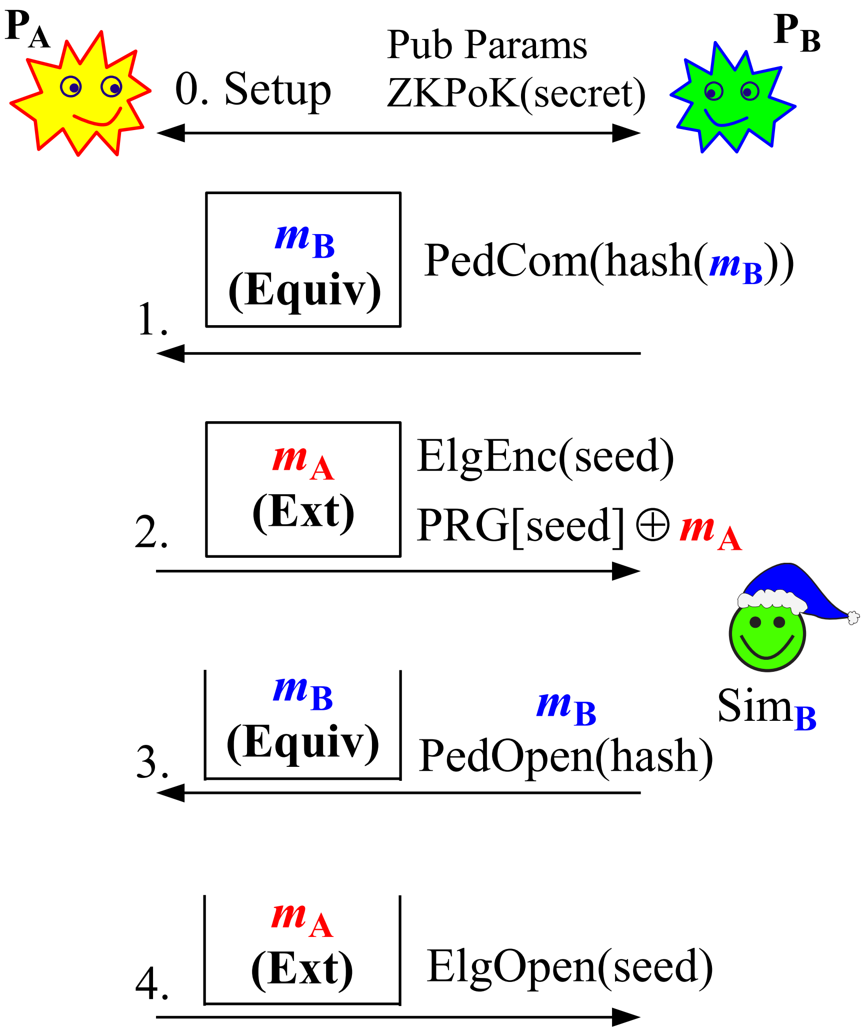
Closer look: possible instantiation and simulation (high level)



Case malicious P_A

Legend: Ped (Pedersen); ElgCom (ElGamal)

Closer look: possible instantiation and simulation (high level)



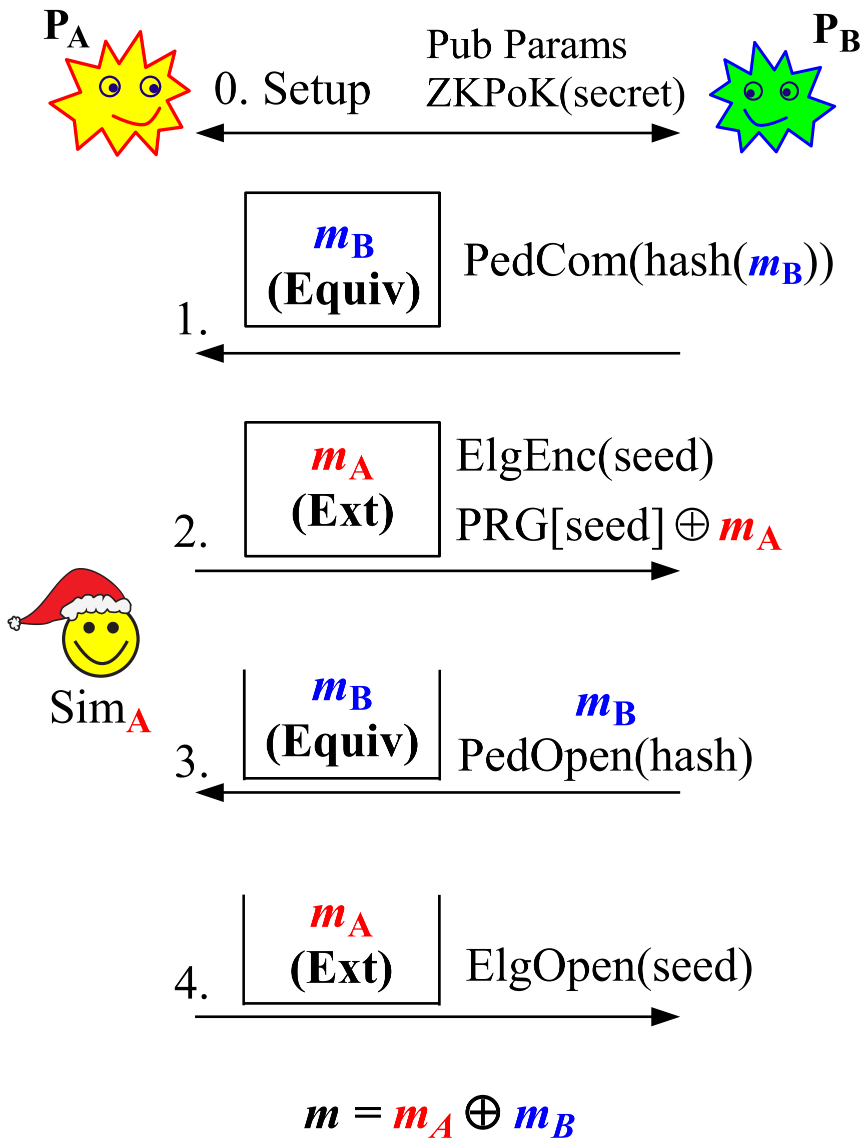
$$m = m_A \oplus m_B$$

Case malicious P_A

- In step 0: Sim_B extract trapdoor
- In step 2: Sim_B extracts m_A ,
- In step 3: Sim_B Equiv-opens $m_B = m \oplus m_A$

Legend: Ped (Pedersen); ElgCom (ElGamal)

Closer look: possible instantiation and simulation (high level)



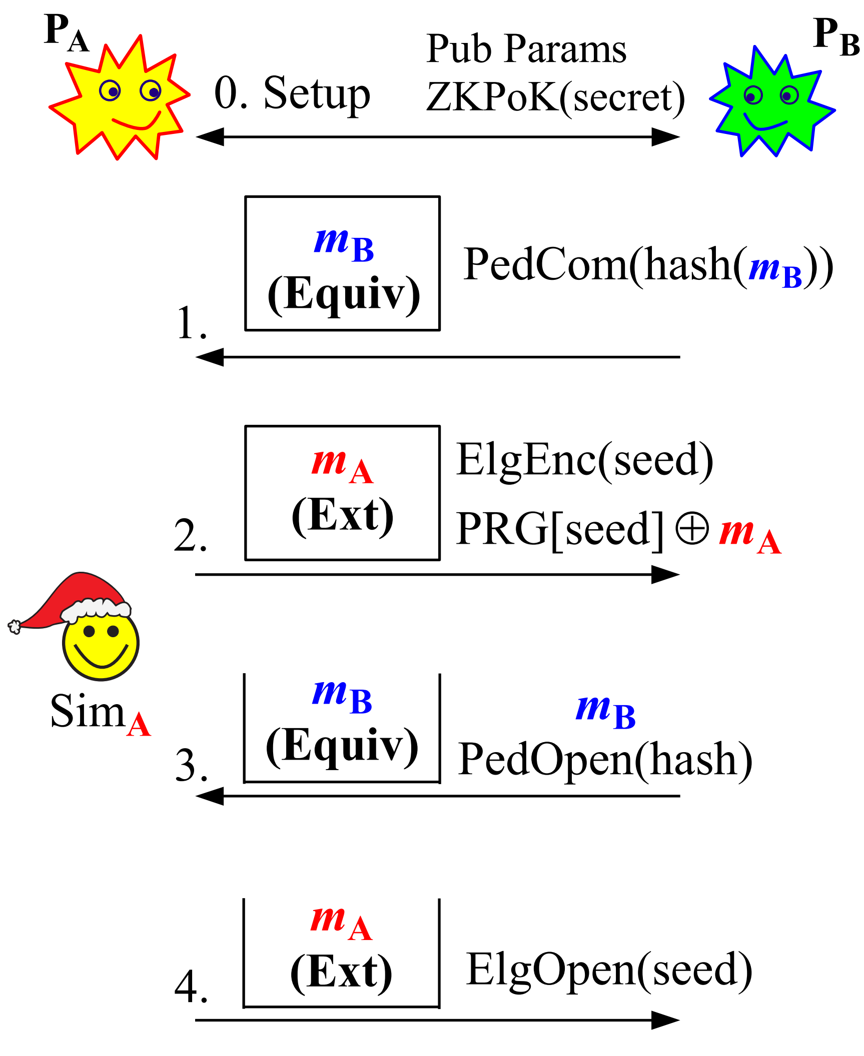
Case malicious P_A

- In step 0: Sim_B extract trapdoor
- In step 2: Sim_B extracts m_A ,
- In step 3: Sim_B Equiv-opens $m_B = m \oplus m_A$

Case malicious P_B

Legend: Ped (Pedersen); ElgCom (ElGamal)

Closer look: possible instantiation and simulation (high level)



$m = m_A \oplus m_B$

Case malicious P_A

- In step 0: Sim_B extract trapdoor
- In step 2: Sim_B extracts m_A ,
- In step 3: Sim_B Equiv-opens $m_B = m \oplus m_A$

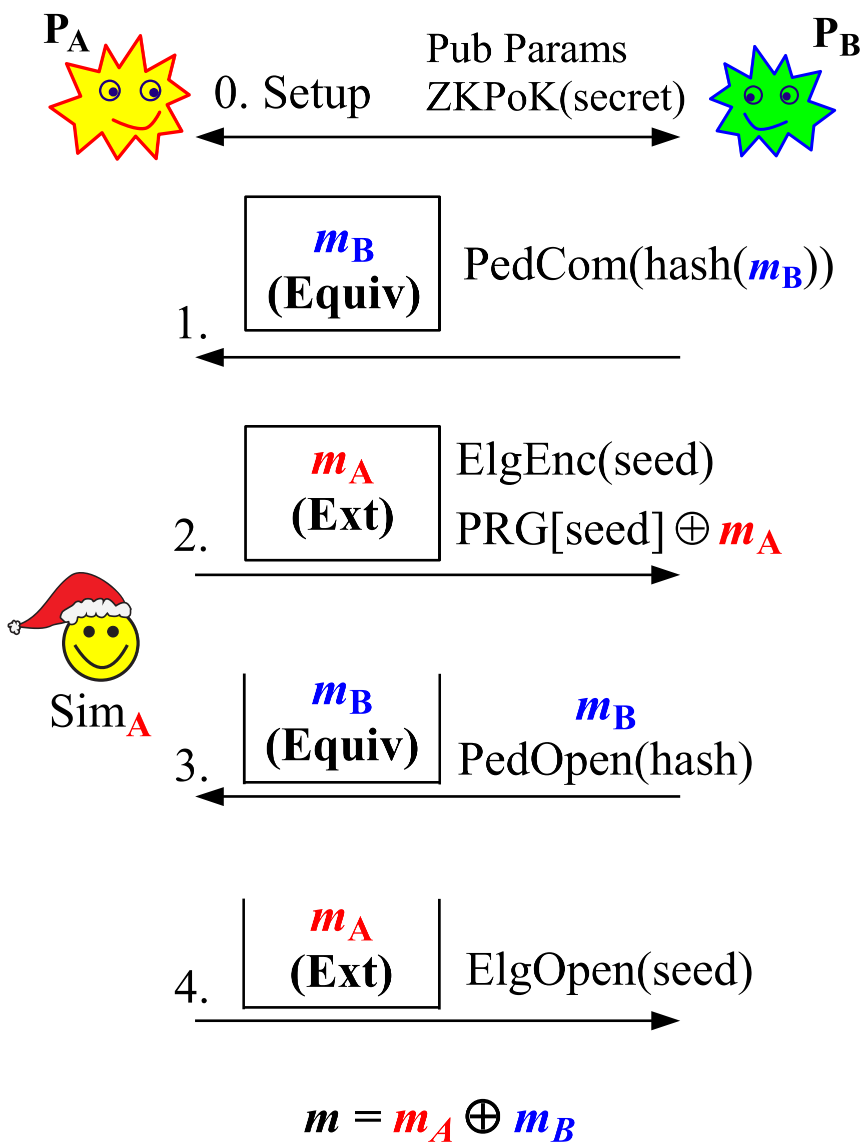
Case malicious P_B

Optimistic simulation:

- In step 2: Sim_A commits random m_A
- In step 3: P_B opens m_B, then Sim_A rewinds
- In step 2: Sim_A commits $m_A = m \oplus m_B$
- In step 3: P_B opens m_B

Legend: Ped (Pedersen); ElgCom (ElGamal)

Closer look: possible instantiation and simulation (high level)



Case malicious P_A

- In step 0: Sim_B extract trapdoor
- In step 2: Sim_B extracts m_A ,
- In step 3: Sim_B Equiv-opens $m_B = m \oplus m_A$

Case malicious P_B

Optimistic simulation:

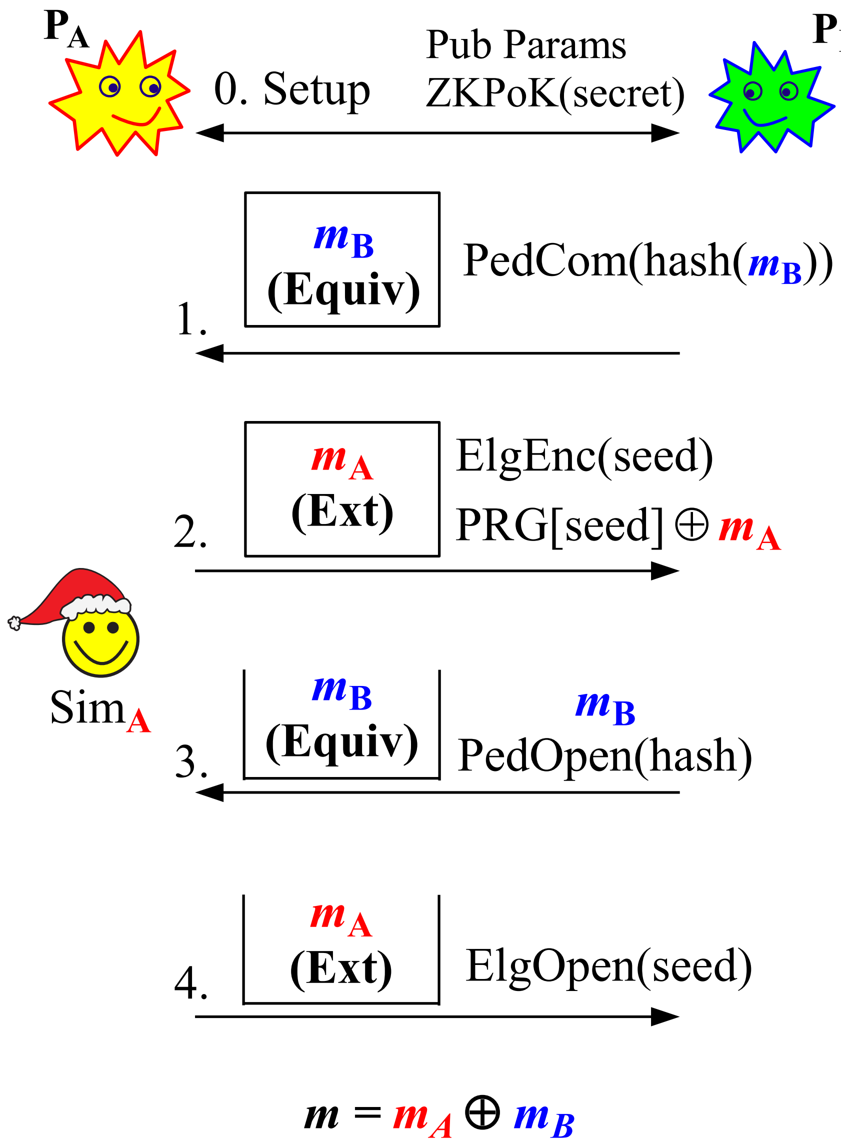
- In step 2: Sim_A commits random m_A
- In step 3: P_B opens m_B, then Sim_A rewinds
- In step 2: Sim_A commits $m_A = m \oplus m_B$
- In step 3: P_B opens m_B

If P_B aborts (⊥) first time in step 3:

- Sim_A emulates abort in ideal world.

Legend: Ped (Pedersen); ElgCom (ElGamal)

Closer look: possible instantiation and simulation (high level)



Case malicious P_A

- In step 0: Sim_B extract trapdoor
- In step 2: Sim_B extracts m_A ,
- In step 3: Sim_B Equiv-opens $m_B = m \oplus m_A$

Case malicious P_B

Optimistic simulation:

- In step 2: Sim_A commits random m_A
- In step 3: P_B opens m_B, then Sim_A rewinds
- In step 2: Sim_A commits $m_A = m \oplus m_B$
- In step 3: P_B opens m_B

If P_B aborts (⊥) first time in step 3:

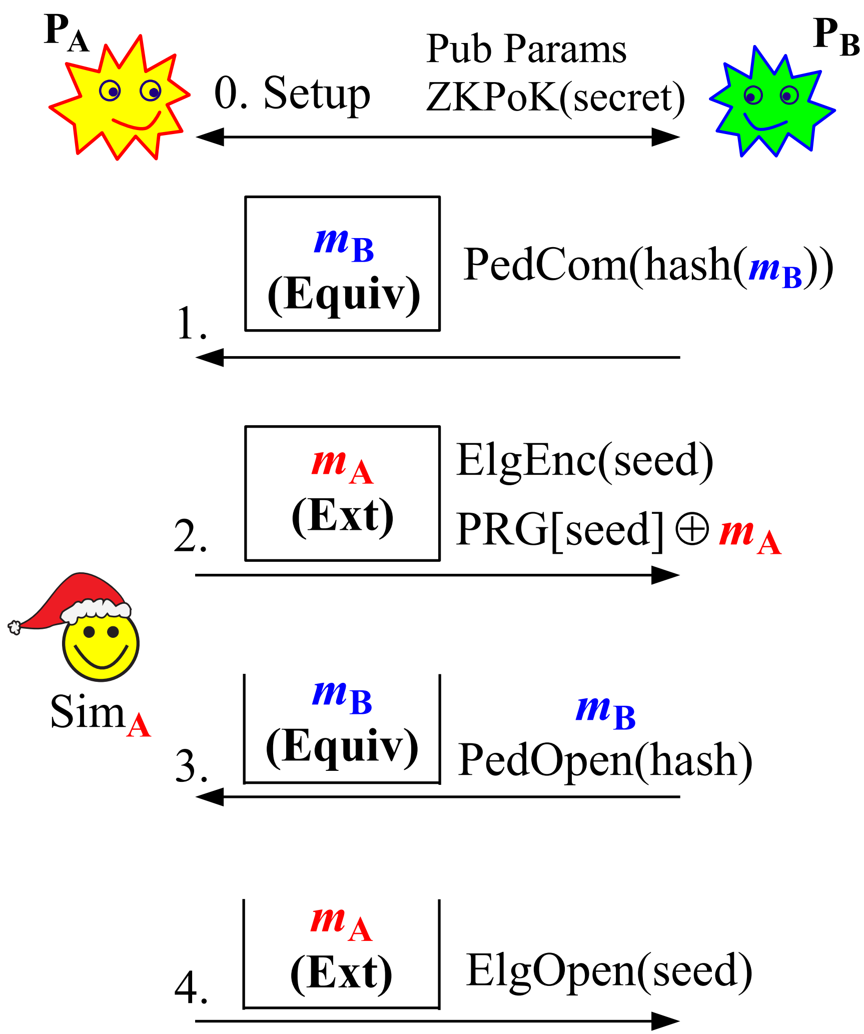
- Sim_A emulates abort in ideal world.

If P_B NOT-⊥ 1st time, but ⊥ 2nd time:

- Sim_A estimates Prob(⊥) ([GK96])
- Sim_A tries till P_B opens or #RWs ≈ $p(k)$/Prob(⊥)

Legend: Ped (Pedersen); ElgCom (ElGamal)

Closer look: possible instantiation and simulation (high level)



$m = m_A \oplus m_B$

Case malicious P_A

- In step 0: Sim_B extract trapdoor
- In step 2: Sim_B extracts m_A ,
- In step 3: Sim_B Equiv-opens $m_B = m \oplus m_A$

Case malicious P_B

Optimistic simulation:

- In step 2: Sim_A commits random m_A
- In step 3: P_B opens m_B, then Sim_A rewinds
- In step 2: Sim_A commits $m_A = m \oplus m_B$
- In step 3: P_B opens m_B

If P_B aborts (⊥) first time in step 3:

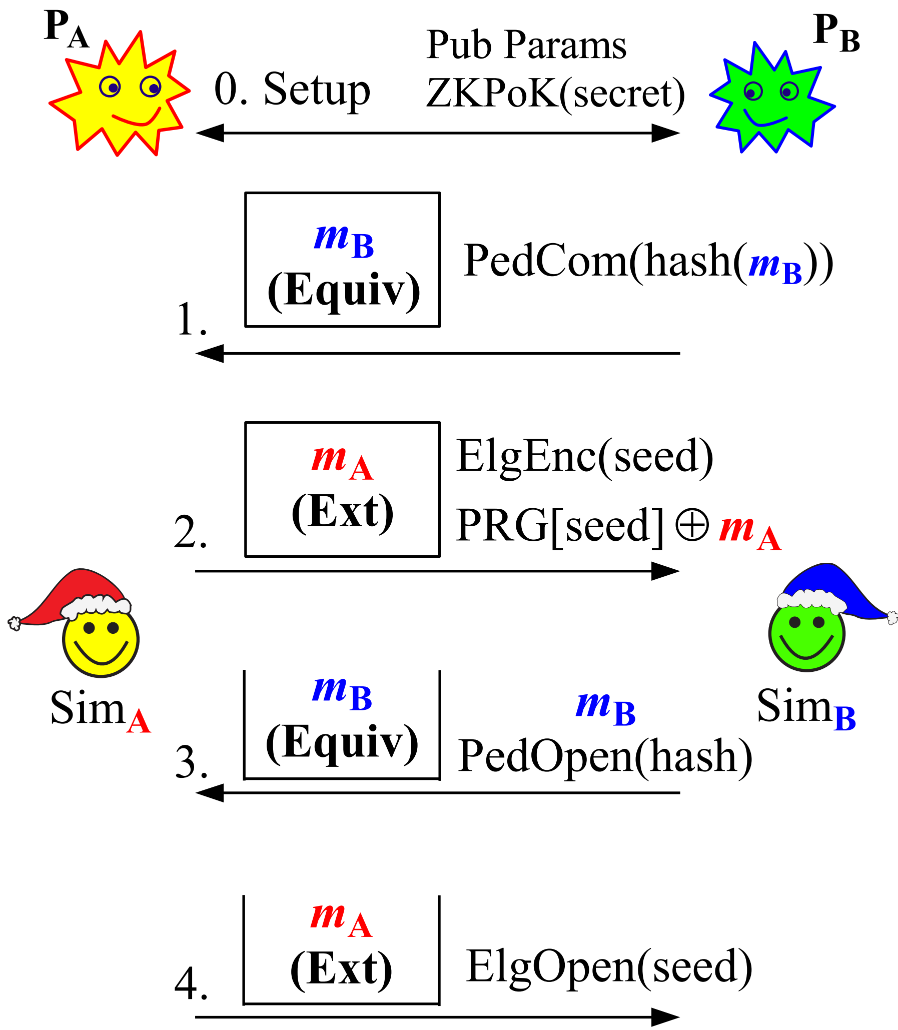
- Sim_A emulates abort in ideal world.

If P_B NOT-⊥ 1st time, but ⊥ 2nd time:

- Sim_A estimates Prob(⊥) ([GK96])
- Sim_A tries till $\mathbf{P_B}$ opens or #RWs ≈ $p(k)$/Prob(⊥)

Legend: Ped (Pedersen); ElgCom (ElGamal)

Closer look: possible instantiation and simulation (high level)



$$m = m_A \oplus m_B$$

Legend: Ped (Pedersen); ElgCom (ElGamal)

Case malicious P_A

- In step 0: Sim_B extract trapdoor
- In step 2: Sim_B extracts m_A ,
- In step 3: Sim_B Equiv-opens $m_B = m \oplus m_A$

Case malicious P_B

Optimistic simulation:
- In step 2: Sim_A commits random m_A
- In step 3: P_B opens m_B, then Sim_A rewinds
- In step 2: Sim_A commits $m_A = m \oplus m_B$
- In step 3: P_B opens m_B

If P_B aborts (⊥) first time in step 3:
- Sim_A emulates abort in ideal world.

If P_B NOT-⊥ 1st time, but ⊥ 2nd time:
- Sim_A estimates Prob(⊥) ([GK96])
- Sim_A tries till **P_B** opens or #RWs ≈ $p(k)$/Prob(⊥)

Legend: $p(k)$ (suitable polynomial of the sec. parameter)

Complexity

Complexity

Fixed offset:

- Setup (optional, e.g., to give trapdoor to simulator)
- Ext-Com scheme: 1 Com/Open of short seed
- Equiv-Com scheme: 1 Com/Open of short hash

Complexity

Fixed offset:

- Setup (optional, e.g., to give trapdoor to simulator)
- Ext-Com scheme: 1 Com/Open of short seed
- Equiv-Com scheme: 1 Com/Open of short hash

(may be based on ZK or cut-and-choose, but only related to 1 or 2 short strings)

Complexity

Fixed offset:

- Setup (optional, e.g., to give trapdoor to simulator)
- Ext-Com scheme: 1 Com/Open of short seed
- Equiv-Com scheme: 1 Com/Open of short hash

(may be based on ZK or cut-and-choose, but only related to 1 or 2 short strings)

Amortized for long strings:

- Communication: 2 bits per flipped coin
- Computation (per party): 1 PRG, 1 CR-Hash, 1 XOR

Roadmap

1. Simulatable coin-flipping and commitments

2. Protocol #1: coin-flipping (simulatable with rewinding)

3. Protocol #2: UC Commitment Scheme

4. Open questions / research directions

Toward an efficient UC-Com scheme

Toward an efficient UC-Com scheme

How to get an Ext&Equiv-Com for LONG strings, with:

- Communication expansion-rate 1+ε
- A FEW Ext-coms for SHORT strings
- A FEW Equiv-coms for SHORT strings
- Symmetric crypto operations (PRG, CR-Hash)

Toward an efficient UC-Com scheme

How to get an Ext&Equiv-Com for LONG strings, with:

- Communication expansion-rate 1+ε
- A FEW Ext-coms for SHORT strings → Ideal Ext-Com
- A FEW Equiv-coms for SHORT strings → Ideal Equiv-Com
- Symmetric crypto operations (PRG, CR-Hash)

UC-Coms do not exist in plain model

Toward an efficient UC-Com scheme

How to get an Ext&Equiv-Com for LONG strings, with:

- Communication expansion-rate 1+ε
- A FEW Ext-coms for SHORT strings → Ideal Ext-Com
- A FEW Equiv-coms for SHORT strings → Ideal Equiv-Com
- Symmetric crypto operations (PRG, CR-Hash)

(Ideal Ext-Com and Ideal Equiv-Com:) UC-Coms do not exist in plain model

(Other recent Rate-1 UC-Com schemes mentioned ahead: [GIKW14, DDGN14, CDD+15, FJNT16])

Toward an efficient UC-Com scheme

How to get an Ext&Equiv-Com for LONG strings, with:

- Communication expansion-rate 1+ε
- A FEW Ext-coms for SHORT strings → Ideal Ext-Com
- A FEW Equiv-coms for SHORT strings → Ideal Equiv-Com
- Symmetric crypto operations (PRG, CR-Hash)

(UC-Coms do not exist in plain model)

(Other recent Rate-1 UC-Com schemes mentioned ahead: [GIKW14, DDGN14, CDD+15, FJNT16])

Progress in two steps:

1. A comm. inefficient scheme, based on *cut-and-choose*
2. Improve comm. efficiency, with *authenticators* and an *erasure-code*

Toward an efficient UC-Com scheme

How to get an Ext&Equiv-Com for LONG strings, with:

- Communication expansion-rate 1+ε
- A FEW Ext-coms for SHORT strings → Ideal Ext-Com
- A FEW Equiv-coms for SHORT strings → Ideal Equiv-Com
- Symmetric crypto operations (PRG, CR-Hash)

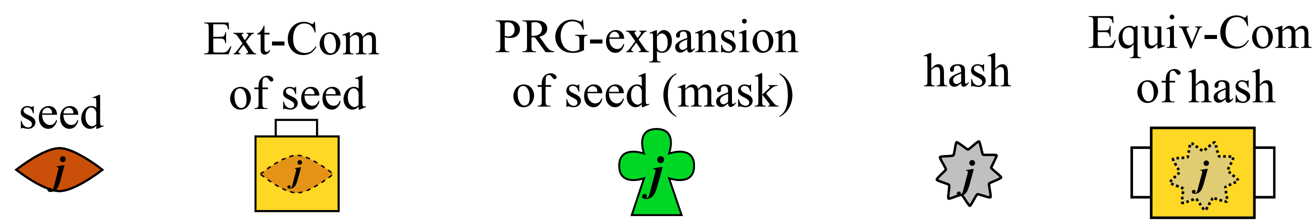
(Ideal Ext-Com and Ideal Equiv-Com:) UC-Coms do not exist in plain model

(Other recent Rate-1 UC-Com schemes mentioned ahead: [GIKW14, DDGN14, CDD+15, FJNT16])

Progress in two steps:

1. A comm. inefficient scheme, based on *cut-and-choose*
2. Improve comm. efficiency, with *authenticators* and an *erasure-code*

Pictorial notation:



Cut-and-choose warmup

(Warning: heavy slide)

Cut-and-choose warmup

(Warning: heavy slide)

1. Commit phase

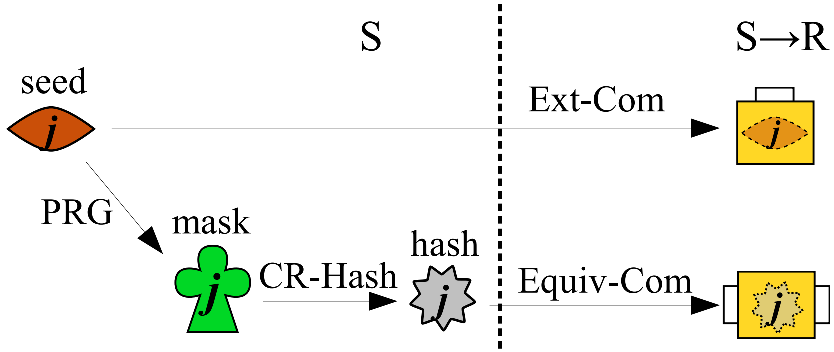
Cut-and-choose warmup

(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase

(Commit seeds and hashes) For $j \in \{1,\ldots,n\}$:



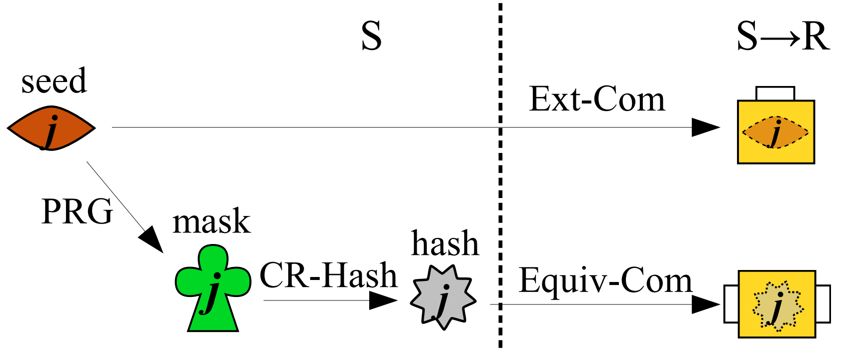
Cut-and-choose warmup

(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase

(Commit seeds and hashes) For $j \in \{1,\ldots, n\}$:



If S*, *hash* may differ from hash of PRG of seed

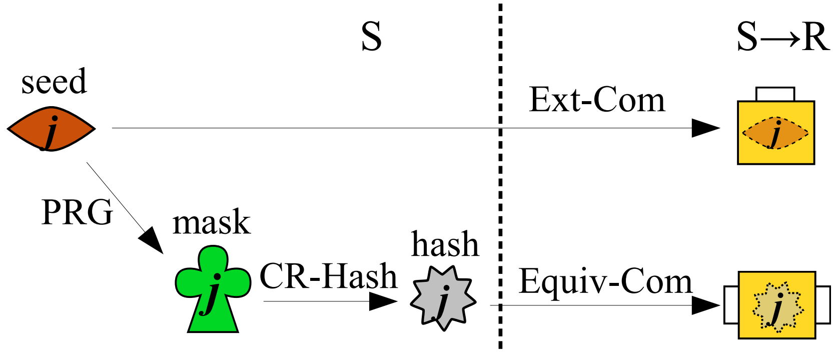
Cut-and-choose warmup

(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase

(Commit seeds and hashes) For $j \in \{1,\ldots, n\}$:



If S*, *hash* may differ from hash of PRG of seed

(Cut-and-Choose challenges) R → S:

$\{CHECK, EVAL\} \leftarrow^{\$} Partitions[\{1,\ldots, n\}]$

Cut-and-choose warmup

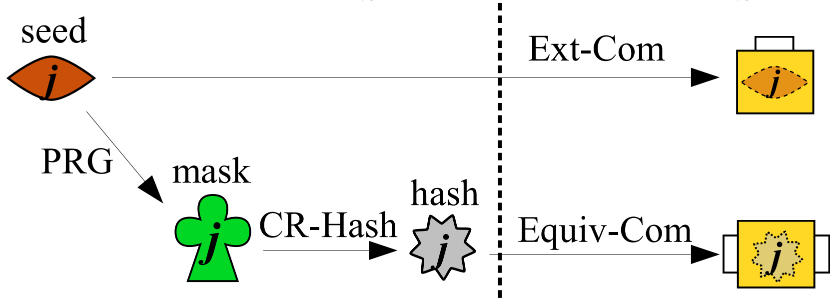
(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase

(Commit seeds and hashes) For $j \in \{1,\ldots, n\}$:

S | S→R



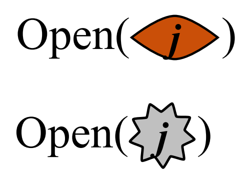
If S*, *hash* may differ from hash of PRG of seed

(Cut-and-Choose challenges) R → S:

$\{\text{CHECK}, \text{EVAL}\} \leftarrow^{\$} \text{Partitions}[\{1,\ldots, n\}]$

For $j \in$ CHECK:

S→R



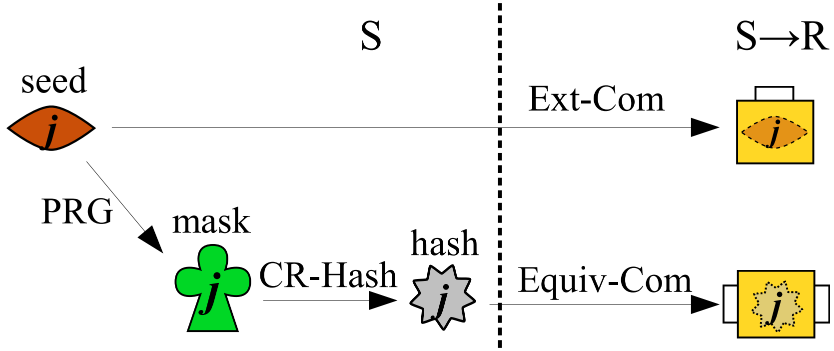
Cut-and-choose warmup

(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase

(Commit seeds and hashes) For $j \in \{1,\ldots,n\}$:

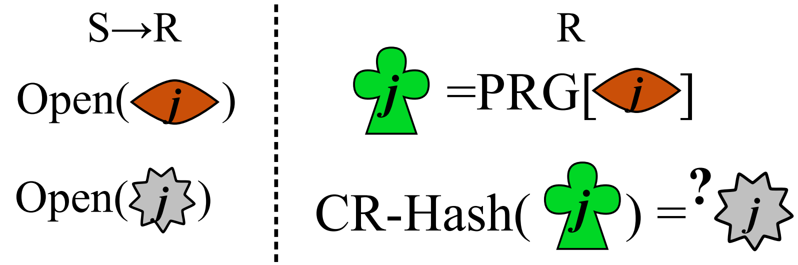


If S*, *hash* may differ from hash of PRG of seed

(Cut-and-Choose challenges) R → S:

$\{\text{CHECK}, \text{EVAL}\} \leftarrow^{\$} \text{Partitions}[\{1,\ldots,n\}]$

For $j \in$ CHECK:



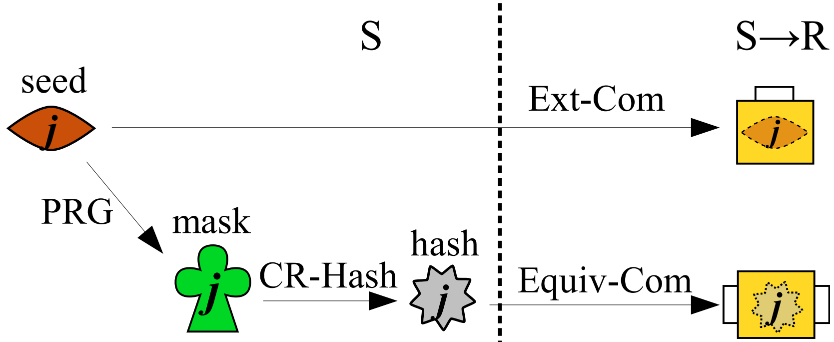
Cut-and-choose warmup

(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase

(Commit seeds and hashes) For $j \in \{1,\ldots, n\}$:

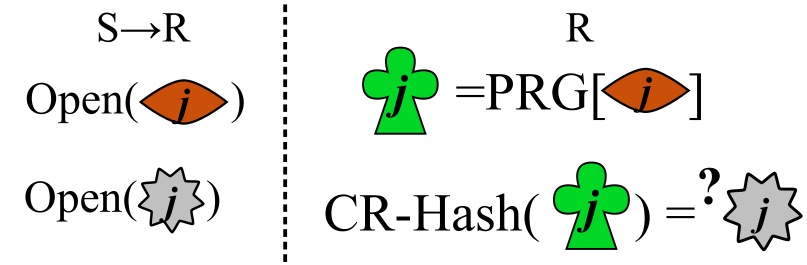


If S*, *hash* may differ from hash of PRG of seed

(Cut-and-Choose challenges) R → S:

$\{\text{CHECK}, \text{EVAL}\} \leftarrow^{\$} \text{Partitions}[\{1,\ldots, n\}]$

For $j \in$ CHECK:



(R believes majority EVAL instances are OK)

Part 3
Outline
Warmup
Improve
Complex
Rel W

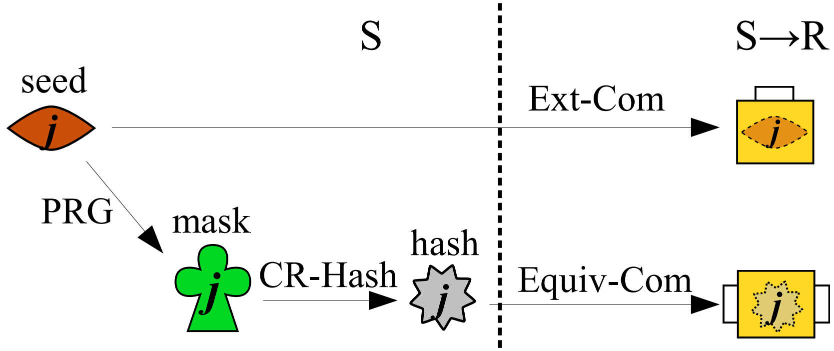
Cut-and-choose warmup

(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase

(Commit seeds and hashes) For $j \in \{1,\ldots,n\}$:

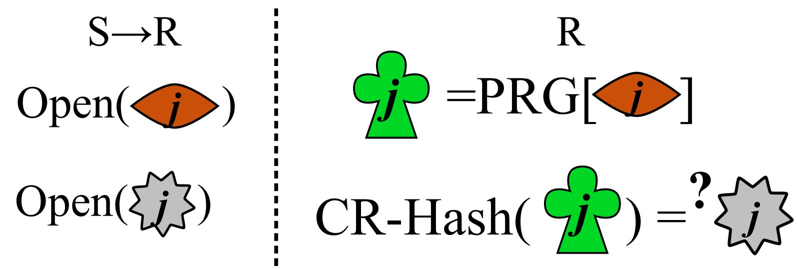


If S*, *hash* may differ from hash of PRG of seed

(Cut-and-Choose challenges) R → S:

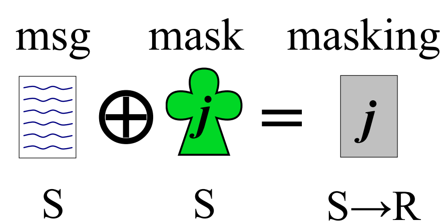
$\{\text{CHECK}, \text{EVAL}\} \xleftarrow{\$} \text{Partitions}[\{1,\ldots,n\}]$

For $j \in$ CHECK:



(R believes majority EVAL instances are OK)

For $j \in$ EVAL:



Part 3
Outline
Warmup
Improve
Complex
Rel W

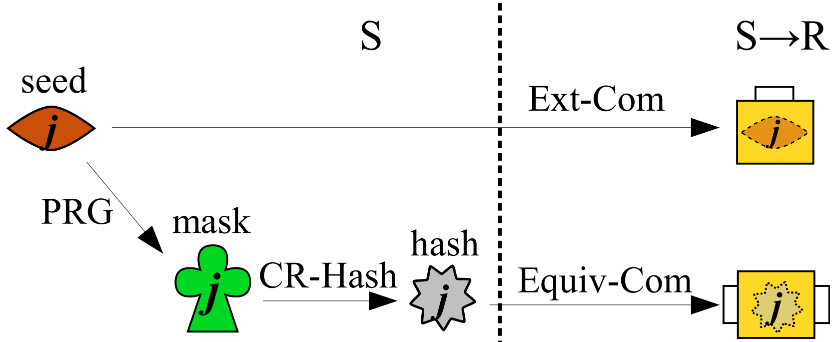
Cut-and-choose warmup

(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase

(Commit seeds and hashes) For $j \in \{1,\ldots,n\}$:

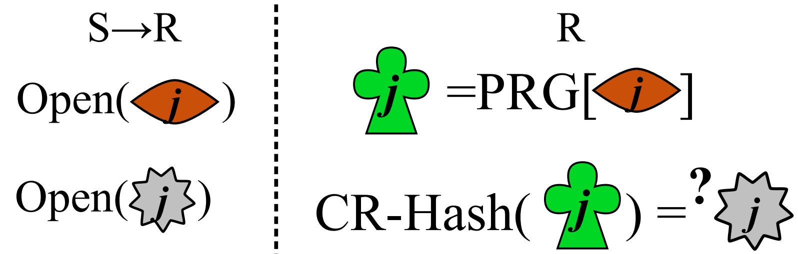


If S*, *hash* may differ from hash of PRG of seed

(Cut-and-Choose challenges) R → S:

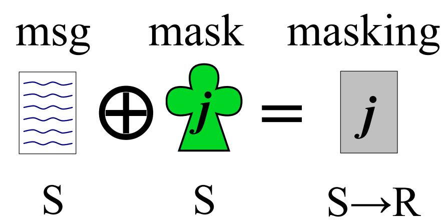
$\{\text{CHECK}, \text{EVAL}\} \xleftarrow{\$} \text{Partitions}[\{1,\ldots,n\}]$

For $j \in$ CHECK:

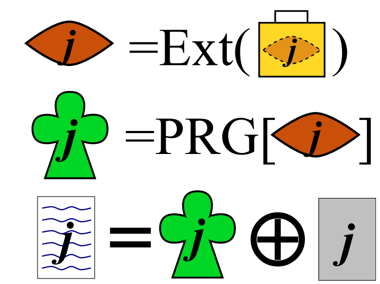


(R believes majority EVAL instances are OK)

For $j \in$ EVAL:



Extraction (Sim_R):

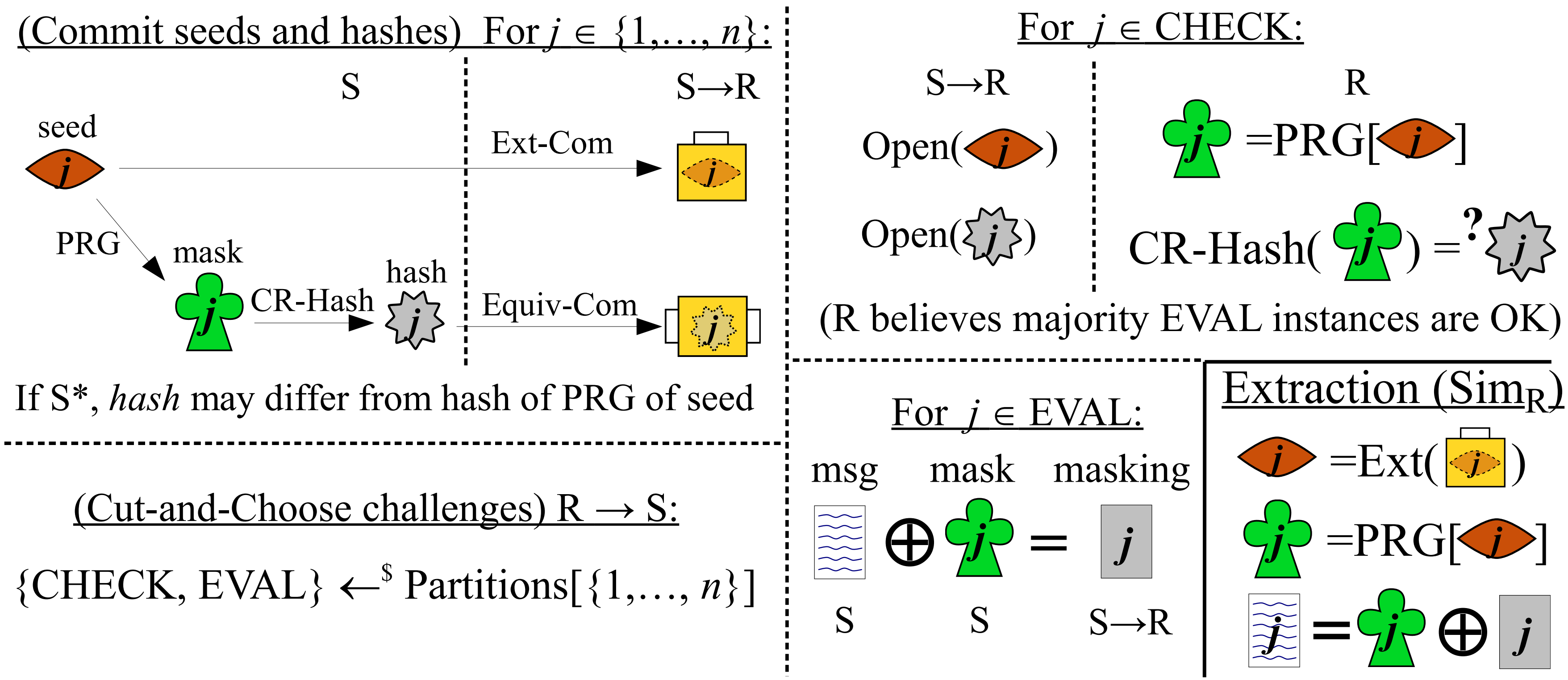


Cut-and-choose warmup

(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase



2. Open phase

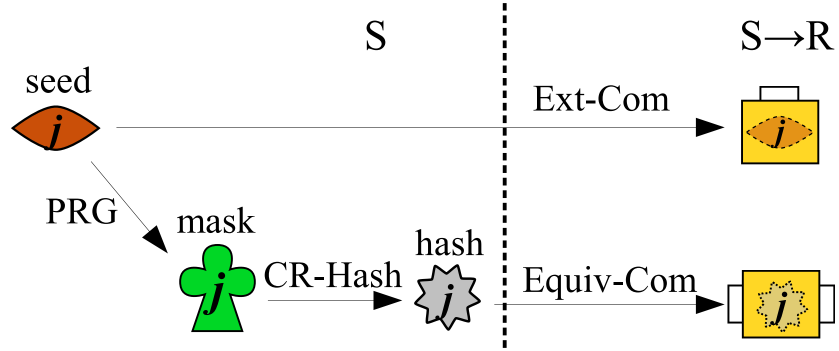
Cut-and-choose warmup

(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase

(Commit seeds and hashes) For $j \in \{1,\ldots,n\}$:



If S*, *hash* may differ from hash of PRG of seed

(Cut-and-Choose challenges) R → S:

$\{\text{CHECK}, \text{EVAL}\} \leftarrow^{\$} \text{Partitions}[\{1,\ldots,n\}]$

For $j \in$ CHECK:

S→R: Open(seed j), Open(hash j)

R: mask j =PRG[seed j]; CR-Hash(mask j) $=^?$ hash j

(R believes majority EVAL instances are OK)

For $j \in$ EVAL:

msg ⊕ mask = masking

S, S, S→R

Extraction (Sim_R):

seed j =Ext(Ext-Com j)

mask j =PRG[seed j]

msg j = mask j ⊕ masking j

2. Open phase

S→R:

Open(hash j) : $j \in$ Eval

Reveal msg

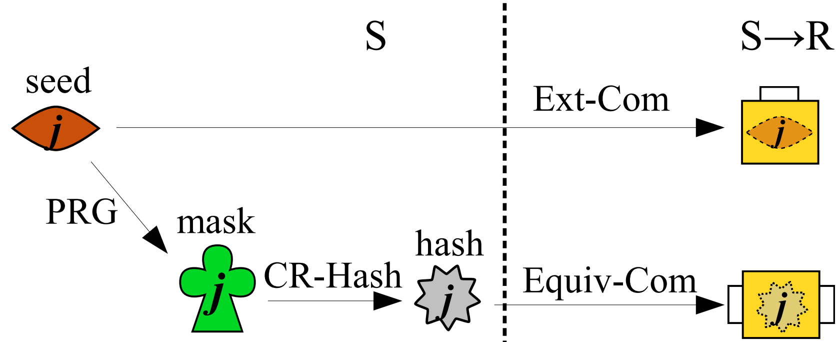
Cut-and-choose warmup

(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase

(Commit seeds and hashes) For $j \in \{1,\ldots,n\}$:

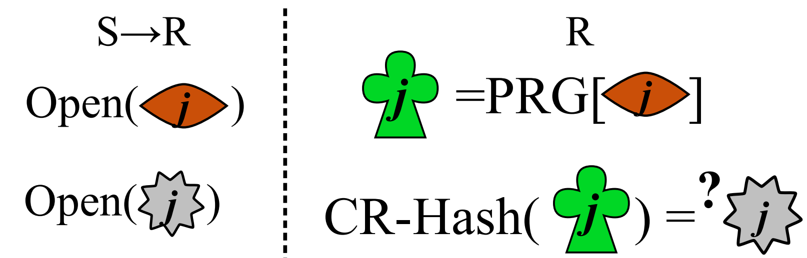


If S*, *hash* may differ from hash of PRG of seed

(Cut-and-Choose challenges) R → S:

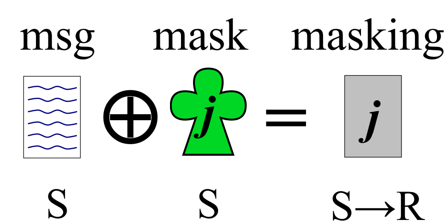
$\{\text{CHECK}, \text{EVAL}\} \leftarrow^{\$} \text{Partitions}[\{1,\ldots,n\}]$

For $j \in$ CHECK:

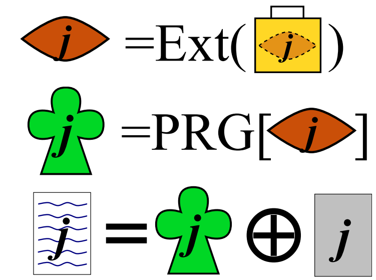


(R believes majority EVAL instances are OK)

For $j \in$ EVAL:

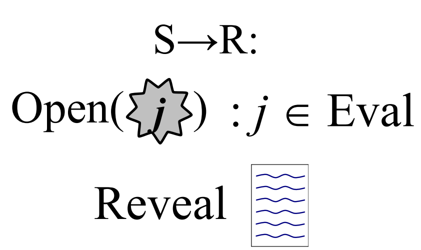


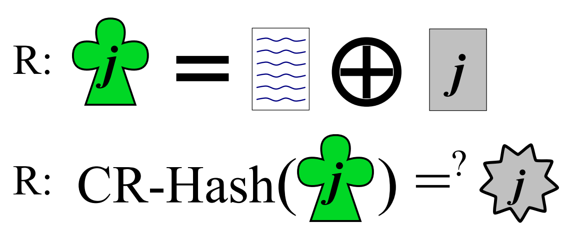
Extraction (Sim_R):



2. Open phase

S→R:





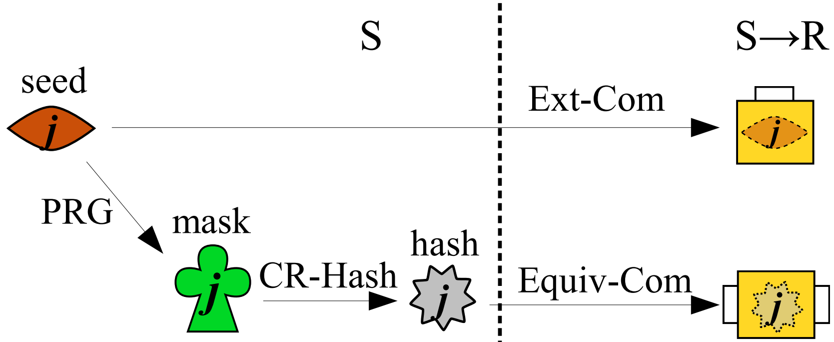
Cut-and-choose warmup

(Warning: heavy slide)

Legend:
S = Sender; R = Receiver
n = # instances;
j = index of instance

1. Commit phase

(Commit seeds and hashes) For $j \in \{1, \ldots, n\}$:



If S*, *hash* may differ from hash of PRG of seed

(Cut-and-Choose challenges) R → S:

{CHECK, EVAL} $\leftarrow^{\$}$ Partitions[{1,…, n}]

For $j \in$ CHECK:

S→R

Open(j)

Open(j)

R

j =PRG[j]

CR-Hash(j) $=^?$ j

(R believes majority EVAL instances are OK)

For $j \in$ EVAL:

msg ⊕ mask = masking

S, S, S→R

Extraction (Sim_R):

j =Ext(j)

j =PRG[j]

j = j ⊕ j

2. Open phase

S→R:

Open(j) : $j \in$ Eval

Reveal

R: j = ⊕ j

R: CR-Hash(j) $=^?$ j

Equivocation by Sim_S:

j = ⊕ j

j =CR-Hash(j)

Equiv-Open(j)

Improving communication

Part 3
Outline
Warmup
Improve
Complex
Rel W

Improving communication

Legend:
m = message; n = # instances;
e = #(EVAL); v = #(CHECK)

Problems with the warmup protocol

- Ensure correct extraction of message m implies many instances
- E.g. 40 bits statistical security $\Rightarrow n \geq 123$, e.g. $(n, v, e) = (123, 74, 49)$.
- High communication complexity: $|m| \times e$

Part 3
Outline
Warmup
Improve
Complex
Rel W

Improving communication

Legend:
m = message; n = # instances;
e = #(EVAL); v = #(CHECK)

Problems with the warmup protocol

- Ensure correct extraction of message m implies many instances
- E.g. 40 bits statistical security ⇒ $n \geq 123$, e.g. $(\boldsymbol{n}, \boldsymbol{v}, \boldsymbol{e}) = (\mathbf{123}, \mathbf{74}, \mathbf{49})$.
- High communication complexity: $|\boldsymbol{m}| \times \boldsymbol{e}$

Add two ingredients:

Improving communication

Legend:
m = message; n = # instances;
e = #(EVAL); v = #(CHECK)

Problems with the warmup protocol

- Ensure correct extraction of message m implies many instances
- E.g. 40 bits statistical security $\Rightarrow n \geq 123$, e.g. $(n, v, e) = (123, 74, 49)$.
- High communication complexity: $|m| \times e$

Add two ingredients:

- Authenticators: "authenticate" the message before masking it

Improving communication

Legend:
m = message; n = # instances;
e = #(EVAL); v = #(CHECK)

Problems with the warmup protocol

- Ensure correct extraction of message m implies many instances
- E.g. 40 bits statistical security $\Rightarrow n \geq 123$, e.g. $(n, v, e) = (123, 74, 49)$.
- High communication complexity: $|m| \times e$

Add two ingredients:

- Authenticators: "authenticate" the message before masking it
  $\Rightarrow$ Sim_R can verify each tentative extracted m for correctness
  $\Rightarrow$ 1 *good* Eval instance is enough $\Rightarrow$ better params $(n, v, e) = (41, \geq 21, \leq 20)$

Improving communication

Legend:
m = message; n = # instances;
e = #(EVAL); v = #(CHECK)

Problems with the warmup protocol

- Ensure correct extraction of message m implies many instances
- E.g. 40 bits statistical security $\Rightarrow n \geq 123$, e.g. $(n, v, e) = (123, 74, 49)$.
- High communication complexity: $|m| \times e$

Add two ingredients:

- Authenticators: "authenticate" the message before masking it
  $\Rightarrow$ Sim_R can verify each tentative extracted m for correctness
  $\Rightarrow$ 1 *good* Eval instance is enough $\Rightarrow$ better params $(n, v, e) = (41, \geq 21, \leq 20)$
- Erasure-code: *split* message into smaller *fragments* (aka shares)

Improving communication

Legend:
m = message; n = # instances;
e = #(EVAL); v = #(CHECK)

Problems with the warmup protocol

- Ensure correct extraction of message m implies many instances

- E.g. 40 bits statistical security $\Rightarrow n \geq 123$, e.g. **(n, v, e) = (123, 74, 49)**.

- High communication complexity: $|m| \times e$

Add two ingredients:

- Authenticators: "authenticate" the message before masking it

  $\Rightarrow$ Sim_R can verify each tentative extracted m for correctness

  $\Rightarrow$ 1 *good* Eval instance is enough $\Rightarrow$ better params (n, v, e) = (41, $\geq$**21**, $\leq$**20**)

- Erasure-code: *split* message into smaller *fragments* (aka shares)

  $\Rightarrow$ Mask each ("authenticated") share, instead of full message m

  $\Rightarrow$ Sim_R extracts m if it extracts enough (t) correct shares out of e shares

  $\Rightarrow$ New params, e.g., (n, v, e, t) = (119, 73, **46**, **23**) $\Rightarrow$ Comm = $|m| \times e / t$

Complexity

Complexity

E.g. C&C and erasure code parameters:

- ***n* = 119** (# instances in cut-and-choose)
- ***v* = 73** (# committed seeds and hashes)
- ***e* = 46** (# shares = # Eval instances)
- ***t* = 23** (# good shares needed by Simulator)

Complexity

<u>E.g. C&C and erasure code parameters:</u>

- ***n* = 119** (# instances in cut-and-choose)
- ***v* = 73** (# committed seeds and hashes)
- ***e* = 46** (# shares = # Eval instances)
- ***t* = 23** (# good shares needed by Simulator)

<u>Comm. and comp. rates:</u>

- ***r* = *e* / *t* = 2** (comm. expansion-rate in commit phase, with rate-1 erasure code)

Complexity

E.g. C&C and erasure code parameters:

- ***n* = 119** (# instances in cut-and-choose)
- ***v* = 73** (# committed seeds and hashes)
- ***e* = 46** (# shares = # Eval instances)
- ***t* = 23** (# good shares needed by Simulator)

Comm. and comp. rates:

- ***r* = *e* / *t* = 2** (comm. expansion-rate in commit phase, with rate-1 erasure code)
- ***r'* = *n* / *t* = 5.17** (length of overall PRG output divided by message length) (same in respect to CR-Hash input)

Complexity

E.g. C&C and erasure code parameters:

- ***n* = 119** (# instances in cut-and-choose)
- ***v* = 73** (# committed seeds and hashes)
- ***e* = 46** (# shares = # Eval instances)
- ***t* = 23** (# good shares needed by Simulator)

Comm. and comp. rates:

- ***r* = *e* / *t* = 2** (comm. expansion-rate in commit phase, with rate-1 erasure code)
- ***r'* = *n* / *t* = 5.17** (length of overall PRG output divided by message length) (same in respect to CR-Hash input)

Some notes:

Part 3
Outline
Warmup
Improve
Complex
Rel W

Complexity

E.g. C&C and erasure code parameters:

- ***n*** **= 119** (# instances in cut-and-choose)
- ***v*** **= 73** (# committed seeds and hashes)
- ***e*** **= 46** (# shares = # Eval instances)
- ***t*** **= 23** (# good shares needed by Simulator)

Comm. and comp. rates:

- ***r*** **=** ***e*** **/** ***t*** **= 2** (comm. expansion-rate in commit phase, with rate-1 erasure code)
- ***r'*** **=** ***n*** **/** ***t*** **= 5.17** (length of overall PRG output divided by message length) (same in respect to CR-Hash input)

Some notes:

- Can decrease rates *r* and *r'* closer to 1 (at the cost of larger erasure-code parameters)

Complexity

E.g. C&C and erasure code parameters:

- ***n* = 119** (# instances in cut-and-choose)
- ***v* = 73** (# committed seeds and hashes)
- ***e* = 46** (# shares = # Eval instances)
- ***t* = 23** (# good shares needed by Simulator)

Comm. and comp. rates:

- ***r* = *e* / *t* = 2** (comm. expansion-rate in commit phase, with rate-1 erasure code)
- ***r'* = *n* / *t* = 5.17** (length of overall PRG output divided by message length) (same in respect to CR-Hash input)

Some notes:

- Can decrease rates *r* and *r'* closer to 1 (at the cost of larger erasure-code parameters)
- Sender and Receiver only need to encode; only simulator needs to decode

Complexity

E.g. C&C and erasure code parameters:

- $\boldsymbol{n}$ **= 119** (# instances in cut-and-choose)
- $\boldsymbol{v}$ **= 73** (# committed seeds and hashes)
- $\boldsymbol{e}$ **= 46** (# shares = # Eval instances)
- $\boldsymbol{t}$ **= 23** (# good shares needed by Simulator)

Comm. and comp. rates:

- $\boldsymbol{r = e / t}$ **= 2** (comm. expansion-rate in commit phase, with rate-1 erasure code)
- $\boldsymbol{r' = n / t}$ **= 5.17** (length of overall PRG output divided by message length) (same in respect to CR-Hash input)

Some notes:

- Can decrease rates r and r' closer to 1 (at the cost of larger erasure-code parameters)
- Sender and Receiver only need to encode; only simulator needs to decode
- # hashes (and # Equiv-Coms) can be reduced to 1, if allowing delayed verification (Sim_R can still extract, or detect non-ability of Sender to open)

Complexity

E.g. C&C and erasure code parameters:

- $\boldsymbol{n}$ **= 119** (# instances in cut-and-choose)
- $\boldsymbol{v}$ **= 73** (# committed seeds and hashes)
- $\boldsymbol{e}$ **= 46** (# shares = # Eval instances)
- $\boldsymbol{t}$ **= 23** (# good shares needed by Simulator)

Comm. and comp. rates:

- $\boldsymbol{r = e / t = 2}$ (comm. expansion-rate in commit phase, with rate-1 erasure code)
- $\boldsymbol{r' = n / t = 5.17}$ (length of overall PRG output divided by message length) (same in respect to CR-Hash input)

Some notes:

- Can decrease rates r and r' closer to 1 (at the cost of larger erasure-code parameters)
- Sender and Receiver only need to encode; only simulator needs to decode
- # hashes (and # Equiv-Coms) can be reduced to 1, if allowing delayed verification (Sim_R can still extract, or detect non-ability of Sender to open)
- Ideal Equiv-Com and ideal Ext-Com can be instantiated in other setups, e.g. CRS

Complexity

E.g. C&C and erasure code parameters:

- ***n* = 119** (# instances in cut-and-choose)
- ***v* = 73** (# committed seeds and hashes)
- ***e* = 46** (# shares = # Eval instances)
- ***t* = 23** (# good shares needed by Simulator)

Comm. and comp. rates:

- ***r* = *e* / *t* = 2** (comm. expansion-rate in commit phase, with rate-1 erasure code)
- ***r’* = *n* / *t* = 5.17** (length of overall PRG output divided by message length) (same in respect to CR-Hash input)

Some notes:

- Can decrease rates *r* and *r*’ closer to 1 (at the cost of larger erasure-code parameters)
- Sender and Receiver only need to encode; only simulator needs to decode
- # hashes (and # Equiv-Coms) can be reduced to 1, if allowing delayed verification (Sim_R can still extract, or detect non-ability of Sender to open)
- Ideal Equiv-Com and ideal Ext-Com can be instantiated in other setups, e.g. CRS
- Interaction due to cut-&-choose can be removed by using Non-Programmable Random Oracle (and increasing statistical security parameter)

Some related work

Some related work

<u>Some UC-Com Schemes in 2011, 2013</u>: [Lin11, FLM11, BCPV13]

Some related work

<u>Some UC-Com Schemes in 2011, 2013</u>: [Lin11, FLM11, BCPV13]

- Comm: several group elements exchanged per committed short-string.

Some related work

<u>Some UC-Com Schemes in 2011, 2013</u>: [Lin11, FLM11, BCPV13]

- Comm: several group elements exchanged per committed short-string.
- Comp: several exponentiations per committed short string.

Some related work

<u>Some UC-Com Schemes in 2011, 2013</u>: [Lin11, FLM11, BCPV13]

- Comm: several group elements exchanged per committed short-string.

- Comp: several exponentiations per committed short string.

- Some constructions achieve adaptive security.

Some related work

<u>Some UC-Com Schemes in 2011, 2013</u>: [Lin11, FLM11, BCPV13]

- Comm: several group elements exchanged per committed short-string.
- Comp: several exponentiations per committed short string.
- Some constructions achieve adaptive security.

<u>2014 onward – rate-1+ε UC-Com schemes (static security)</u>

Some related work

<u>Some UC-Com Schemes in 2011, 2013</u>: [Lin11, FLM11, BCPV13]

- Comm: several group elements exchanged per committed short-string.
- Comp: several exponentiations per committed short string.
- Some constructions achieve adaptive security.

<u>2014 onward – rate-1+ε UC-Com schemes (static security)</u>

- [GIKW14]:
  - First proposal; uses δ-OT instead of C&C.
  - Requires Error-correction code (ECC, for semantic errors), instead of erasure code.

Some related work

Some UC-Com Schemes in 2011, 2013: [Lin11, FLM11, BCPV13]

- Comm: several group elements exchanged per committed short-string.

- Comp: several exponentiations per committed short string.

- Some constructions achieve adaptive security.

2014 onward – rate-1+ε UC-Com schemes (static security)

- [GIKW14]:
  - First proposal; uses δ-OT instead of C&C.
  - Requires Error-correction code (ECC, for semantic errors), instead of erasure code.
- [DDGN14,CDD+15]
  - Also OT and ECC based
  - Enable Homomorphic commitments.

Some related work

Some UC-Com Schemes in 2011, 2013: [Lin11, FLM11, BCPV13]

- Comm: several group elements exchanged per committed short-string.

- Comp: several exponentiations per committed short string.

- Some constructions achieve adaptive security.

2014 onward – rate-1+ε UC-Com schemes (static security)

- [GIKW14]:
  - First proposal; uses δ-OT instead of C&C.
  - Requires Error-correction code (ECC, for semantic errors), instead of erasure code.
- [DDGN14,CDD+15]
  - Also OT and ECC based
  - Enable Homomorphic commitments.
- [FJNT16] (Also OT based):
  - Uses consistency check to allow erasure code instead of ECC
  - Enable homomorphic commitments.

Roadmap

1. Simulatable coin-flipping and commitments

2. Protocol #1: coin-flipping (simulatable with rewinding)

3. Protocol #2: UC Commitment Scheme

4. Open questions / research directions

Possible research directions:

Part 4

Open

Thanks

Refs

Possible research directions:

- Formalize ideal Ext-but-not-Equiv and Equiv-but-not-Ext Com schemes (initial attempt at full version of the paper)

Possible research directions:

- Formalize ideal Ext-but-not-Equiv and Equiv-but-not-Ext Com schemes (initial attempt at full version of the paper)
- Actual instantiations / efficiency measurement (erasure code, …) / tradeoffs (communication vs. computation)

Possible research directions:

- Formalize ideal Ext-but-not-Equiv and Equiv-but-not-Ext Com schemes (initial attempt at full version of the paper)
- Actual instantiations / efficiency measurement (erasure code, …) / tradeoffs (communication vs. computation)
- Efficient UC-Com schemes (rate-1, linear-time) in adaptive model?

Possible research directions:

- Formalize ideal Ext-but-not-Equiv and Equiv-but-not-Ext Com schemes (initial attempt at full version of the paper)
- Actual instantiations / efficiency measurement (erasure code, …) / tradeoffs (communication vs. computation)
- Efficient UC-Com schemes (rate-1, linear-time) in adaptive model?
- Decrease erasure-code parameters needed for statistical security parameter?

Possible research directions:

- Formalize ideal Ext-but-not-Equiv and Equiv-but-not-Ext Com schemes (initial attempt at full version of the paper)
- Actual instantiations / efficiency measurement (erasure code, …) / tradeoffs (communication vs. computation)
- Efficient UC-Com schemes (rate-1, linear-time) in adaptive model?
- Decrease erasure-code parameters needed for statistical security parameter?
- Homomorphic properties?

Possible research directions:

- Formalize ideal Ext-but-not-Equiv and Equiv-but-not-Ext Com schemes (initial attempt at full version of the paper)
- Actual instantiations / efficiency measurement (erasure code, …) / tradeoffs (communication vs. computation)
- Efficient UC-Com schemes (rate-1, linear-time) in adaptive model?
- Decrease erasure-code parameters needed for statistical security parameter?
- Homomorphic properties?
- Selective opening of parts of message?

Possible research directions:

- Formalize ideal Ext-but-not-Equiv and Equiv-but-not-Ext Com schemes (initial attempt at full version of the paper)
- Actual instantiations / efficiency measurement (erasure code, …) / tradeoffs (communication vs. computation)
- Efficient UC-Com schemes (rate-1, linear-time) in adaptive model?
- Decrease erasure-code parameters needed for statistical security parameter?
- Homomorphic properties?
- Selective opening of parts of message?
- More efficient UC Coin-Flipping (2 bits / flipped coin & comp. efficient)?

Thank you for your attention

?

Very Efficient Simulatable Flipping of Many Coins into-a-well

luis.papers@gmail.com

https://ia.cr/2015/640

References mentioned in this presentation

(More references in paper)

- [Blu83]: Blum: Coin flipping by telephone – a protocol for solving impossible problems. SIGACT News 15, 23–27 (1983). Appeared also at CRYPTO 1981
- [Lin03]: Lindell. Parallel coin-tossing and constant-round secure two-party computation. Jr Cryptology, 16(3), 2003.
- [PW09]: Pass and Wee. Black-box constructions of two-party protocols from one-way functions. TCC 2009
- [Lin11]: Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. EUROCRYPT 2011
- [BCVP13]: Blazy and Chevalier and Pointcheval, and Vergnaud. Analysis and improvement of Lindell's UC-secure commitment schemes. ACNS 2013
- [FLM11]: Fischlin and Libert and Manulis. Non-interactive and re-usable universally composable string commitments with adaptive security. ASIACRYPT 2011
- [GK96]: Goldreich and Kahan. How to construct constant-round zero-knowledge proof systems for NP. Jr Cryptology, 9(3), 1996.
- [GIKW14] Garay and Ishai and Kumaresan and Wee. On the complexity of UC commitments. EUROCRYPT 2014
- [CDD+15]: Cascudo and Damgård and David and Giacomelli and Nielsen and Trifiletti. Additively homomorphic UC commitments with optimal amortized overhead. PKC 2015
- [DDGN14]: Damgård and David and Giacomelli and Nielsen. Compact VSS and efficient homomorphic UC commitments. ASIACRYPT 2014
- [FJNT16]: Frederiksen, Jakobsen, Nielsen, and Trifiletti. On the complexity of additively homomorphic UC commitments. TCC 2016-A

References mentioned in this presentation

(More references in paper)

- [Blu83]: Blum: Coin flipping by telephone – a protocol for solving impossible problems. SIGACT News 15, 23–27 (1983). Appeared also at CRYPTO 1981
- [Lin03]: Lindell. Parallel coin-tossing and constant-round secure two-party computation. Jr Cryptology, 16(3), 2003.
- [PW09]: Pass and Wee. Black-box constructions of two-party protocols from one-way functions. TCC 2009
- [Lin11]: Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. EUROCRYPT 2011
- [BCVP13]: Blazy and Chevalier and Pointcheval, and Vergnaud. Analysis and improvement of Lindell's UC-secure commitment schemes. ACNS 2013
- [FLM11]: Fischlin and Libert and Manulis. Non-interactive and re-usable universally composable string commitments with adaptive security. ASIACRYPT 2011
- [GK96]: Goldreich and Kahan. How to construct constant-round zero-knowledge proof systems for NP. Jr Cryptology, 9(3), 1996.
- [GIKW14] Garay and Ishai and Kumaresan and Wee. On the complexity of UC commitments. EUROCRYPT 2014
- [CDD+15]: Cascudo and Damgård and David and Giacomelli and Nielsen and Trifiletti. Additively homomorphic UC commitments with optimal amortized overhead. PKC 2015
- [DDGN14]: Damgård and David and Giacomelli and Nielsen. Compact VSS and efficient homomorphic UC commitments. ASIACRYPT 2014
- [FJNT16]: Frederiksen, Jakobsen, Nielsen, and Trifiletti. On the complexity of additively homomorphic UC commitments. TCC 2016-A

The following images were obtained (or edited) from files in the public domain (downloaded at clker dot com):