Introduction to Zero-Knowledge Proofs and the MPC-in-the-Head Paradigm

Matthieu Rivain

PQ-TLS Summer School

Jun 18, 2024, Anglet

CryptoExperts

WE INNOVATE TO SECURE YOUR BUSINESS

Roadmap

- Today:
    - Quick Intro
    - Introduction to Zero-Knowledge Proofs
    - Introduction to the MPC-in-the-Head Paradigm
- Tomorrow:
    - Modern MPC-in-the-Head Techniques
    - Specific Post-Quantum Signatures

Roadmap

- Today:
    - Quick Intro
    - Introduction to Zero-Knowledge Proofs
    - Introduction to the MPC-in-the-Head Paradigm
- Tomorrow:
    - Modern MPC-in-the-Head Techniques
    - Specific Post-Quantum Signatures

Quick Intro to MPC in the Head

One-way function

$$F : x \mapsto y$$

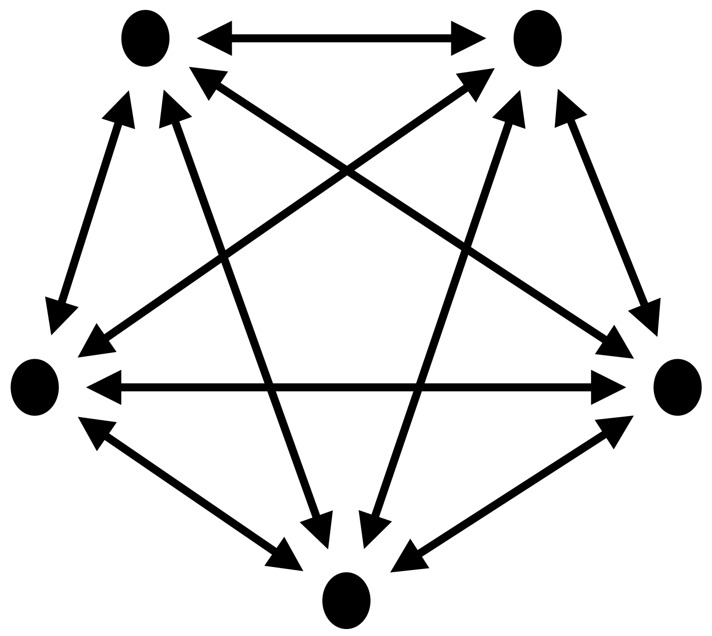
E.g. AES, MQ system,
Syndrome decoding

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)



Input sharing $[\![x]\!]$

Joint evaluation of:

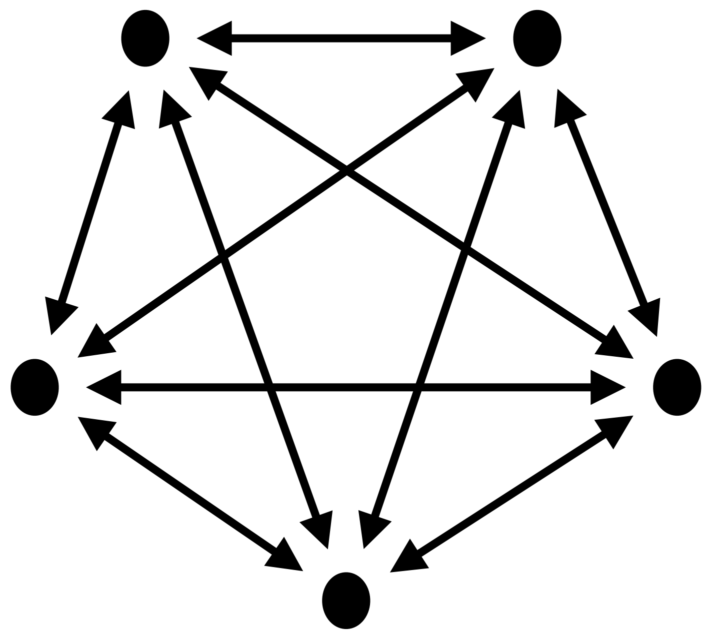
$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

One-way function

$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Multiparty computation (MPC)

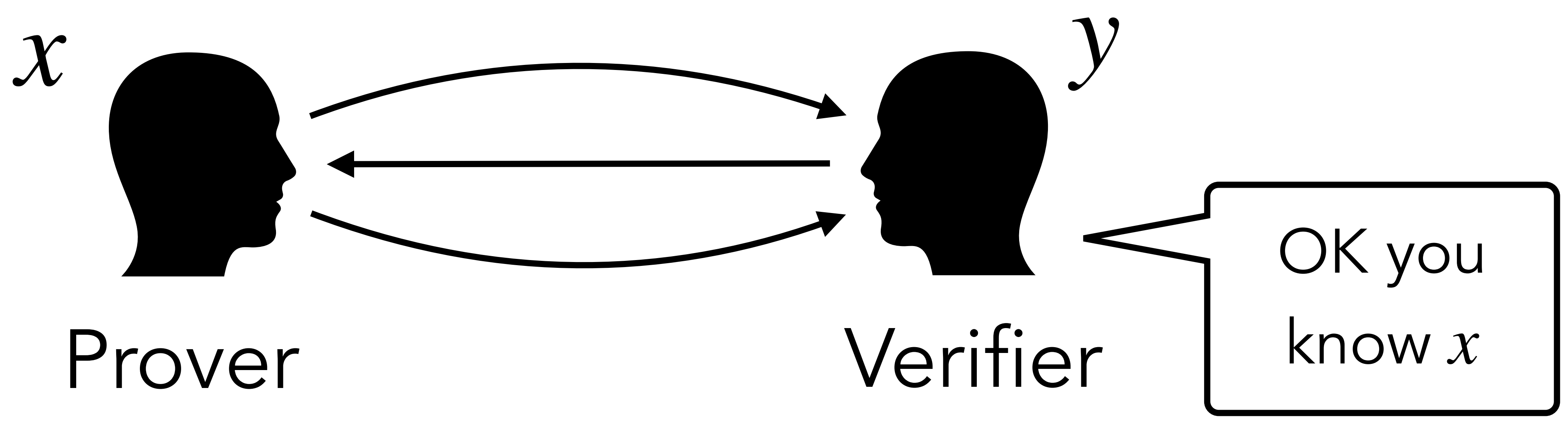


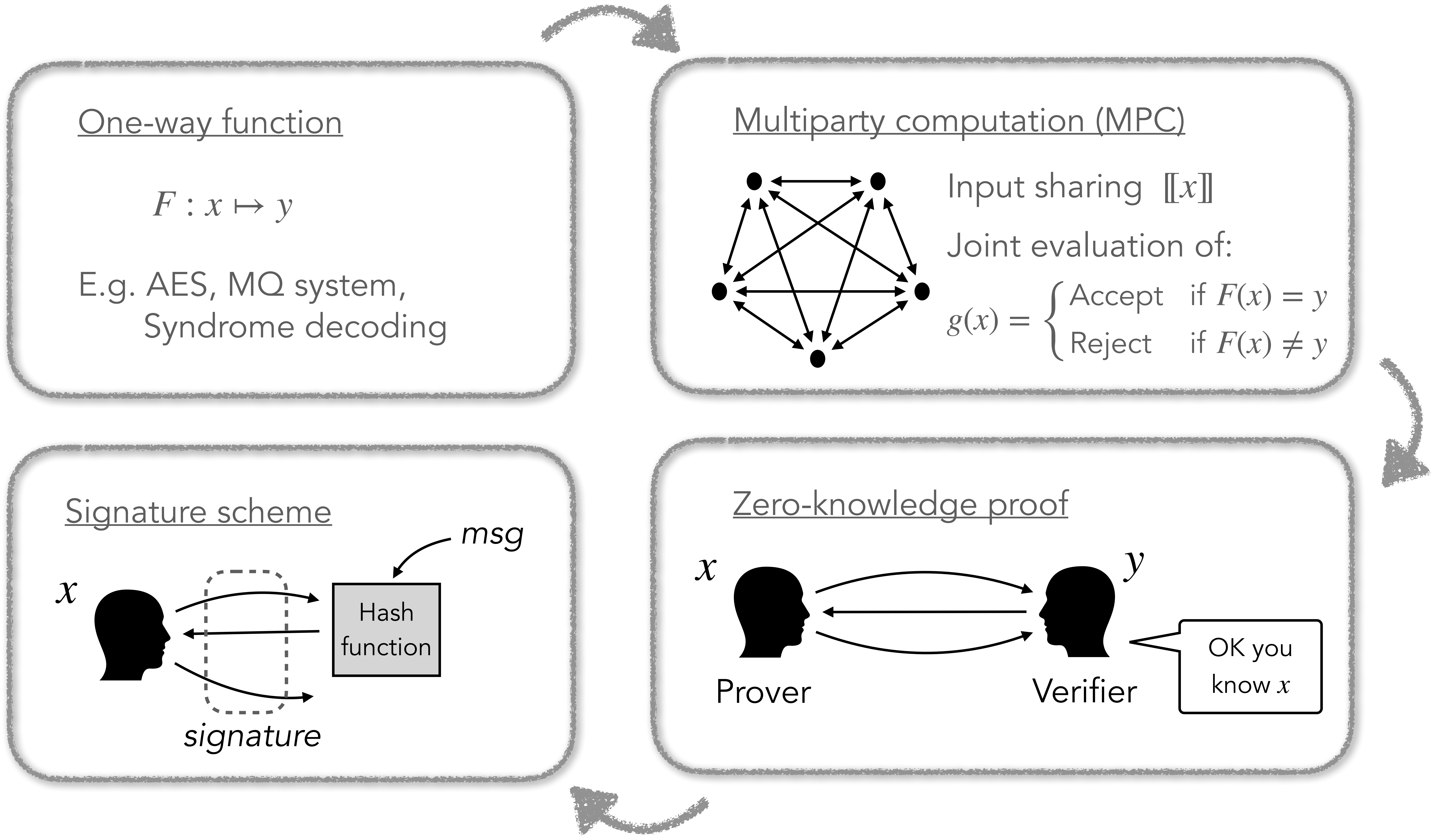
Input sharing $[\![x]\!]$

Joint evaluation of:

$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

Zero-knowledge proof



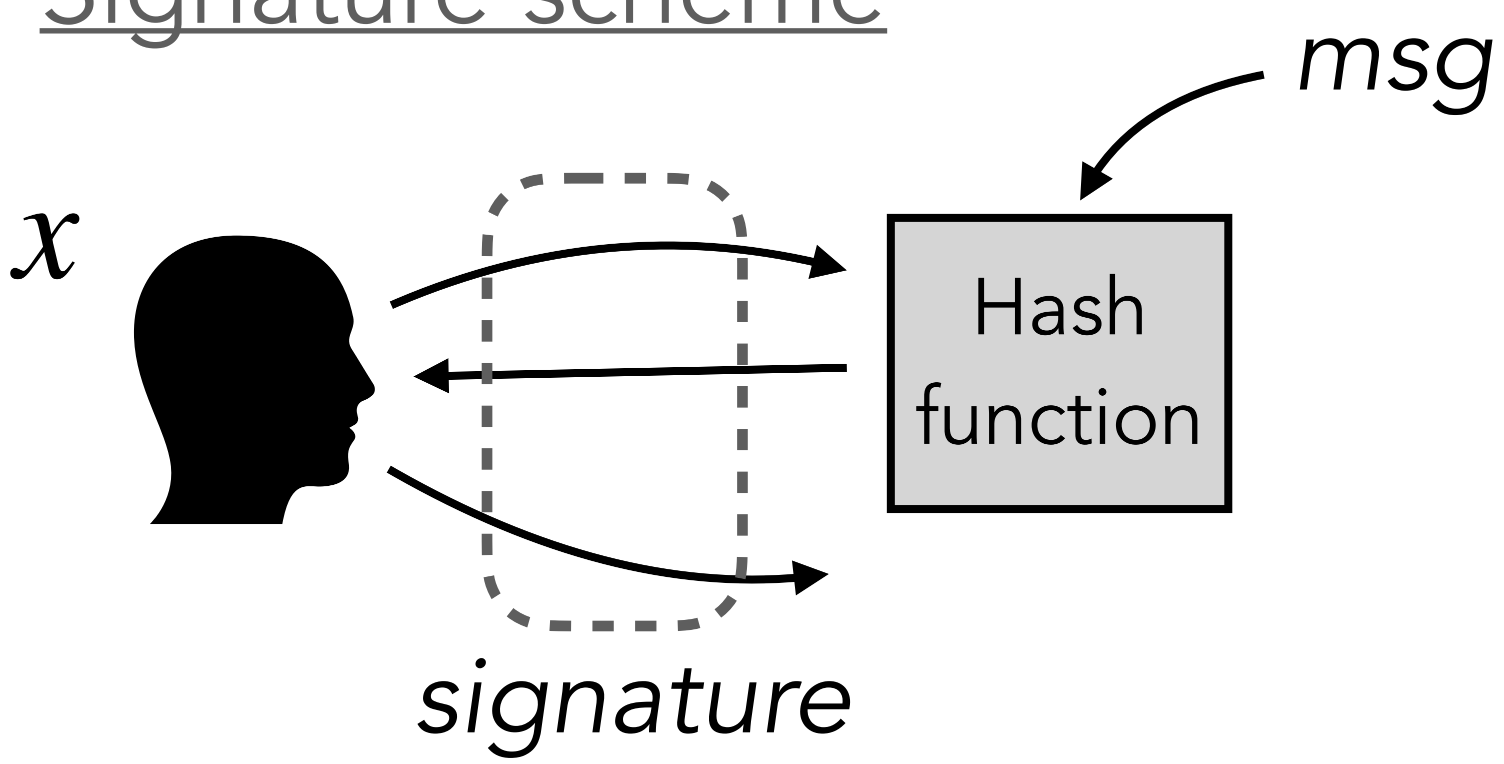

One-way function
$F : x \mapsto y$
E.g. AES, MQ system,
Syndrome decoding
Multiparty computation (MPC)
Input sharing $[\![x]\!]$
Joint evaluation of:
$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$
Zero-knowledge proof
x
y
Prover
Verifier
OK you know x
Signature scheme
x
msg
Hash function
signature

One-way function

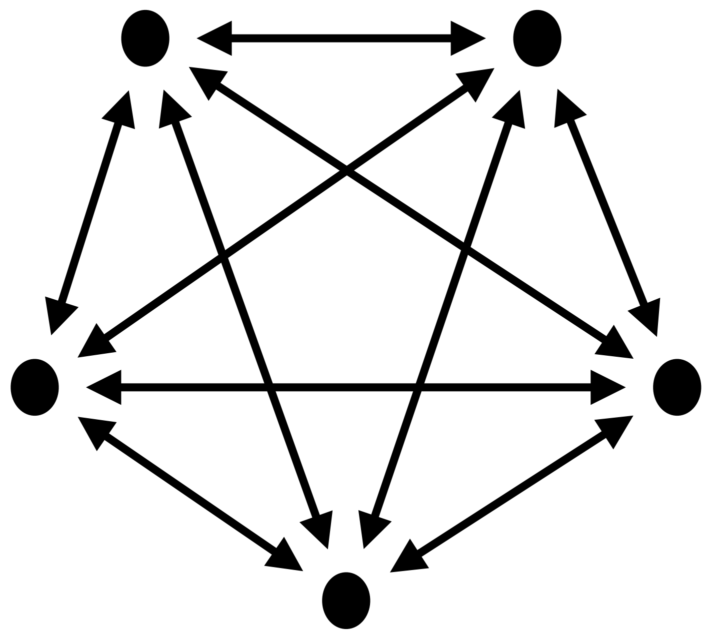
$$F : x \mapsto y$$

E.g. AES, MQ system,
Syndrome decoding

Signature scheme



Multiparty computation (MPC)



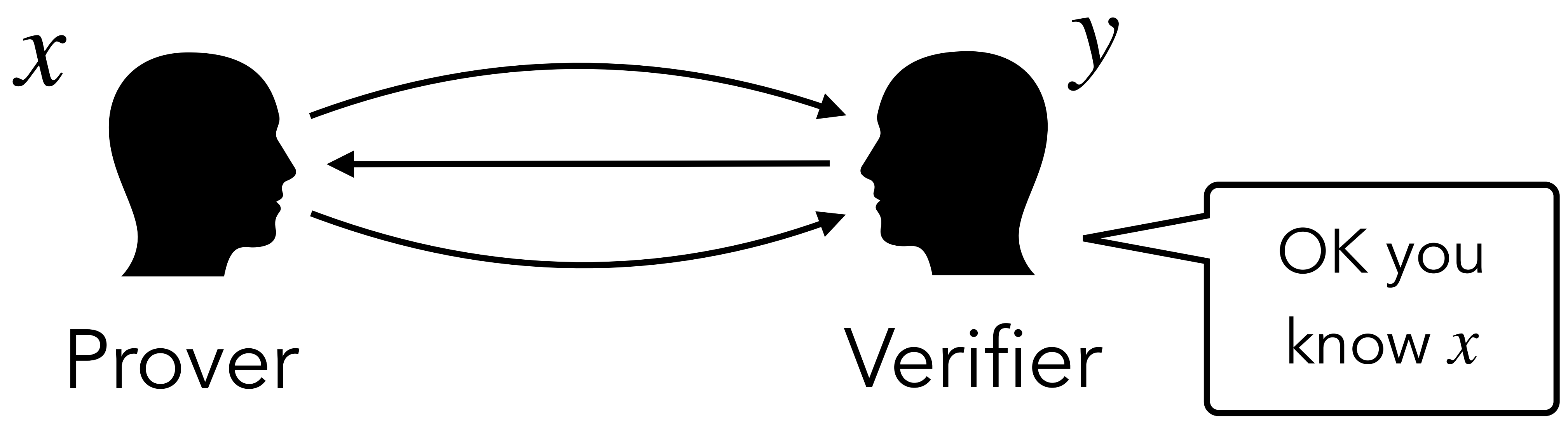
Input sharing $[\![x]\!]$

Joint evaluation of:

$$g(x) = \begin{cases} \text{Accept} & \text{if } F(x) = y \\ \text{Reject} & \text{if } F(x) \neq y \end{cases}$$

MPC in the Head

Zero-knowledge proof



Brief History

- 2007: **[IKOS07]** Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai: "Zero-knowledge from secure multiparty computation" (STOC 2007)
- 2016: [GMO16] "**ZKBoo**: Faster Zero-Knowledge for Boolean Circuits" (Usenix 2016)
- 2017: **Picnic** submission to NIST
  - MPCitH applied to LowMC
- 2017 → today: Active area of research
  - Drastic improvements
  - Application to various PQ problems
- 2023: **NIST call for additional PQ signatures**
  - 7 (to 9) MPCitH schemes / 40 round-1 candidates

Some Numbers

	Assumption	|pk|	|sig|	|pk| + |sig|	Sign	Verify
RSA	Factorisation	272 B	256 B	528 B	27 Mc	45 kc
EdDSA	Discret Log	32 B	64 B	96 B	42 kc	130 kc
Dilithium	Structured Lattice	1 312 B	2 420 B	3 732 B	333 kc	118 kc
Falcon	Structured Lattice	897 B	666 B	1 563 B	1.0 Mc	81 kc
SPHINCS+	Hash	32 B	7 856 B	7 888 B	4 682 Mc	4.7 Mc
			17 088 B	17 120 B	239 Mc	12.9 Mc

Some Numbers

	Assumption	*|pk|*	*|sig|*	*|pk| + |sig|*	*Sign*	*Verify*
RSA	**Factorisation**	272 B	256 B	528 B	27 Mc	45 kc
EdDSA	**Discret Log**	32 B	64 B	96 B	42 kc	130 kc
Dilithium	**Structured Lattice**	1 312 B	2 420 B	3 732 B	333 kc	118 kc
Falcon	**Structured Lattice**	897 B	666 B	1 563 B	1.0 Mc	81 kc
SPHINCS+	Hash	32 B	**7 856 B**	**7 888 B**	**4 682 Mc**	**4.7 Mc**
			17 088 B	**17 120 B**	**239 Mc**	**12.9 Mc**

MPCitH	Conservative / unstructured assumptions					

Some Numbers

	Assumption	*|pk|*	*|sig|*	*|pk| + |sig|*	*Sign*	*Verify*
RSA	**Factorisation**	272 B	256 B	528 B	**27 Mc**	45 kc
EdDSA	**Discret Log**	32 B	64 B	96 B	42 kc	130 kc
Dilithium	**Structured Lattice**	1 312 B	2 420 B	3 732 B	333 kc	118 kc
Falcon	**Structured Lattice**	897 B	666 B	1 563 B	1.0 Mc	81 kc
SPHINCS+	Hash	32 B	**7 856 B**	**7 888 B**	**4 682 Mc**	**4.7 Mc**
			17 088 B	**17 120 B**	**239 Mc**	**12.9 Mc**

MPCitH	Conservative / unstructured assumptions	Small: 32 B - ~ 100 B				

Some Numbers

	Assumption	*|pk|*	*|sig|*	*|pk| + |sig|*	*Sign*	*Verify*
RSA	**Factorisation**	272 B	256 B	528 B	**27 Mc**	45 kc
EdDSA	**Discret Log**	32 B	64 B	96 B	42 kc	130 kc
Dilithium	**Structured Lattice**	1 312 B	2 420 B	3 732 B	333 kc	118 kc
Falcon	**Structured Lattice**	897 B	666 B	1 563 B	1.0 Mc	81 kc
SPHINCS+	Hash	32 B	**7 856 B**	**7 888 B**	**4 682 Mc**	**4.7 Mc**
			17 088 B	**17 120 B**	**239 Mc**	**12.9 Mc**

MPCitH	Conservative / unstructured assumptions	Small: 32 B - ~ 100 B	(typically) 5-10 kB			

Some Numbers

	Assumption	|pk|	|sig|	*|pk| + |sig|*	*Sign*	*Verify*
RSA	**Factorisation**	272 B	256 B	528 B	27 Mc	45 kc
EdDSA	**Discret Log**	32 B	64 B	96 B	42 kc	130 kc
Dilithium	**Structured Lattice**	1 312 B	2 420 B	3 732 B	333 kc	118 kc
Falcon	**Structured Lattice**	897 B	666 B	1 563 B	1.0 Mc	81 kc
SPHINCS+	Hash	32 B	**7 856 B**	**7 888 B**	**4 682 Mc**	**4.7 Mc**
			17 088 B	**17 120 B**	**239 Mc**	**12.9 Mc**
MPCitH	Conservative / unstructured assumptions	Small: 32 B - ~ 100 B	(typically) 5-10 kB (recently) AES: 4-6 kB MQ: 2.5-3 kB Rank: ~3 kB			

Some Numbers

	Assumption	|pk|	|sig|	|pk| + |sig|	*Sign*	*Verify*
RSA	**Factorisation**	272 B	256 B	528 B	27 Mc	45 kc
EdDSA	**Discret Log**	32 B	64 B	96 B	42 kc	130 kc
Dilithium	**Structured Lattice**	1 312 B	2 420 B	3 732 B	333 kc	118 kc
Falcon	**Structured Lattice**	897 B	666 B	1 563 B	1.0 Mc	81 kc
SPHINCS+	Hash	32 B	**7 856 B**	**7 888 B**	**4 682 Mc**	**4.7 Mc**
			17 088 B	**17 120 B**	**239 Mc**	**12.9 Mc**
MPCitH	Conservative / unstructured assumptions	Small: 32 B - ~ 100 B	(typically) 5-10 kB (recently) AES: 4-6 kB MQ: 2.5-3 kB Rank: ~3 kB	~ same as |sig|		

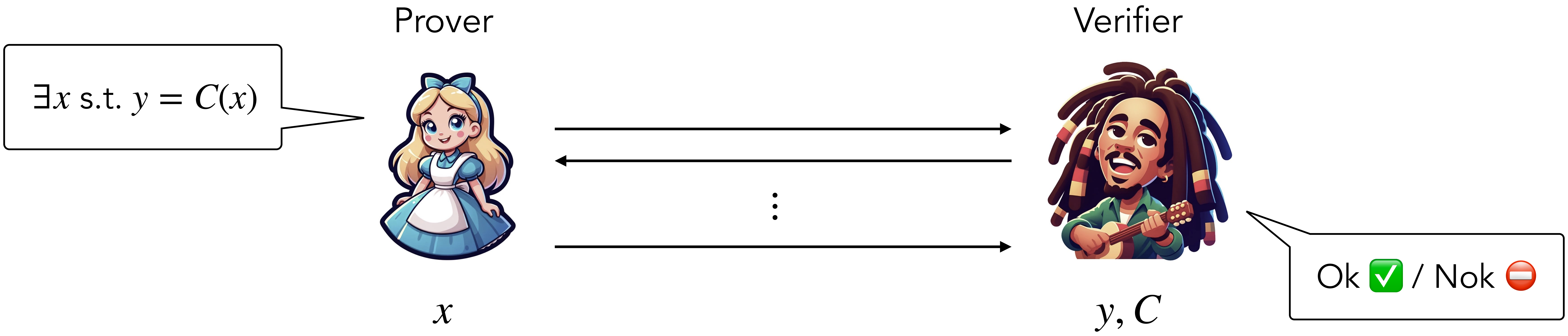
Some Numbers

	Assumption	|pk|	|sig|	|pk| + |sig|	Sign	Verify
RSA	Factorisation	272 B	256 B	528 B	27 Mc	45 kc
EdDSA	Discret Log	32 B	64 B	96 B	42 kc	130 kc
Dilithium	Structured Lattice	1 312 B	2 420 B	3 732 B	333 kc	118 kc
Falcon	Structured Lattice	897 B	666 B	1 563 B	1.0 Mc	81 kc
SPHINCS+	Hash	32 B	7 856 B	7 888 B	4 682 Mc	4.7 Mc
			17 088 B	17 120 B	239 Mc	12.9 Mc
MPCitH	Conservative / unstructured assumptions	Small: 32 B - ~ 100 B	(typically) 5-10 kB (recently) AES: 4-6 kB MQ: 2.5-3 kB Rank: ~3 kB	~ same as |sig|	(typically) ~10-50 Mc	(typically) same as sign

Introduction to Zero-Knowledge Proofs

Interactive Proof

Interactive Proof



Interactive Proof

$\exists x \text{ s.t. } y = C(x)$

Prover

x

Verifier

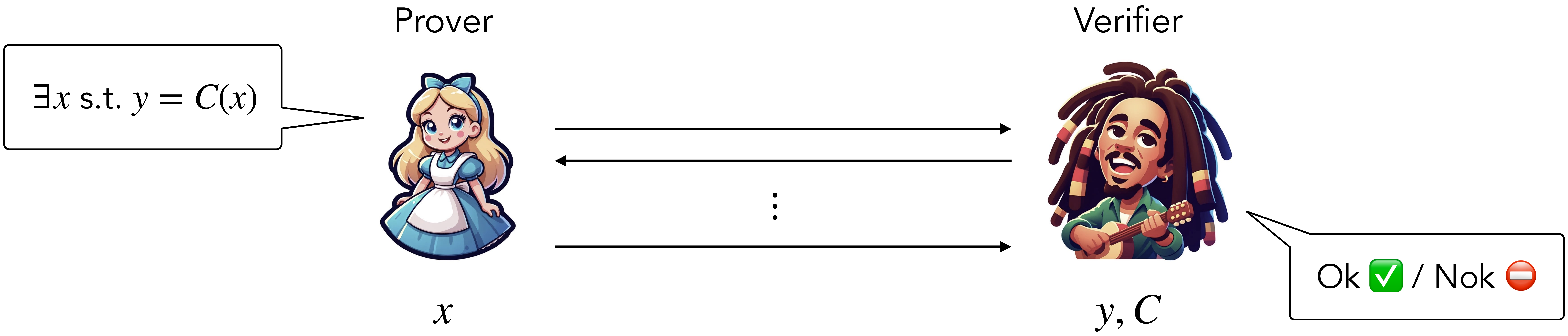
y, C

Ok ✅ / Nok ⛔

Completeness

$$P\left[\ ✅ \mid \exists\, x \text{ s.t. } C(x) = y \right] = 1$$

Interactive Proof



Completeness

$$P[\,✅ \mid \exists\, x \text{ s.t. } C(x) = y\,] = 1$$

Soundness

$$P[\,✅ \mid \nexists\, x \text{ s.t. } C(x) = y\,] \leq \varepsilon$$

Interactive Proof

$\exists x$ s.t. $y = C(x)$

Prover

x

$\vdots$

Verifier

y, C

Ok ✅ / Nok ⛔

Completeness

$P[\ ✅ \mid \exists\, x \text{ s.t. } C(x) = y\ ] \;=\; 1$

Soundness

$P[\ ✅ \mid \nexists\, x \text{ s.t. } C(x) = y\ ] \;\leq\; \varepsilon$

soundness error

Interactive Proof

Prover

$\exists x$ s.t. $y = C(x)$

Verifier

Ok ✅ / Nok ⛔

x

the witness

y, C

the statement

Completeness

$$P[\text{✅} \mid \exists\, x \text{ s.t. } C(x) = y] \;=\; 1$$

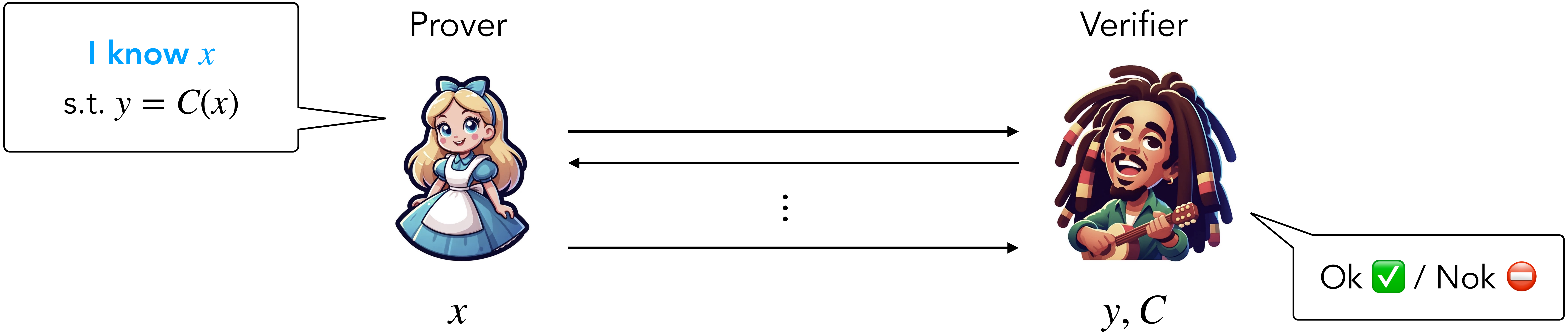
Soundness

$$P[\text{✅} \mid \nexists\, x \text{ s.t. } C(x) = y] \;\leq\; \varepsilon$$

soundness error

Proof of Knowledge

Proof of Knowledge



Proof of Knowledge

I know x

s.t. $y = C(x)$

Prover

x

Verifier

y, C

Ok ✅ / Nok ⛔

Knowledge Soundness (informal)

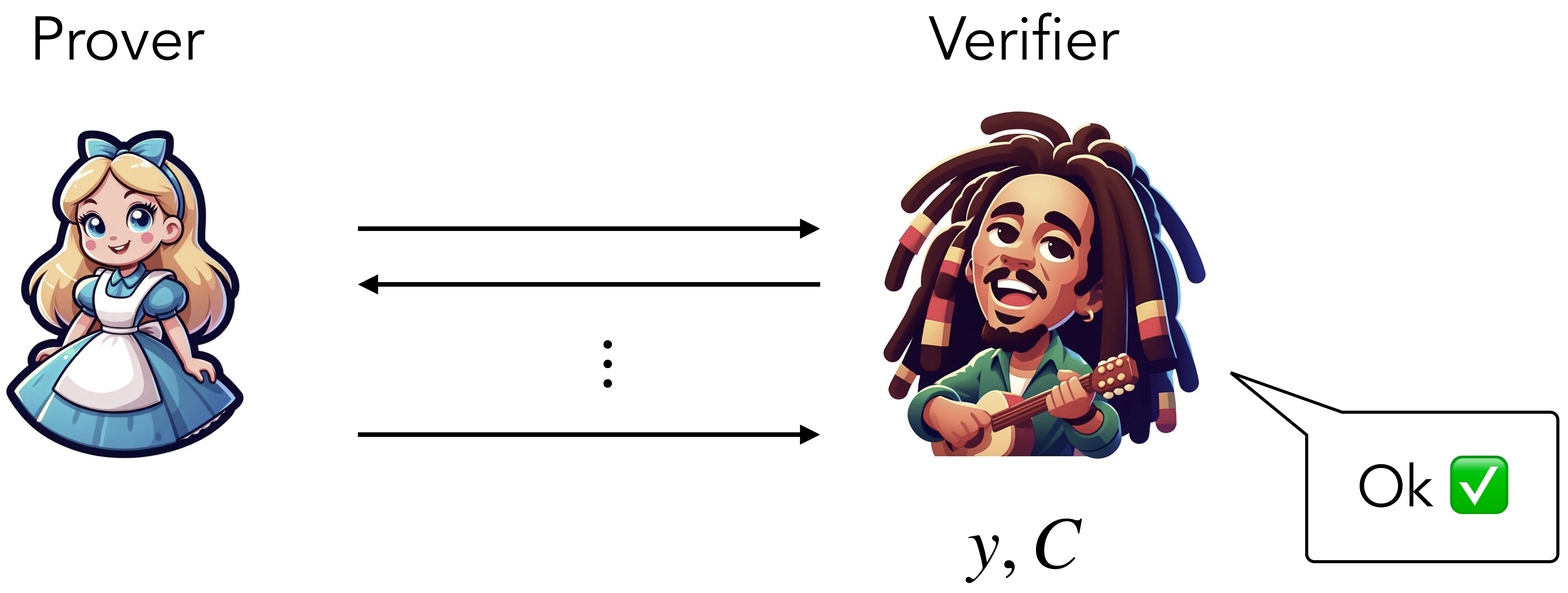
$P\big[$ ✅ |  **doesn't know** x s.t. $C(x) = y\big] \leq \varepsilon$

Knowledge Soundness

Knowledge Soundness

If ∃ Prover s.t.

$P[\text{ Verifier ✅ }] > \varepsilon$



Knowledge Soundness

If ∃ Prover s.t.
$P[$ Verifier ✅ $] > \varepsilon$

then ∃ Extractor
which recovers x

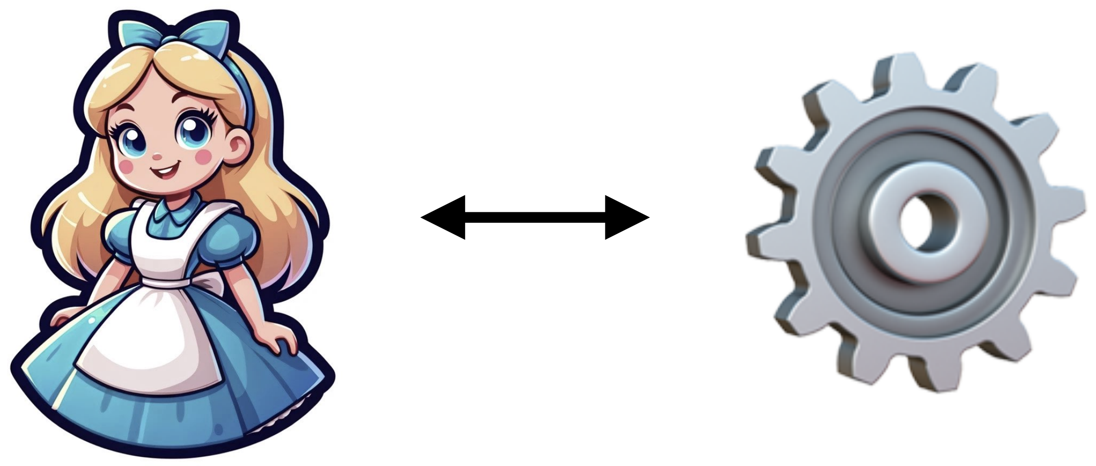
Prover

Verifier

y, C

Ok ✅

Extractor



x

Knowledge Soundness

If $\exists$ Prover s.t.

$P[\text{ Verifier ✅ }] > \varepsilon$

then $\exists$ Extractor

which recovers x

Prover

Verifier

Ok ✅

y, C

Extractor

x

Contraposition

If  doesn't know x (we cannot extract x)

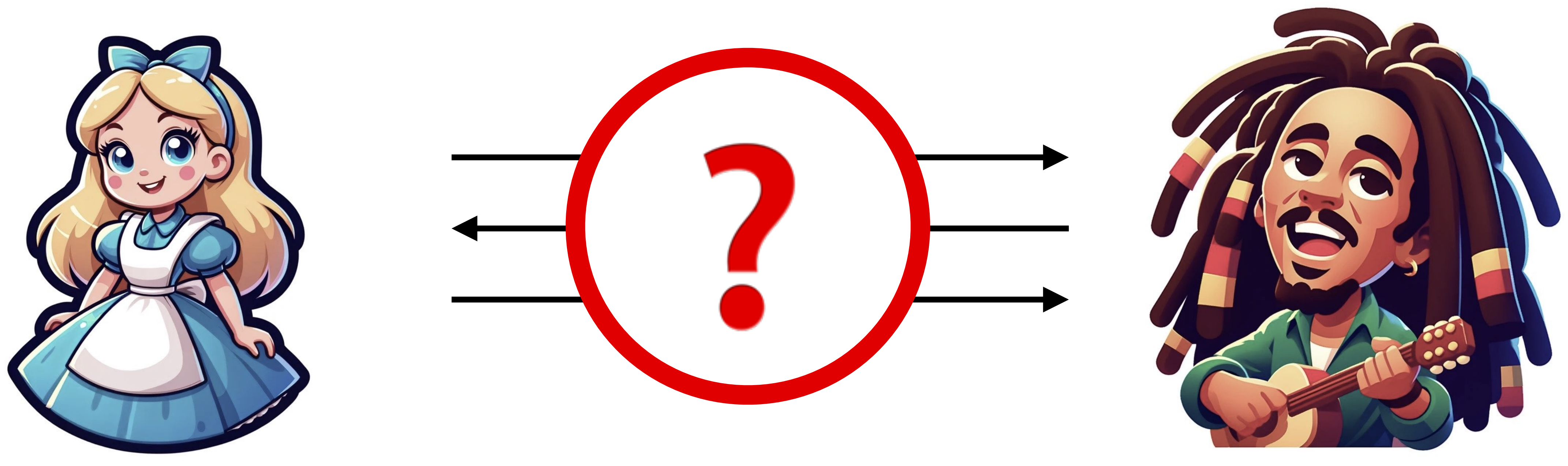
then $P[\text{ Verifier ✅ }] \leq \varepsilon$

Question 1

Question 1

Prover

Verifier



y

I know k s.t. $y = \mathrm{AES}_k(0)$

Question 1

Prover

$$\xrightarrow{k}$$

Verifier

y

I know k s.t. $y = \mathrm{AES}_k(0)$

Indeed, $y = \mathrm{AES}_k(0)$

Useful Proof of Knowledge

Prover | Verifier

x | y, C

Useful Proof of Knowledge

Prover

x

Verifier

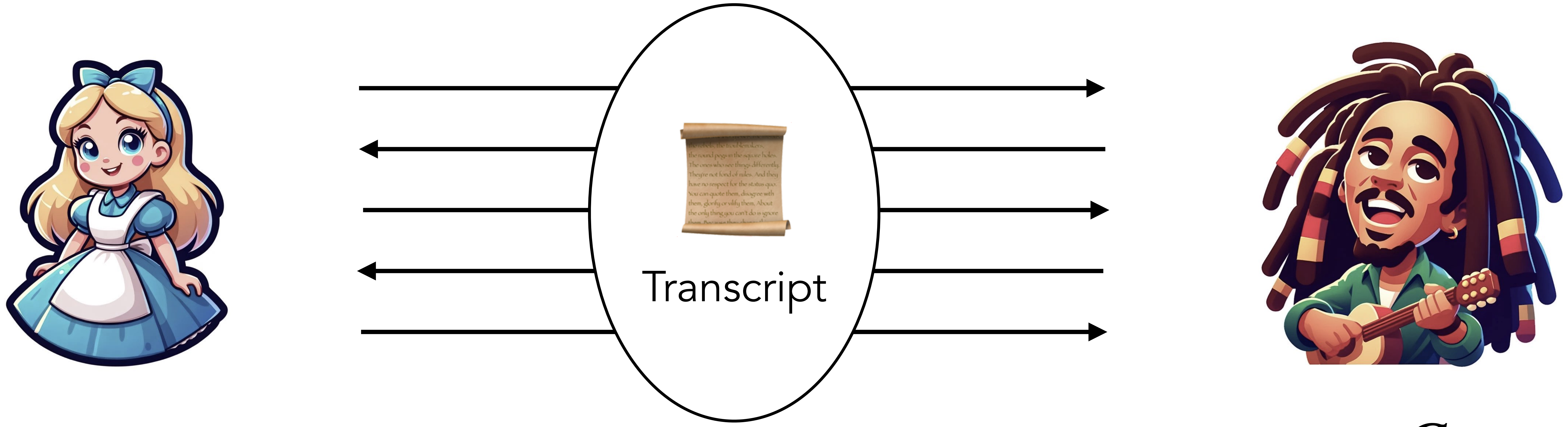
y, C

Zero Knowledge (informal)

 learns nothing about x

Useful Proof of Knowledge

Prover | Verifier

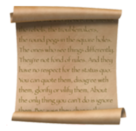


x | y, C

Zero Knowledge (informal)

 learns nothing about x

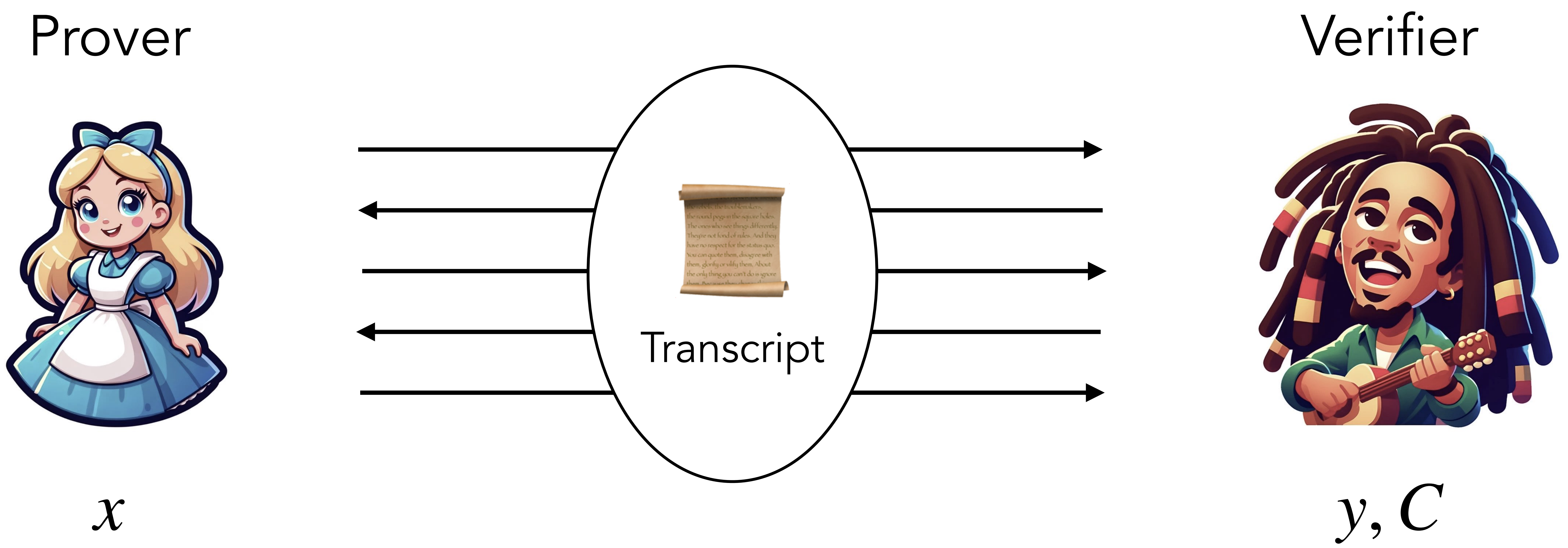
Succinctness (informal)

|  | $\ll |x|, |C|, |y|$

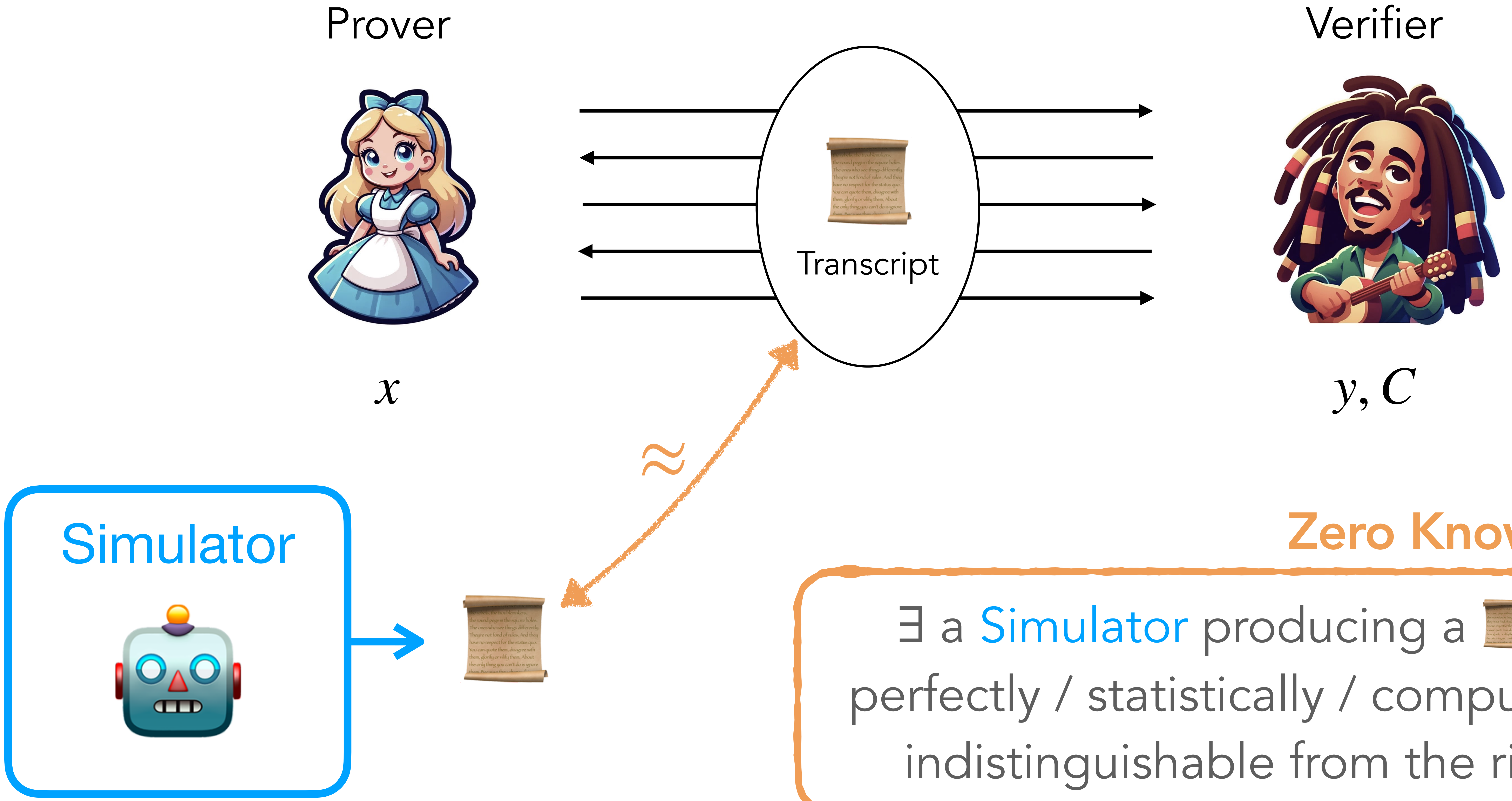
verif. time $\ll |x|, |C|, |y|$

Zero Knowledge Proof

Zero Knowledge Proof



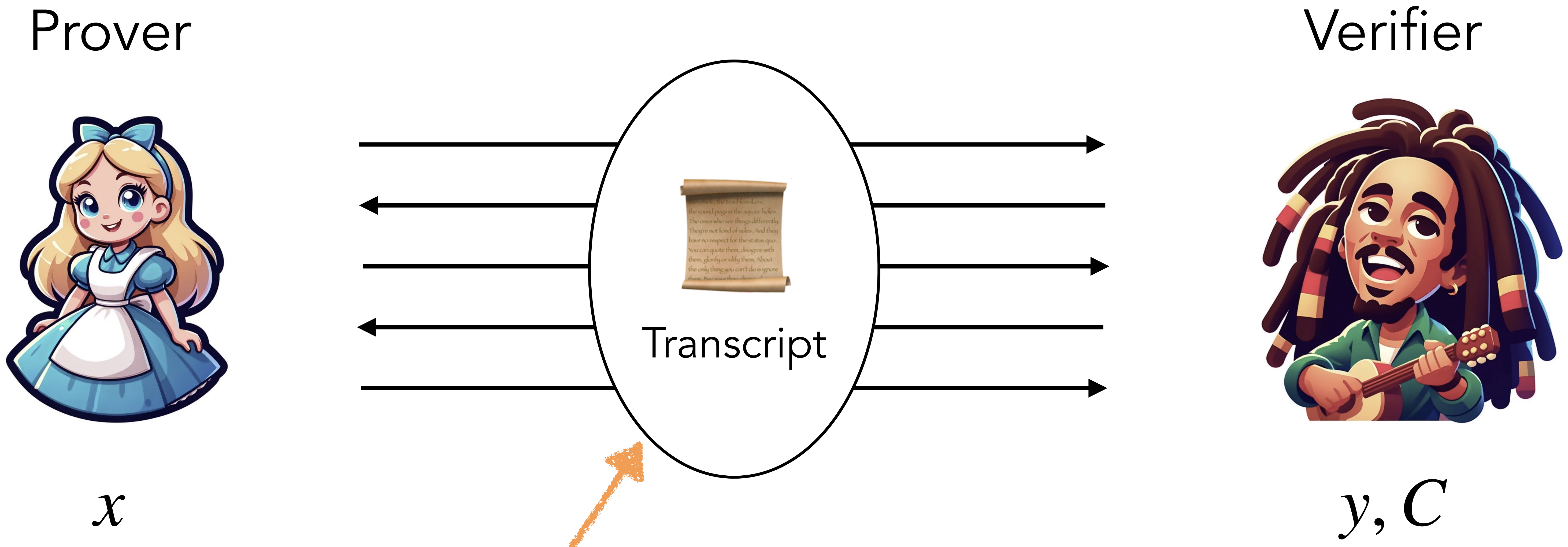
Zero Knowledge Proof

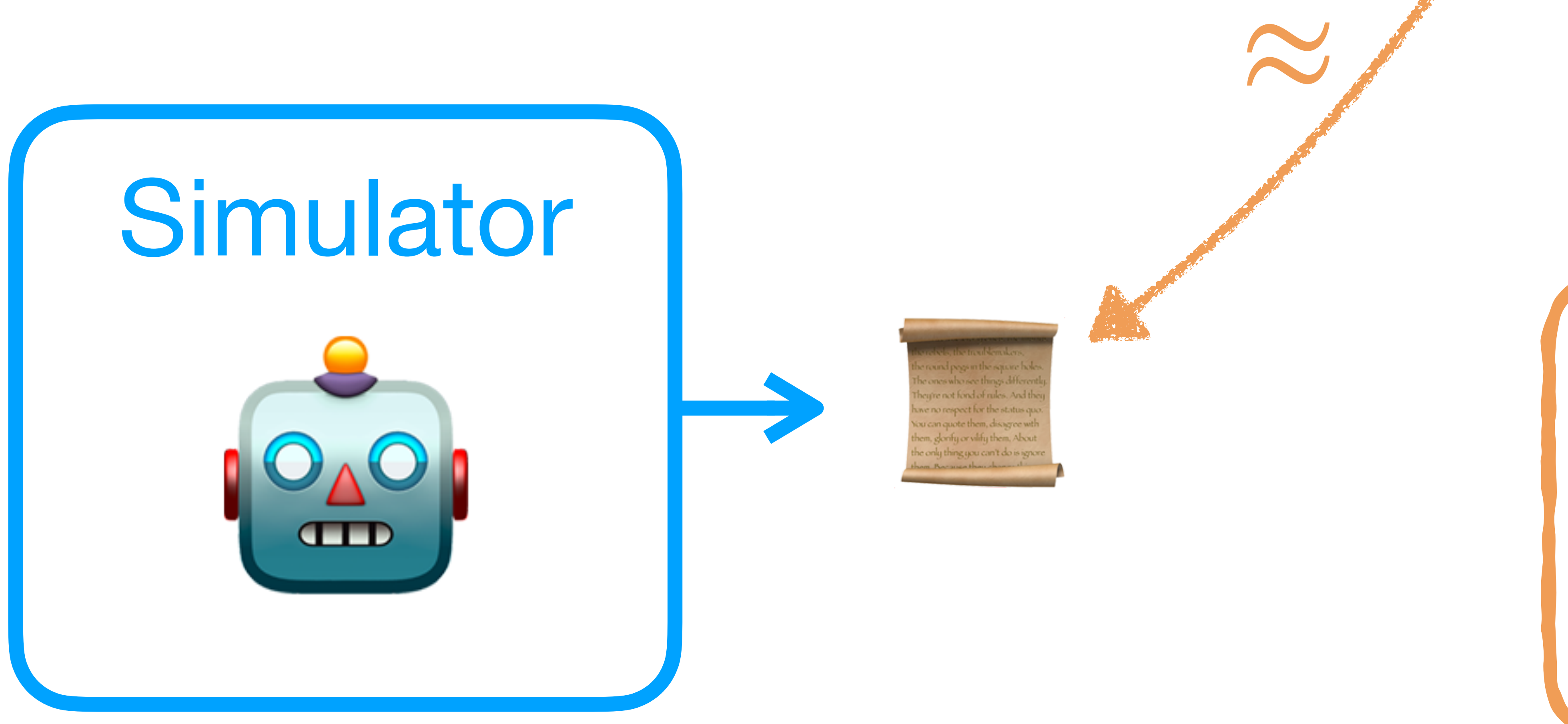


Zero Knowledge

∃ a Simulator producing a 📜 that is perfectly / statistically / computationally indistinguishable from the right 📜.

Zero Knowledge Proof





(Honest Verifier) Zero Knowledge

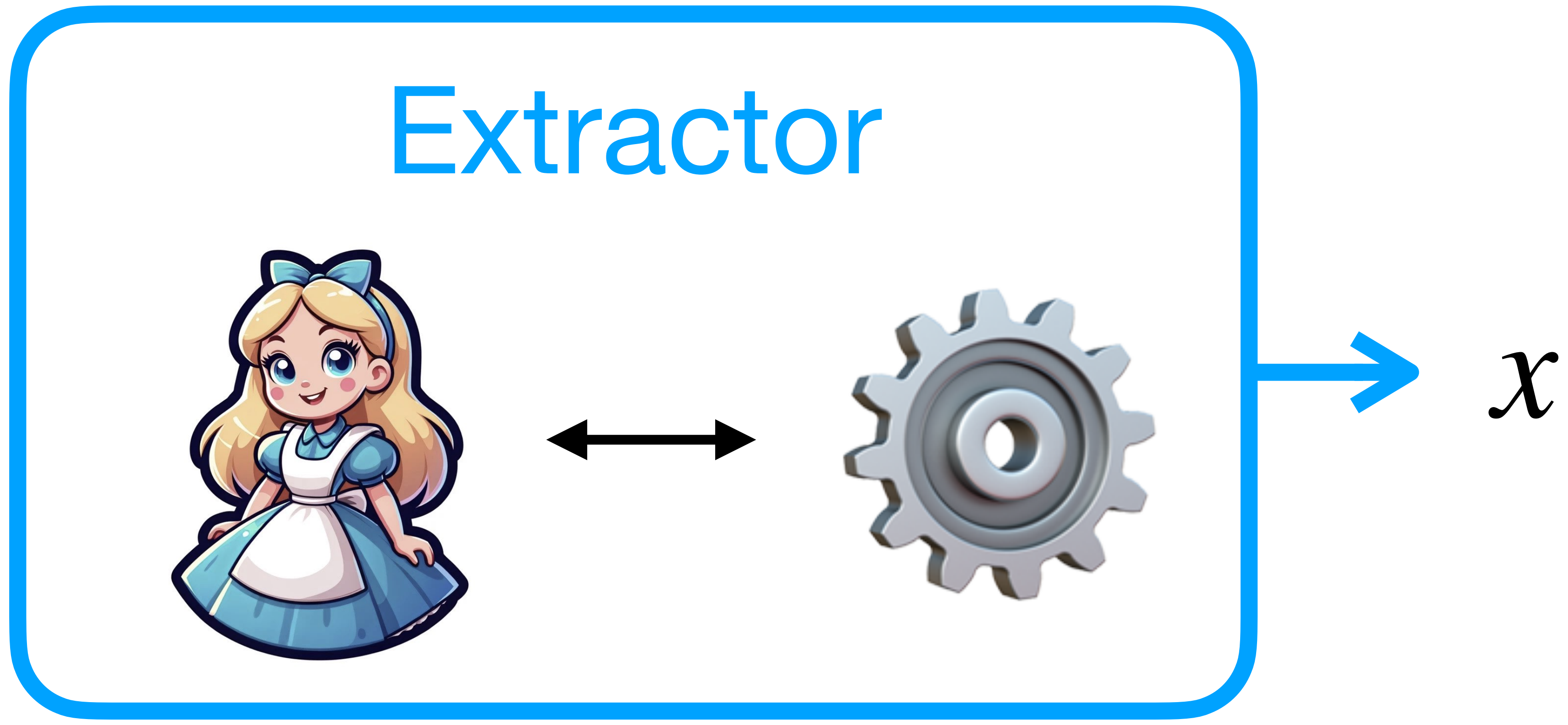
∃ a Simulator producing a 📜 that is perfectly / statistically / computationally indistinguishable from the right 📜.

Back to Knowledge Soundness

Knowledge Soundness

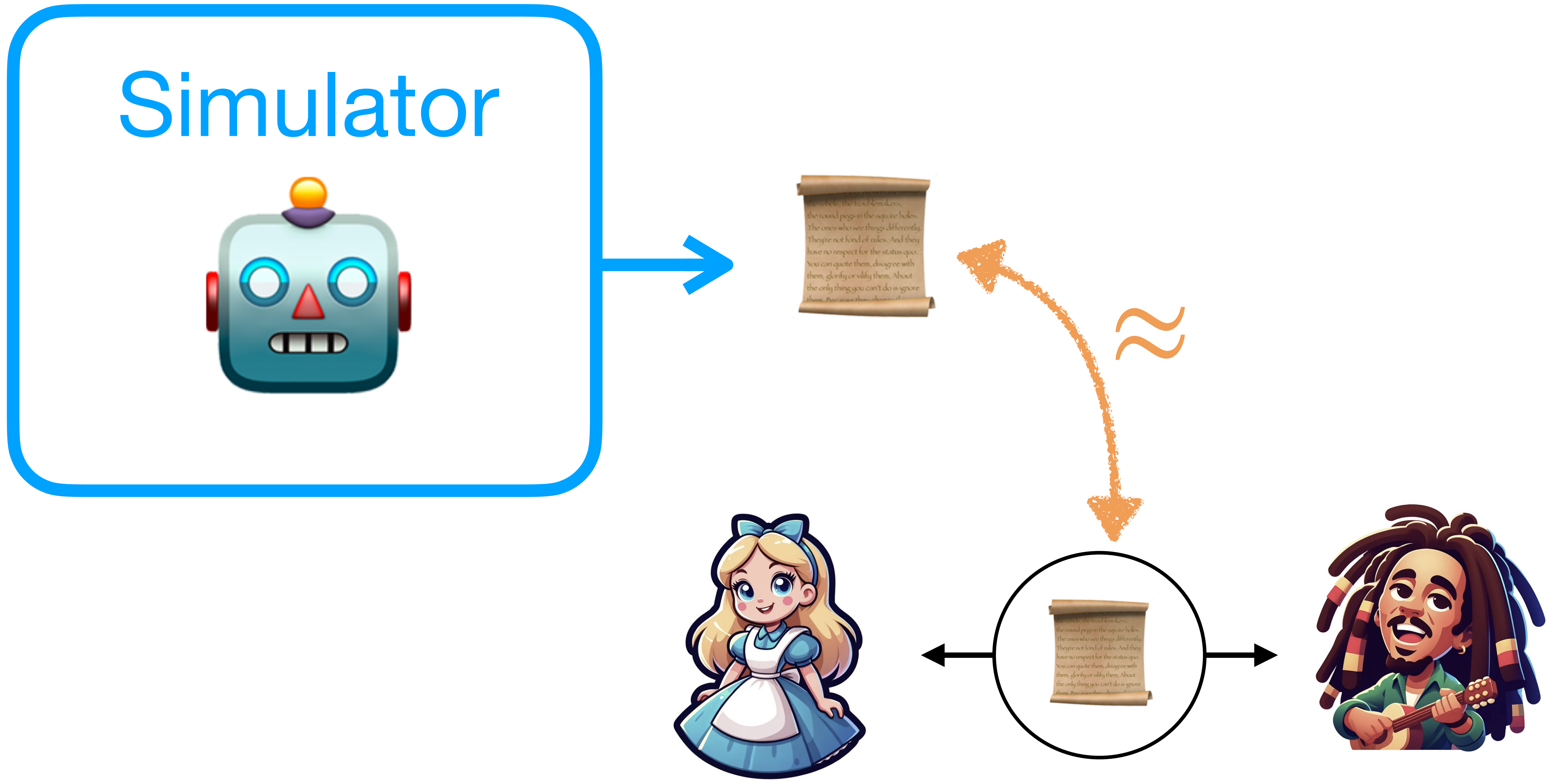
If ∃ Prover s.t. $P\Big[$ Verifier ✅ $\Big] > \varepsilon$

then ∃ Extractor which recovers x



Zero Knowledge

∃ Simulator producing genuine transcripts

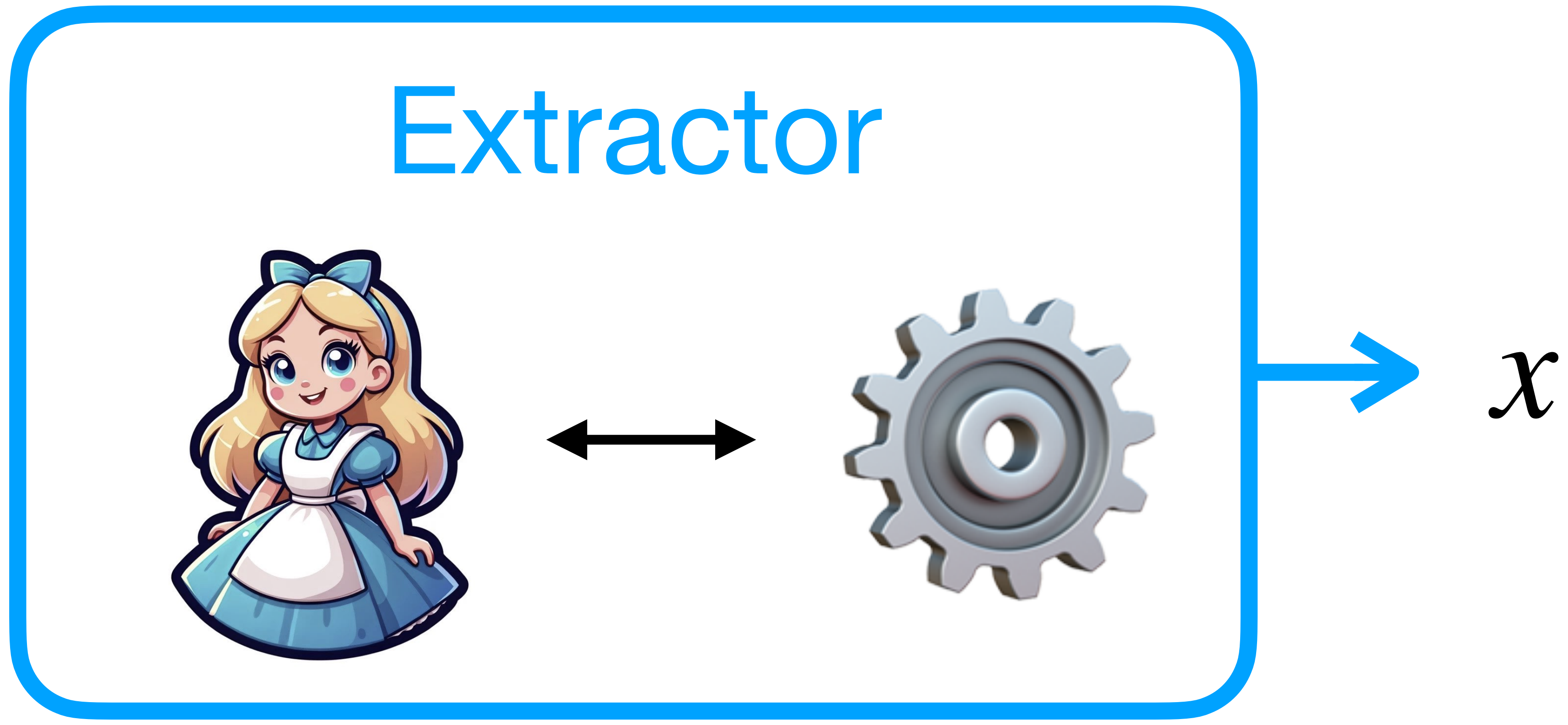


Back to Knowledge Soundness

Knowledge Soundness

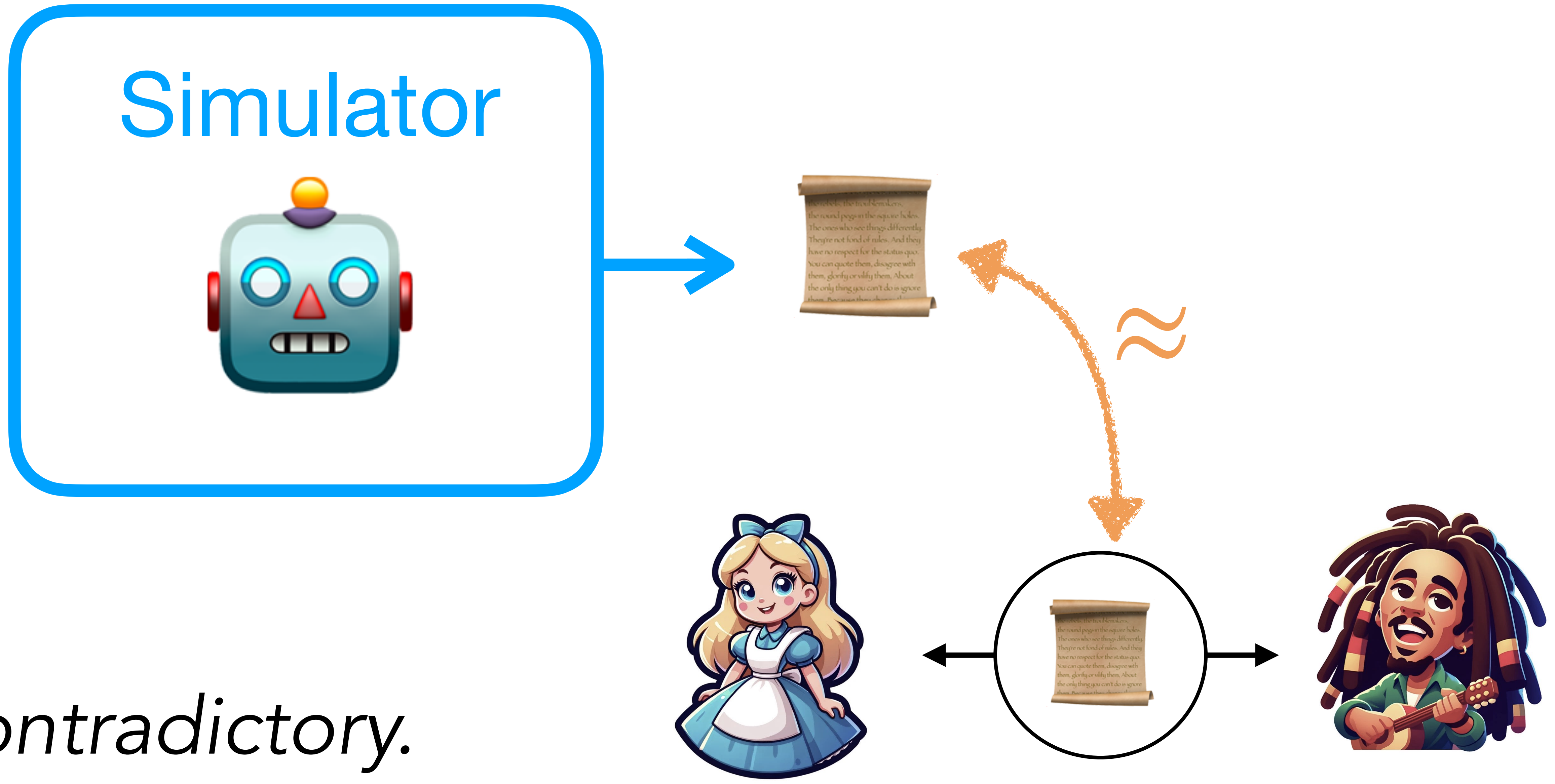
If ∃ Prover s.t. $P\big[$ Verifier ✅ $\big] > \varepsilon$

then ∃ Extractor which recovers x



Zero Knowledge

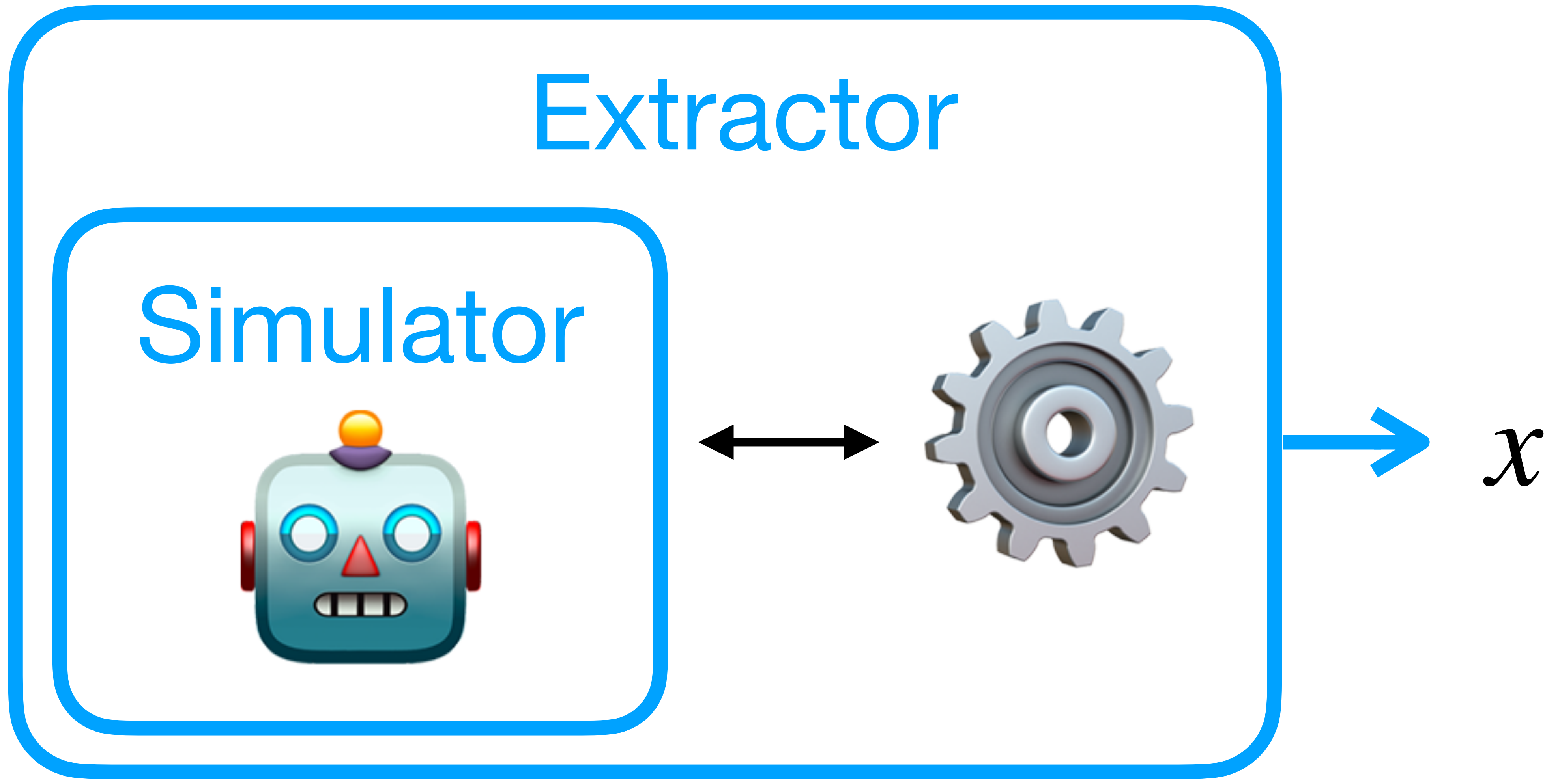
∃ Simulator producing genuine transcripts



🤔 *Those 2 seem somehow contradictory.*

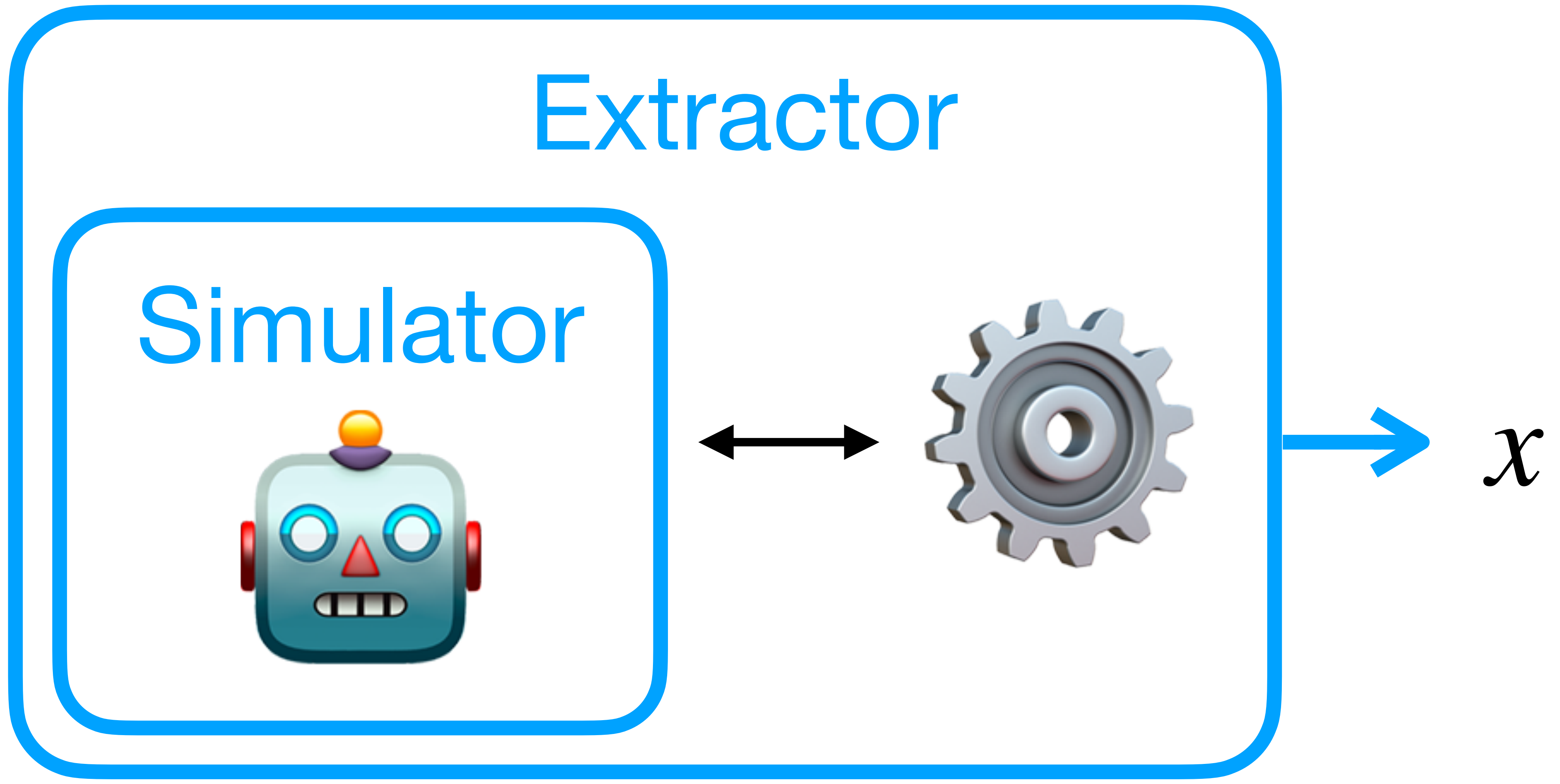
Question 2

Question 2



Q. Why this doesn't work?

Question 2



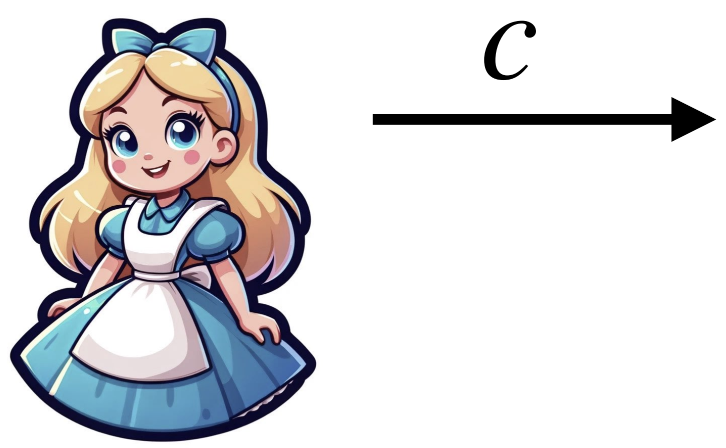
Q. Why this doesn't work?

A. Simulator only outputs 📜
Prover is stateful, it can be copied and forked.

Extraction

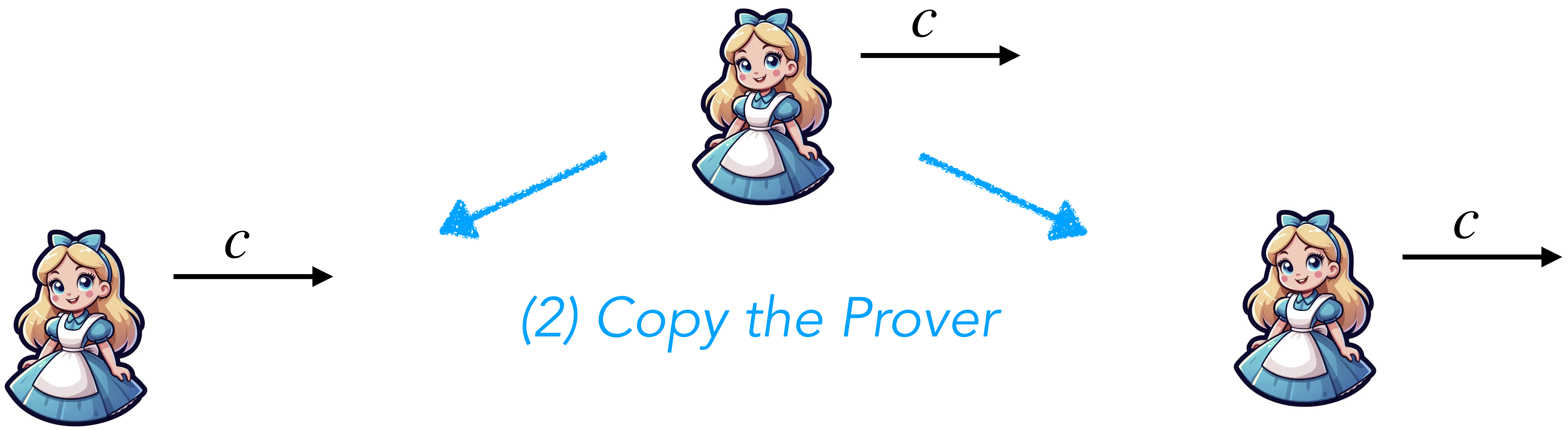
Extraction

(1) Start interaction

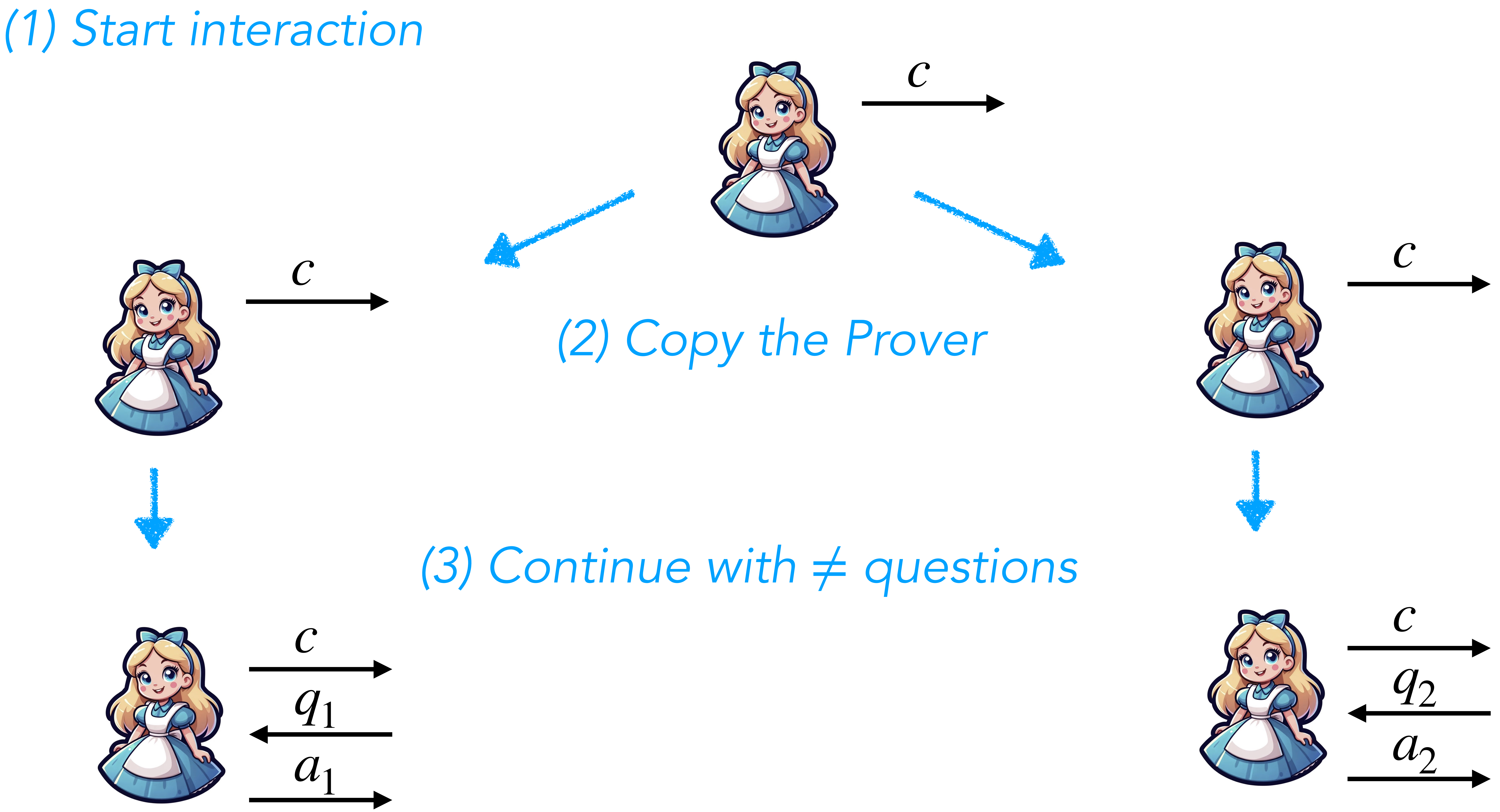


Extraction

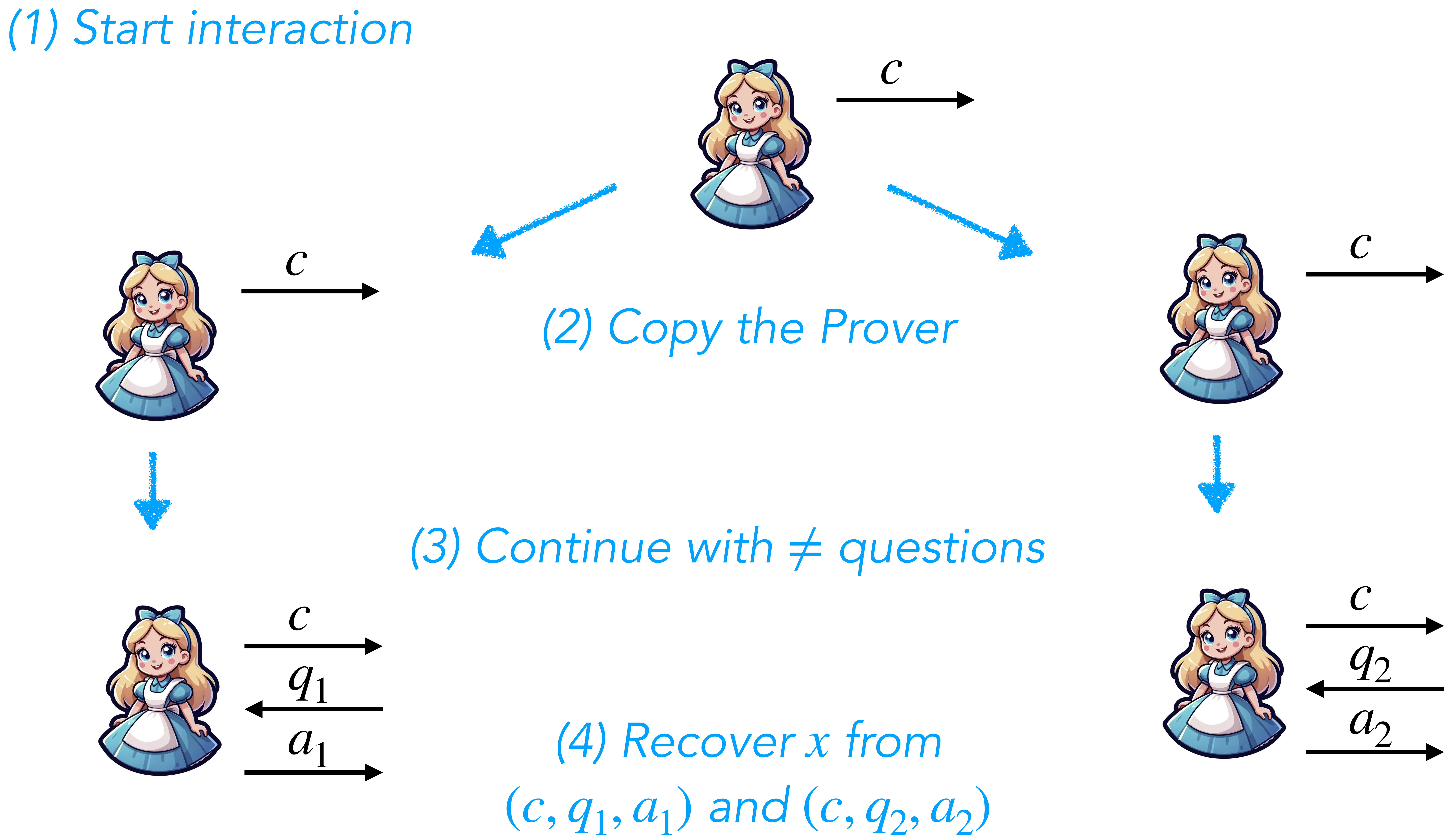
(1) Start interaction



Extraction


(1) Start interaction
c
c
(2) Copy the Prover
c
(3) Continue with $\neq$ questions
c
q_1
a_1
c
q_2
a_2

Extraction

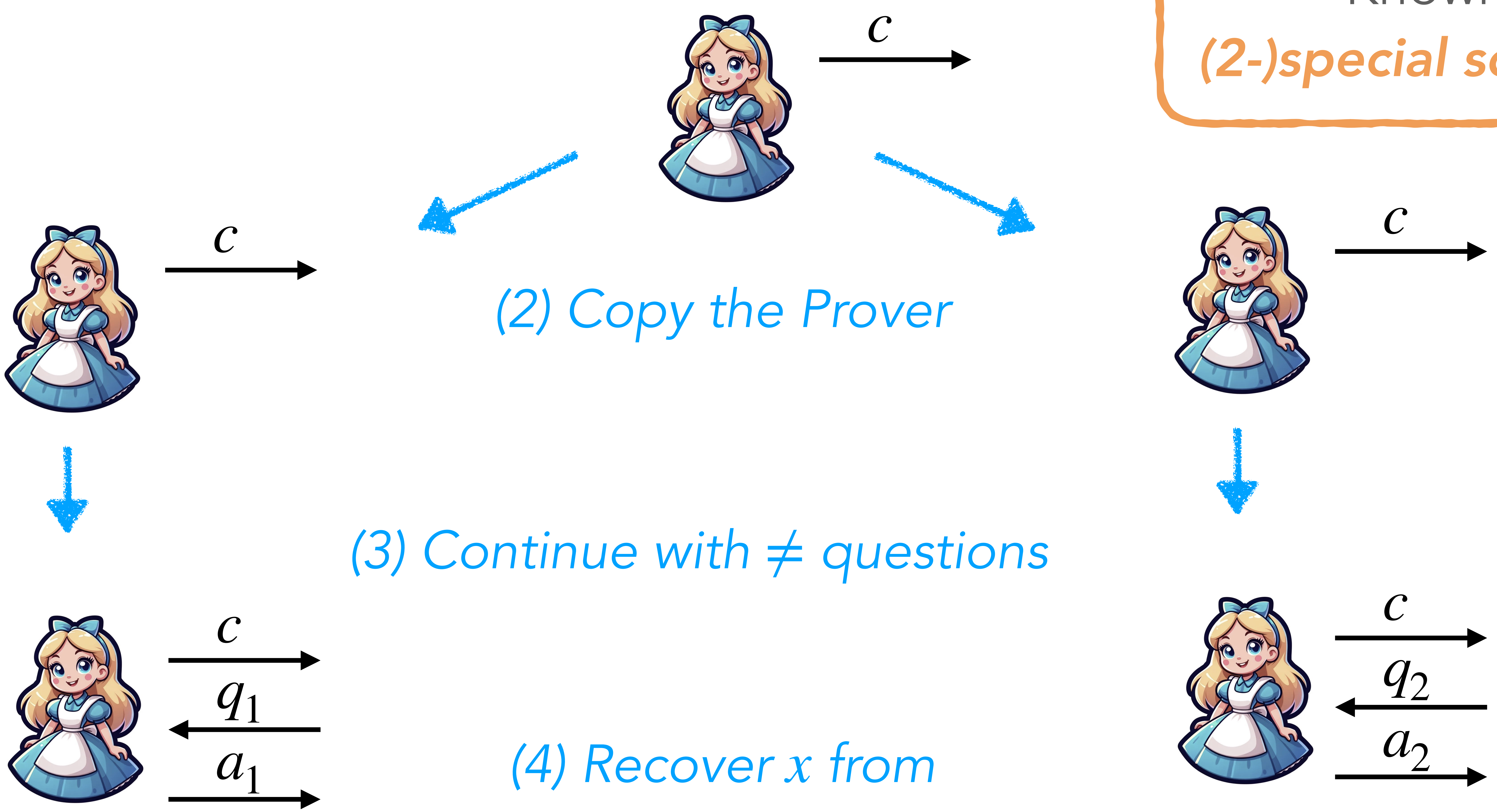

(1) Start interaction
c
(2) Copy the Prover
c
c
(3) Continue with $\neq$ questions
c
q_1
a_1
c
q_2
a_2
(4) Recover x from
(c, q_1, a_1) and (c, q_2, a_2)

Extraction

(1) Start interaction

Known as

(2-)special soundness



(2) Copy the Prover

(3) Continue with $\neq$ questions

(4) Recover x from

(c, q_1, a_1) and (c, q_2, a_2)

(Pre-Quantum) Example: Schnorr Protocol

Prover

x

Verifier

$y = g^x$

(Pre-Quantum) Example: Schnorr Protocol

Prover

x

$k \leftarrow \$$

$c = g^k$

$\xrightarrow{\quad c \quad}$

Verifier

$y = g^x$

(Pre-Quantum) Example: Schnorr Protocol

Prover

x

$k \leftarrow \$$

$c = g^k$

c →

← q

$q \leftarrow \$$

Verifier

$y = g^x$

(Pre-Quantum) Example: Schnorr Protocol

Prover

x

$k \leftarrow \$$

$c = g^k$

$\xrightarrow{\quad c \quad}$

$\xleftarrow{\quad q \quad}$

$a = qx + k$

$\xrightarrow{\quad a \quad}$

Verifier

$y = g^x$

$q \leftarrow \$$

(Pre-Quantum) Example: Schnorr Protocol

Prover

x

Verifier

$y = g^x$

$k \leftarrow \$$

$c = g^k$ $\xrightarrow{\quad c \quad}$

$\xleftarrow{\quad q \quad}$ $q \leftarrow \$$

$a = qx + k$ $\xrightarrow{\quad a \quad}$

Check $g^a = y^q \cdot c$

Question 3

Question 3

Prover

x

Verifier

$y = g^x$

Prover			Verifier
$k \leftarrow \$$ $c = g^k$	$\xrightarrow{\;c\;}$		
	$\xleftarrow{\;q\;}$	$q \leftarrow \$$	
$a = qx + k$	$\xrightarrow{\;a\;}$		
		Check $g^a = y^q \cdot c$	

Q. Why is Schnorr protocol zero-knowledge?

Answer

Simulator 🤖

Prover

x

Verifier

$y = g^x$

$k \leftarrow \$$

$c = g^k$

$\xrightarrow{c}$

$\xleftarrow{q}$

$q \leftarrow \$$

$a = qx + k$

$\xrightarrow{a}$

Check $g^a = y^q \cdot c$

Answer

Simulator 🤖

$q \leftarrow \$$

$a \leftarrow \$$

$c = g^a / y^q$

(c, a, q)

Perfect zero-knowledge

Prover

x

$k \leftarrow \$$

$c = g^k$

$\xrightarrow{c}$

$\xleftarrow{q}$

$a = qx + k$

$\xrightarrow{a}$

Verifier

$y = g^x$

$q \leftarrow \$$

Check $g^a = y^q \cdot c$

Answer

Simulator

$q \leftarrow \$$

$a \leftarrow \$$

$c = g^a / y^q$

(c, a, q)

Perfect zero-knowledge

Prover				Verifier
	$k \leftarrow \$$ $c = g^k$	$\xrightarrow{c}$ (commitment)		
x		$\xleftarrow{q}$	$q \leftarrow \$$	$y = g^x$
	$a = qx + k$	$\xrightarrow{a}$		
			Check $g^a = y^q \cdot c$	

Knowing the question (a.k.a. challenge) before the commitment enables perfect simulation.

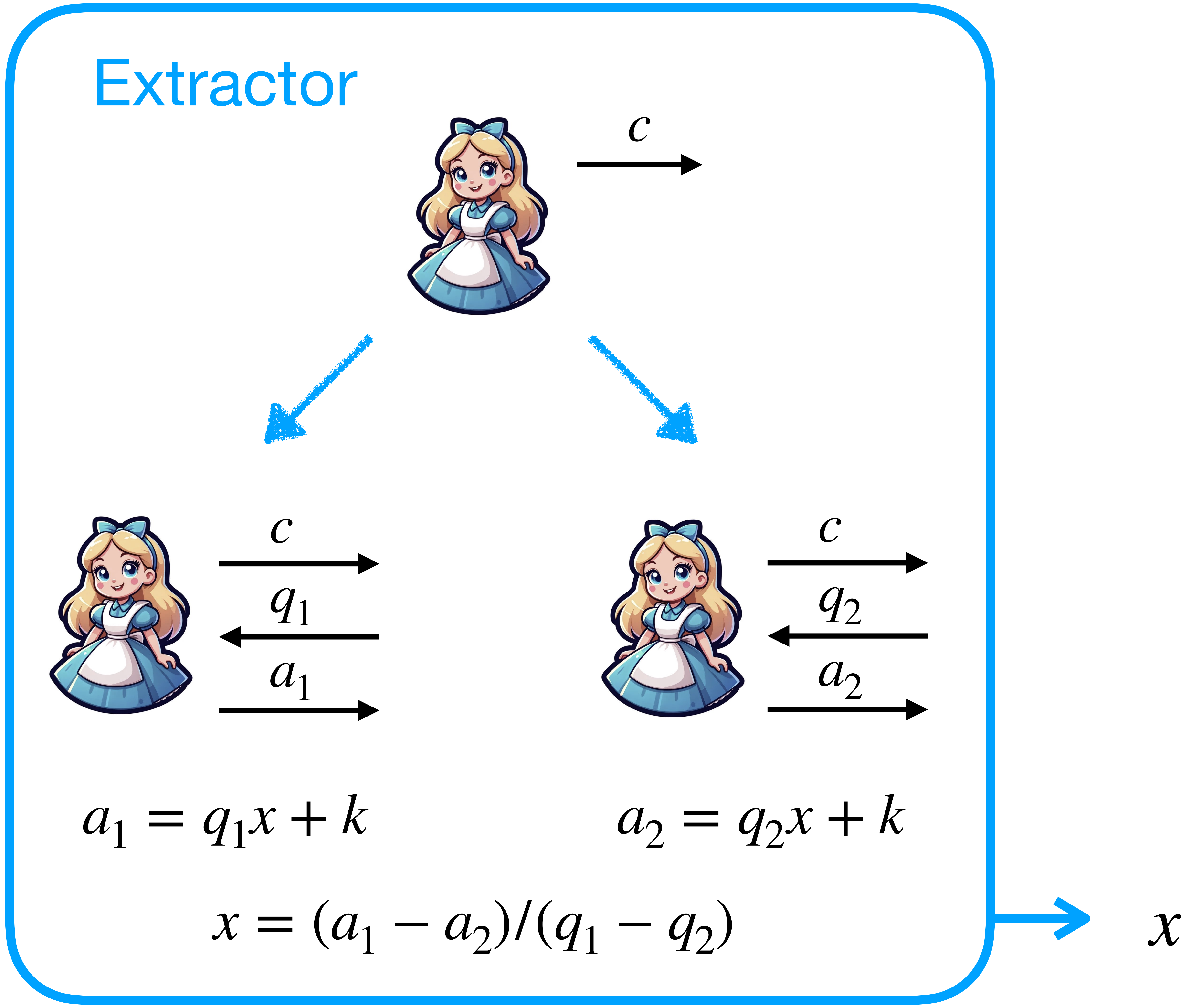
Question 4

Question 4

Prover			Verifier
	$k \leftarrow \$$ $c = g^k$	$\xrightarrow{c}$	
		$\xleftarrow{q}$	$q \leftarrow \$$
x			$y = g^x$
	$a = qx + k$	$\xrightarrow{a}$	
			Check $g^a = y^q \cdot c$

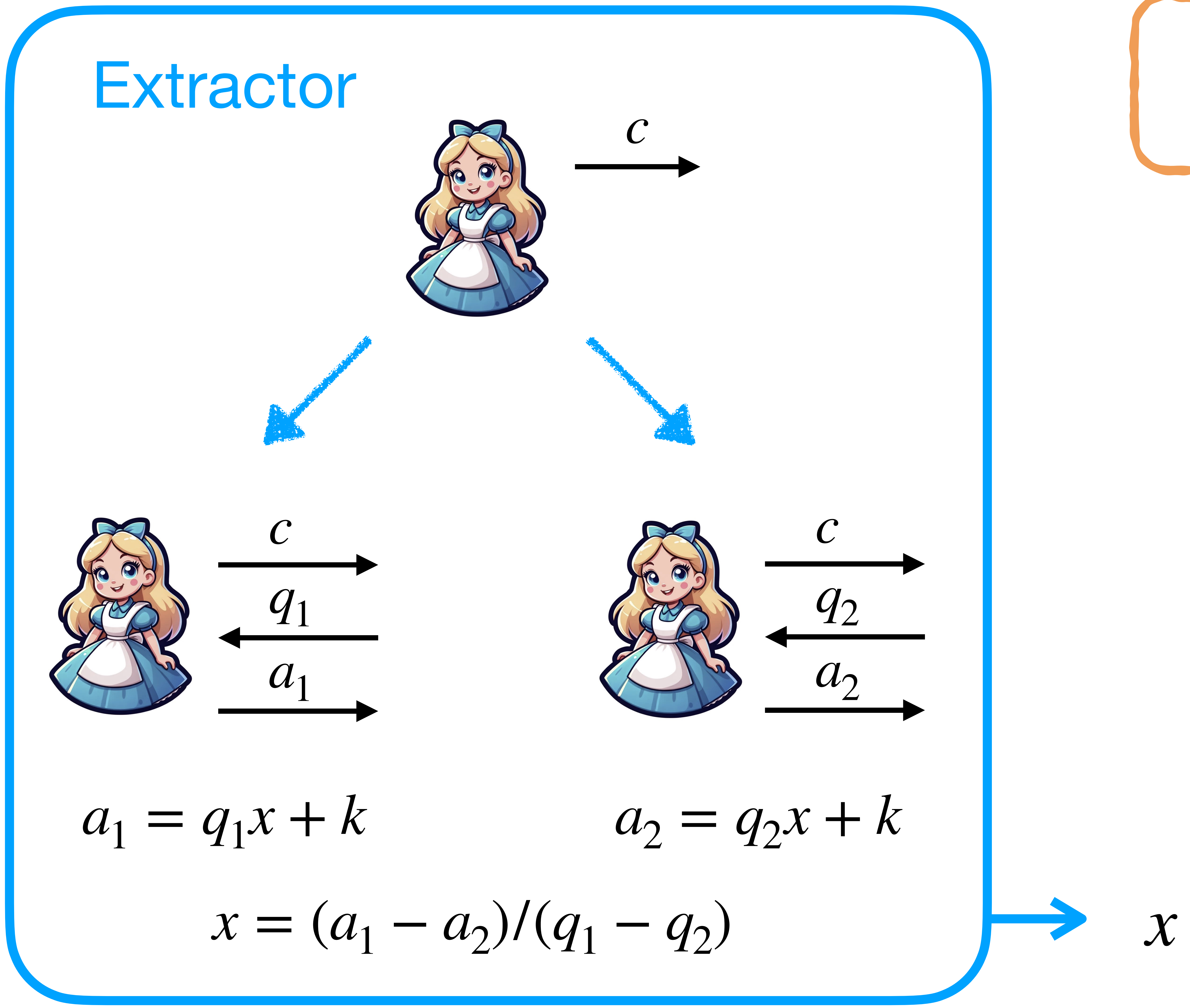
Q. Why is Schnorr protocol knowledge sound?

Answer

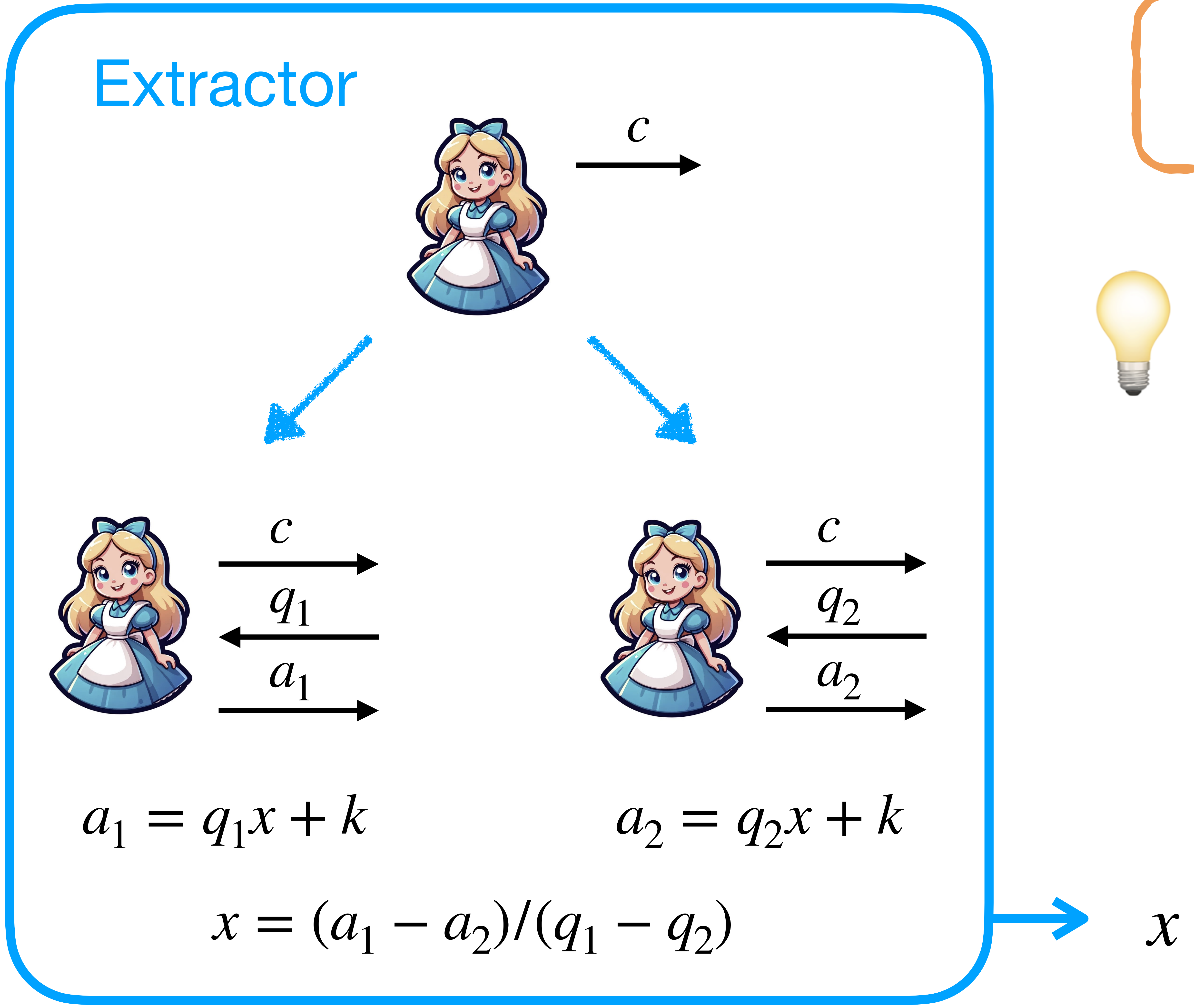


Answer

2-Special Knowledge Soundness



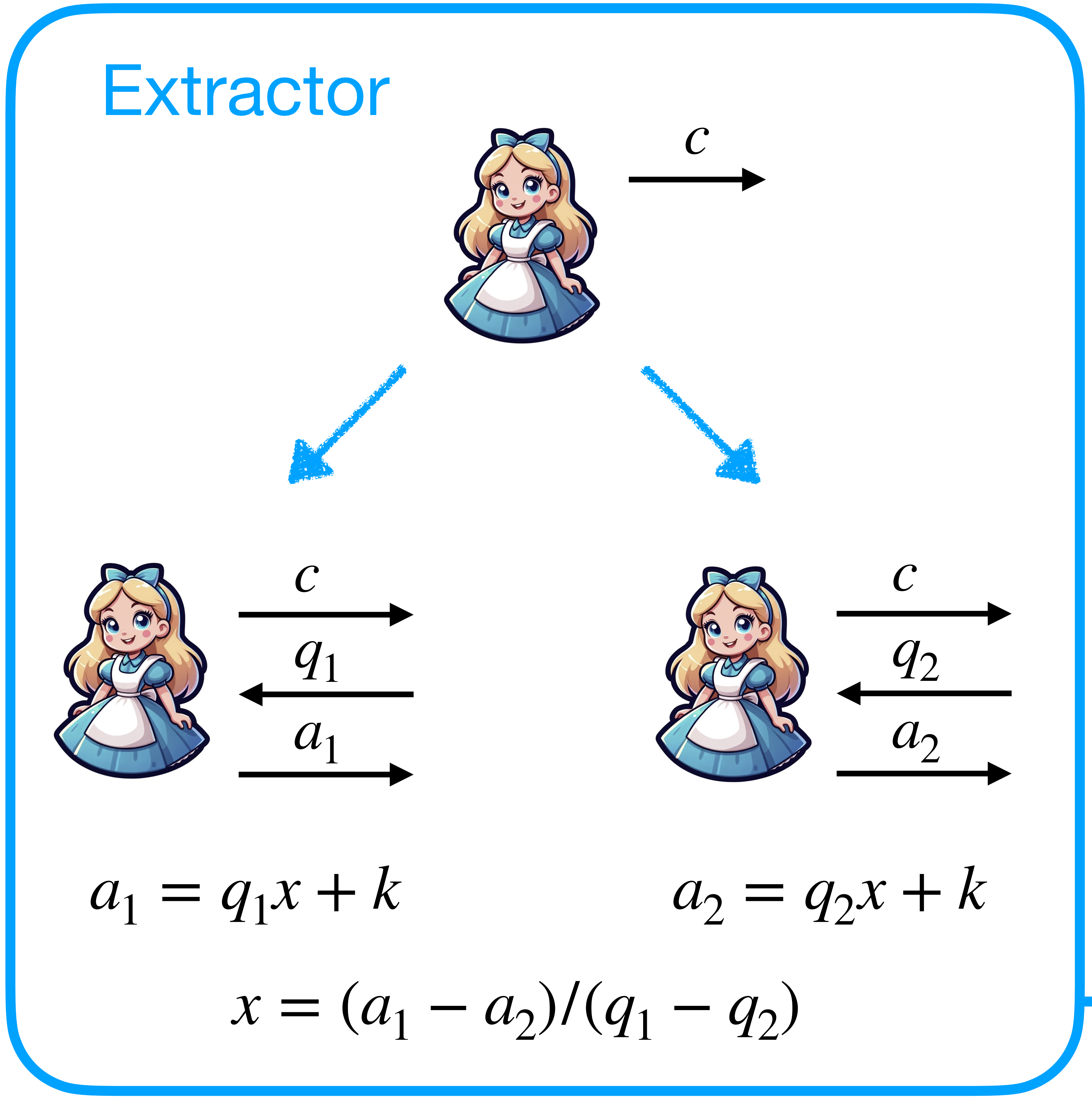
Answer



2-Special Knowledge Soundness

For any c, if  *can produce 2 $\neq$ transcripts (c, q, a) (with same c), then Extractor gets x.*

Answer



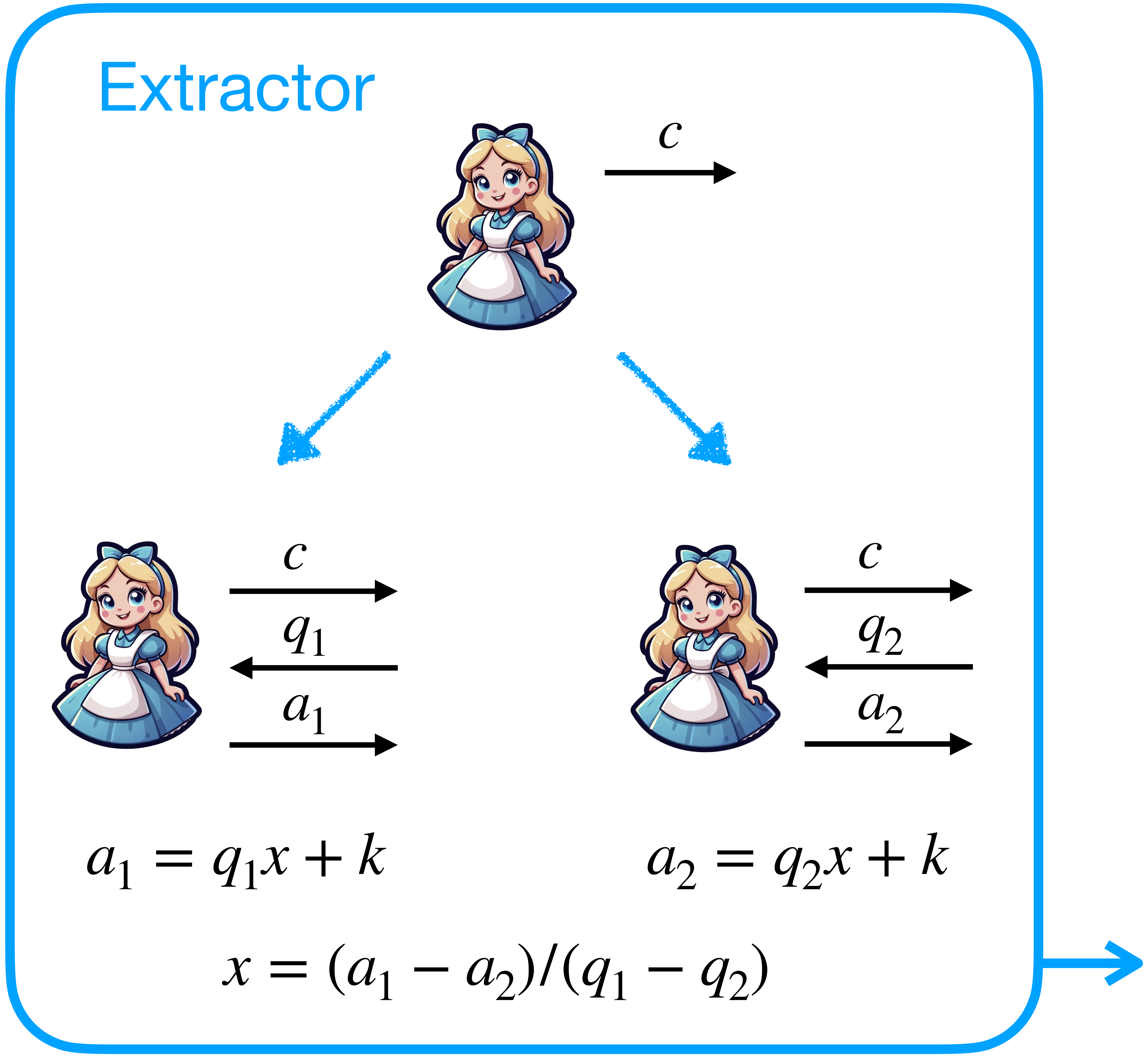
x

2-Special Knowledge Soundness

For any c, if  *can produce 2 $\neq$ transcripts (c, q, a) (with same c), then Extractor gets x.*

$\Rightarrow$ If  *don't know x, they can produce at most one such transcript.*

Answer



2-Special Knowledge Soundness

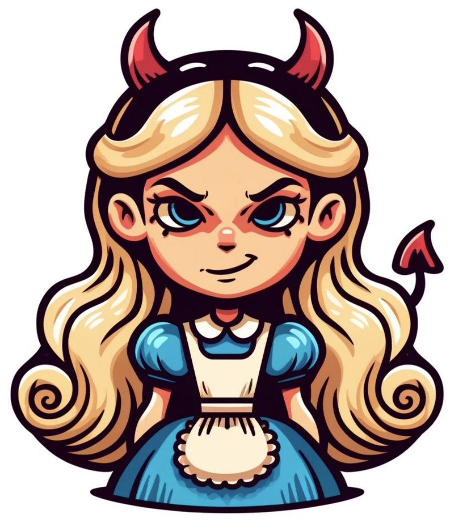
 *For any c, if*  *can produce 2 $\neq$ transcripts (c, q, a) (with same c), then Extractor gets x.*

$\Rightarrow$ If  *don't know x, they can produce at most one such transcript.*

$\Rightarrow$ Soundness error = $P\left[\text{"getting the right } q\text{"}\right]$
$= 2^{-|q|}$

Soundness Amplification

Prover

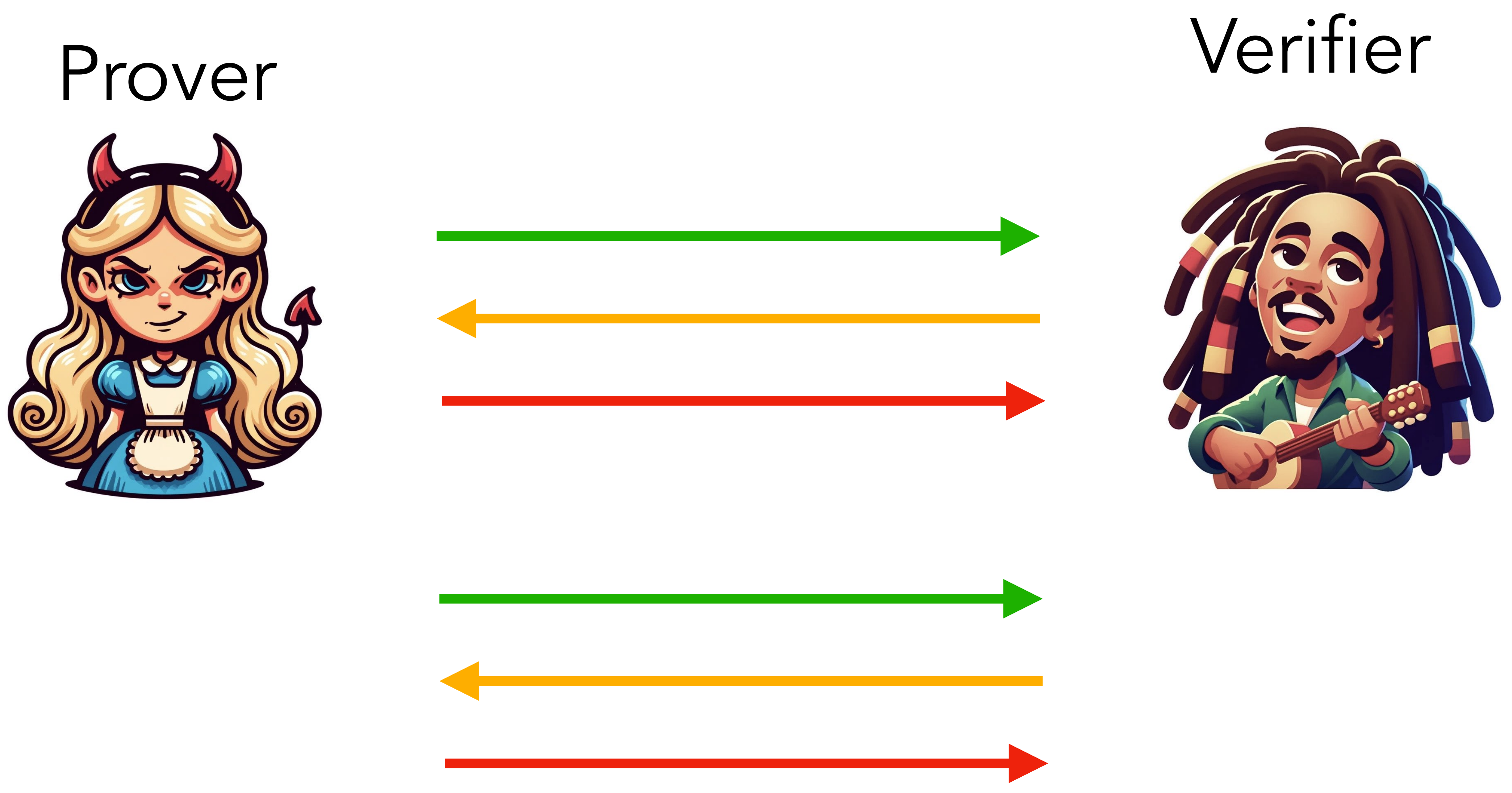


Verifier

$\rightarrow$ $\varepsilon = P[$✅$]$

⚠️ *Might be non-negligible!*

Soundness Amplification



$\rightarrow$ $\varepsilon = P[$✅$]$

$\rightarrow$ $\varepsilon = P[$✅$]$

Soundness Amplification

Prover

Verifier

$$\varepsilon = P[\text{✅}]$$

$$\times$$

$$\varepsilon = P[\text{✅}]$$

$$\varepsilon^2 = P[\text{✅ twice}]$$

Soundness Amplification

Prover

Verifier

$\varepsilon = P[$ ✅ $]$

$\times$

$\varepsilon = P[$ ✅ $]$

⋮

$\varepsilon = P[$ ✅ $]$

$\varepsilon^{\tau} = P[$ ✅ τ *times* $]$

Soundness Amplification

Prover

Verifier

Sequential repetition

$\varepsilon = P[$✅$]$

$\times$

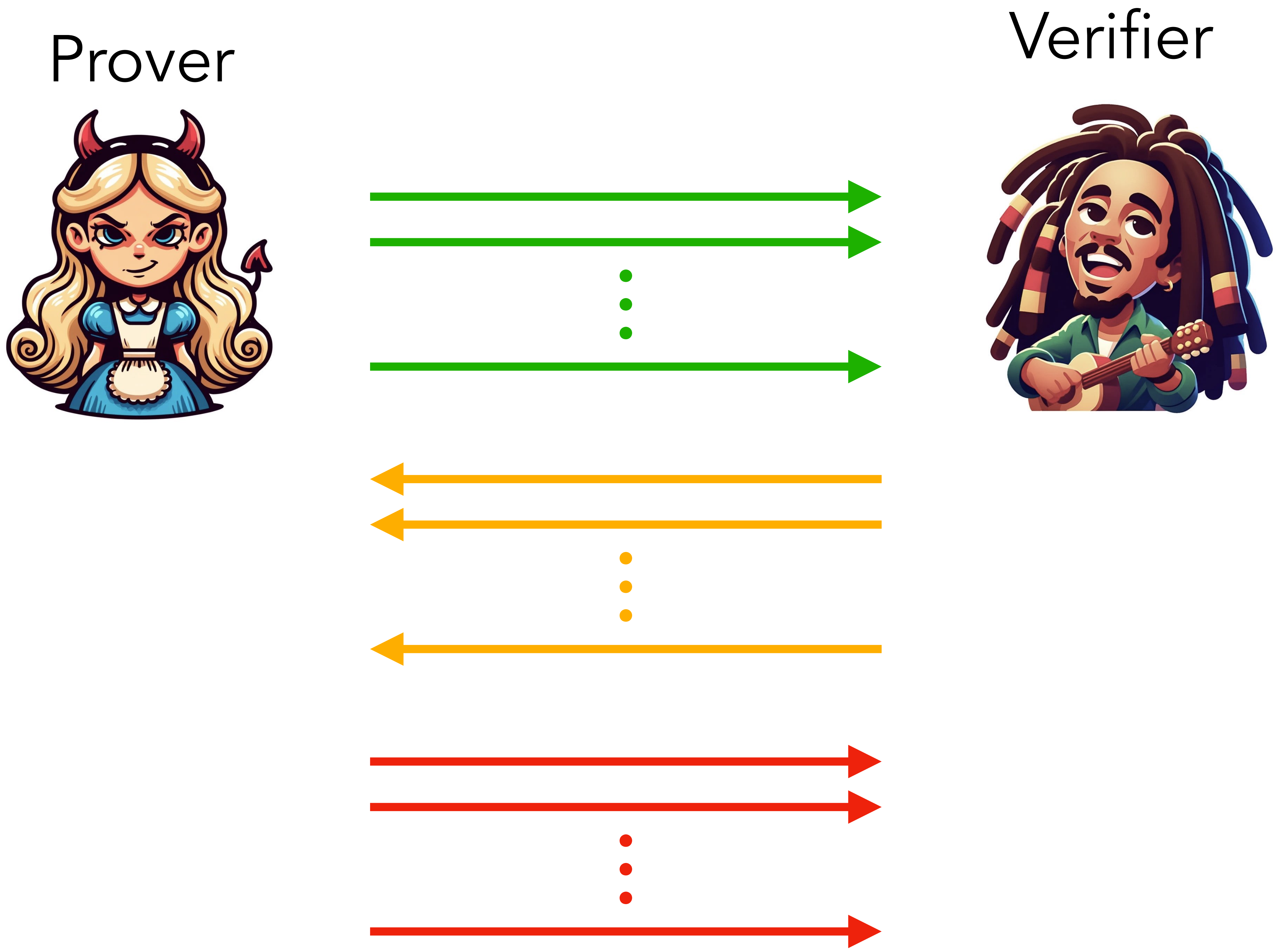
$\varepsilon = P[$✅$]$

$\vdots$

$\varepsilon = P[$✅$]$

$\varepsilon^{\tau} = P[$✅ τ *times* $]$

Soundness Amplification

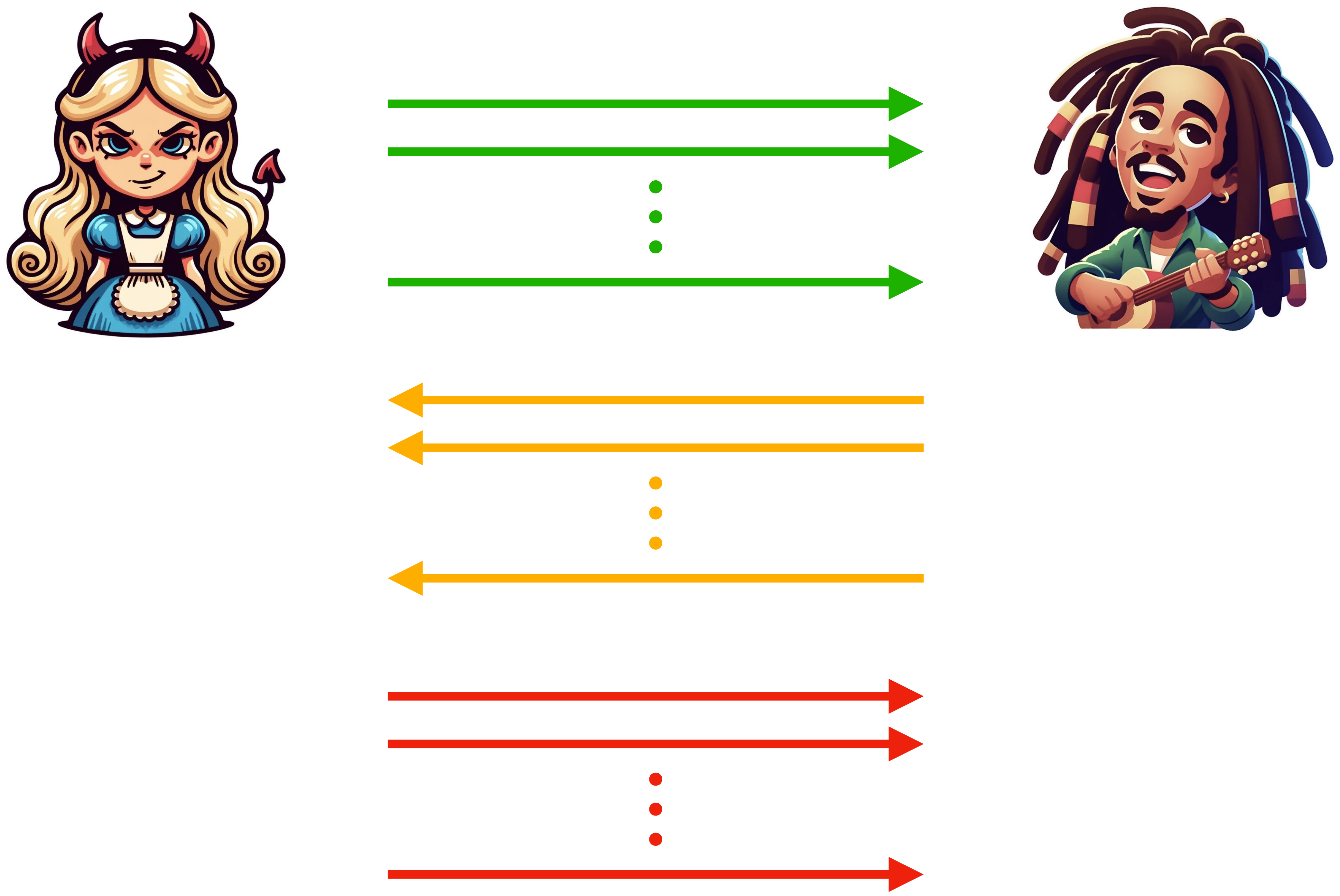


Parallel repetition

Soundness Amplification

Prover

Verifier



$\varepsilon^{\tau} = P\big[\text{✅ } \tau \text{ times}\big]$

Parallel repetition

Non-Interactive Proof

Non-Interactive Proof

I know x
s.t. $y = C(x)$

Prover

x

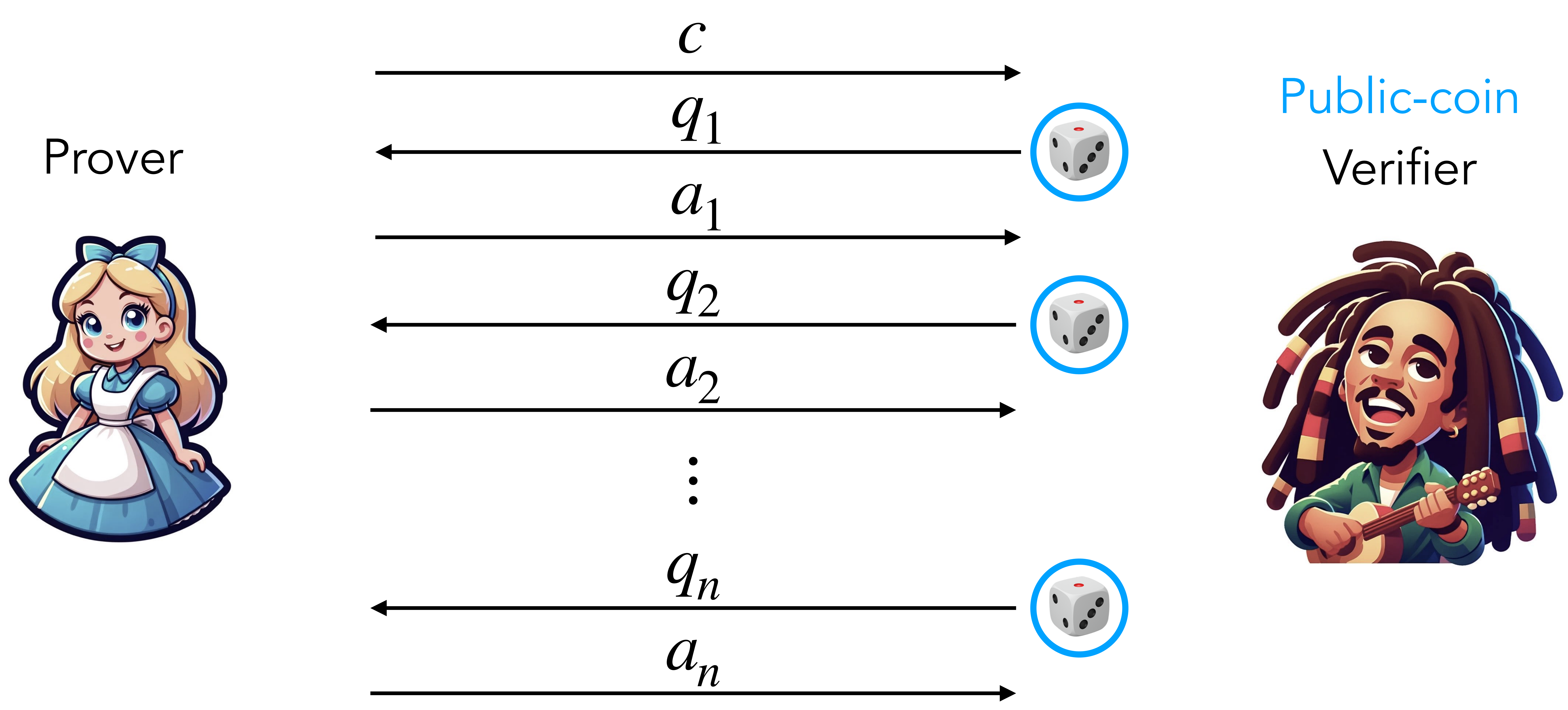
π

Verifier

y, C

Verif : $\pi \mapsto$
Ok ✅ / Nok ⛔

Fiat-Shamir Transform



Fiat-Shamir Transform

Prover

c

HASH

$q_1 = \text{Hash}(c)$

q_1

a_1

q_2

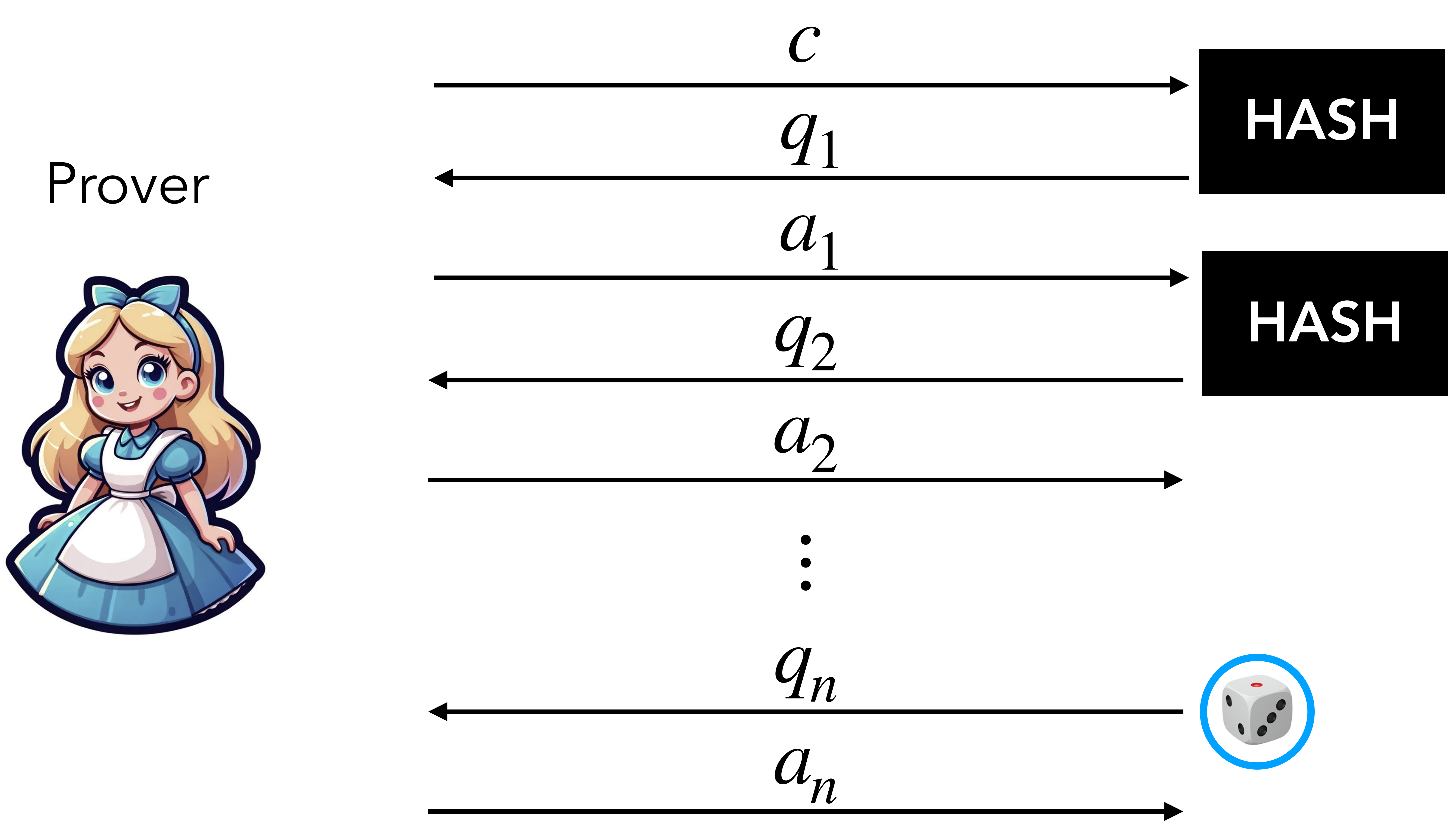
a_2

$\vdots$

q_n

a_n

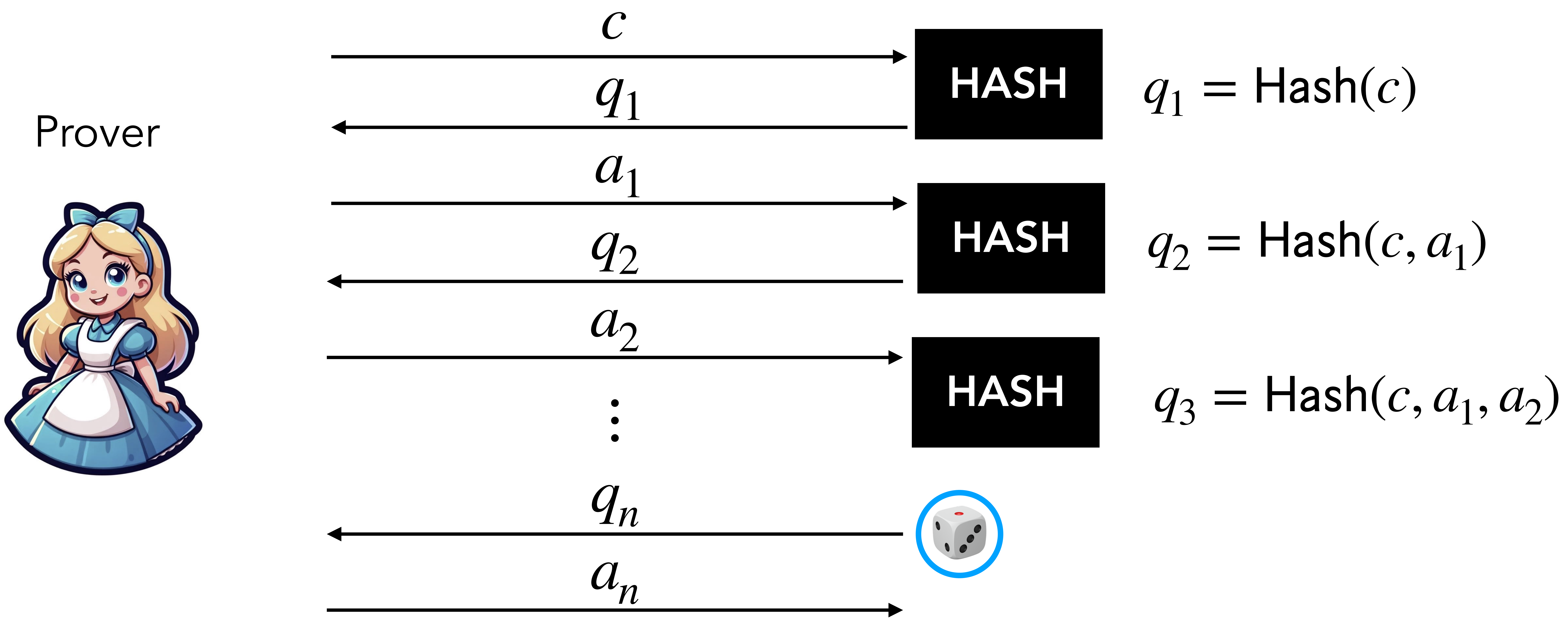
Fiat-Shamir Transform



$q_1 = \text{Hash}(c)$

$q_2 = \text{Hash}(c, a_1)$

Fiat-Shamir Transform



Fiat-Shamir Transform

Prover

c

HASH $\quad q_1 = \text{Hash}(c)$

q_1

a_1

HASH $\quad q_2 = \text{Hash}(c, a_1)$

q_2

a_2

HASH $\quad q_3 = \text{Hash}(c, a_1, a_2)$

$\vdots$

q_n

a_n

$$\pi = (c, a_1, \ldots, a_n)$$

Fiat-Shamir Transform

Prover

c → HASH

q_1 ←

a_1 → HASH

q_2 ←

a_2 → HASH

$\vdots$

q_n ←

a_n →

$q_1 = \text{Hash}(c)$

$q_2 = \text{Hash}(c, a_1)$

$q_3 = \text{Hash}(c, a_1, a_2)$

$\vdots$

$\pi = (c, a_1, \ldots, a_n)$

checks π by recomputing the hashs instead of randomly picking the q_i's

Fiat-Shamir Transform

Prover

c

q_1

a_1

q_2

a_2

$\vdots$

q_n

a_n

HASH $\quad q_1 = \text{Hash}(c)$

HASH $\quad q_2 = \text{Hash}(c, a_1)$

HASH $\quad q_3 = \text{Hash}(c, a_1, a_2)$

$\vdots$

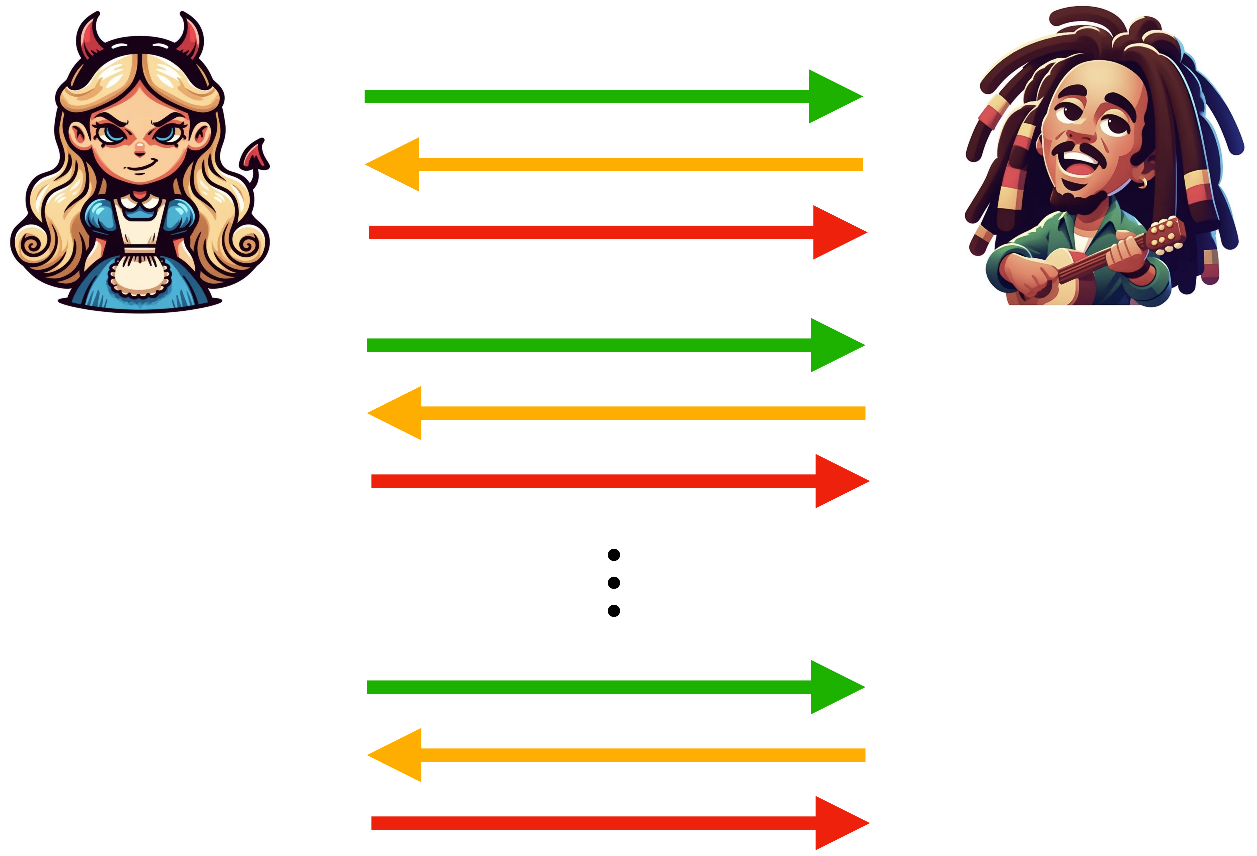
$\pi = (c, a_1, \ldots, a_n)$

Hash$(\,\cdot\,)$ behaves as a random function.
Security in the Random Oracle Model (ROM).

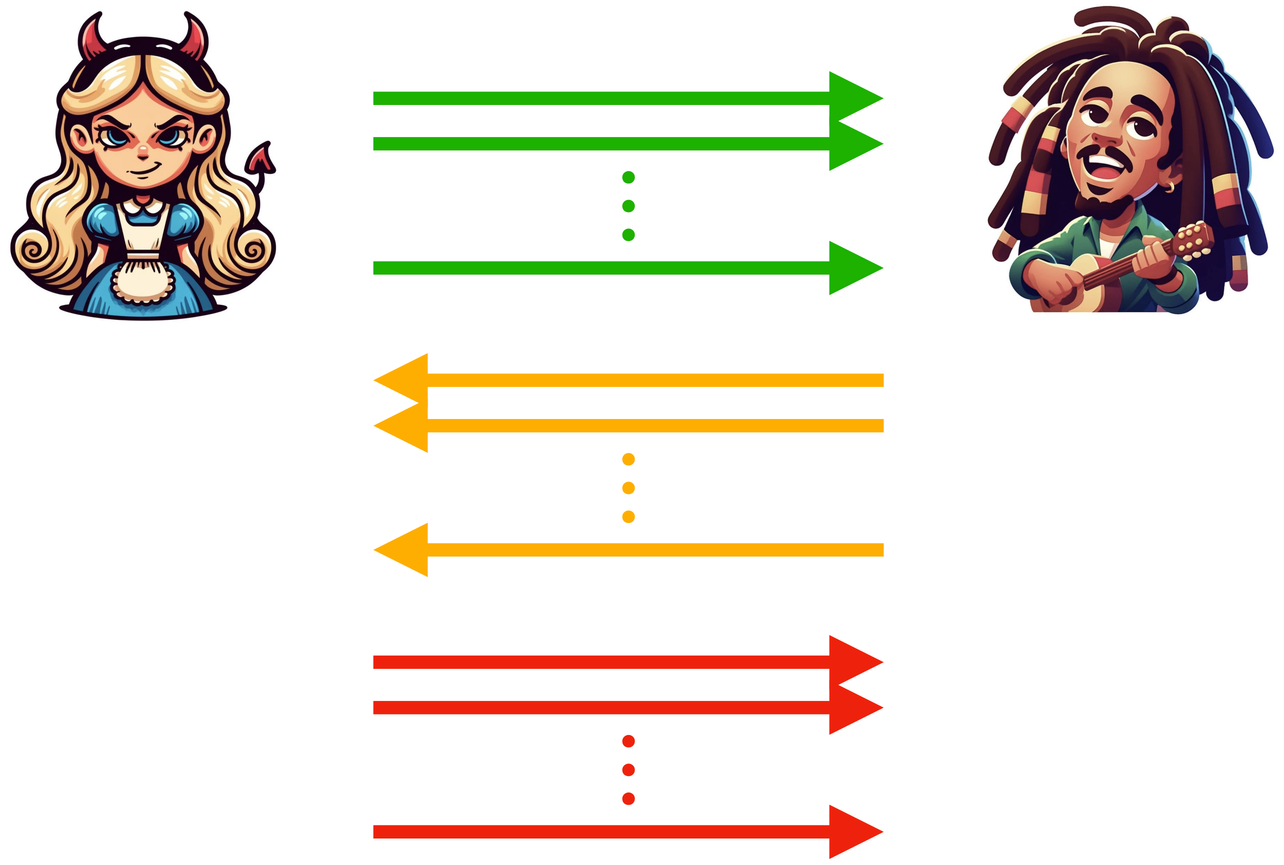
checks π by recomputing the hashs
instead of randomly picking the q_i's

Question 5

Question 5



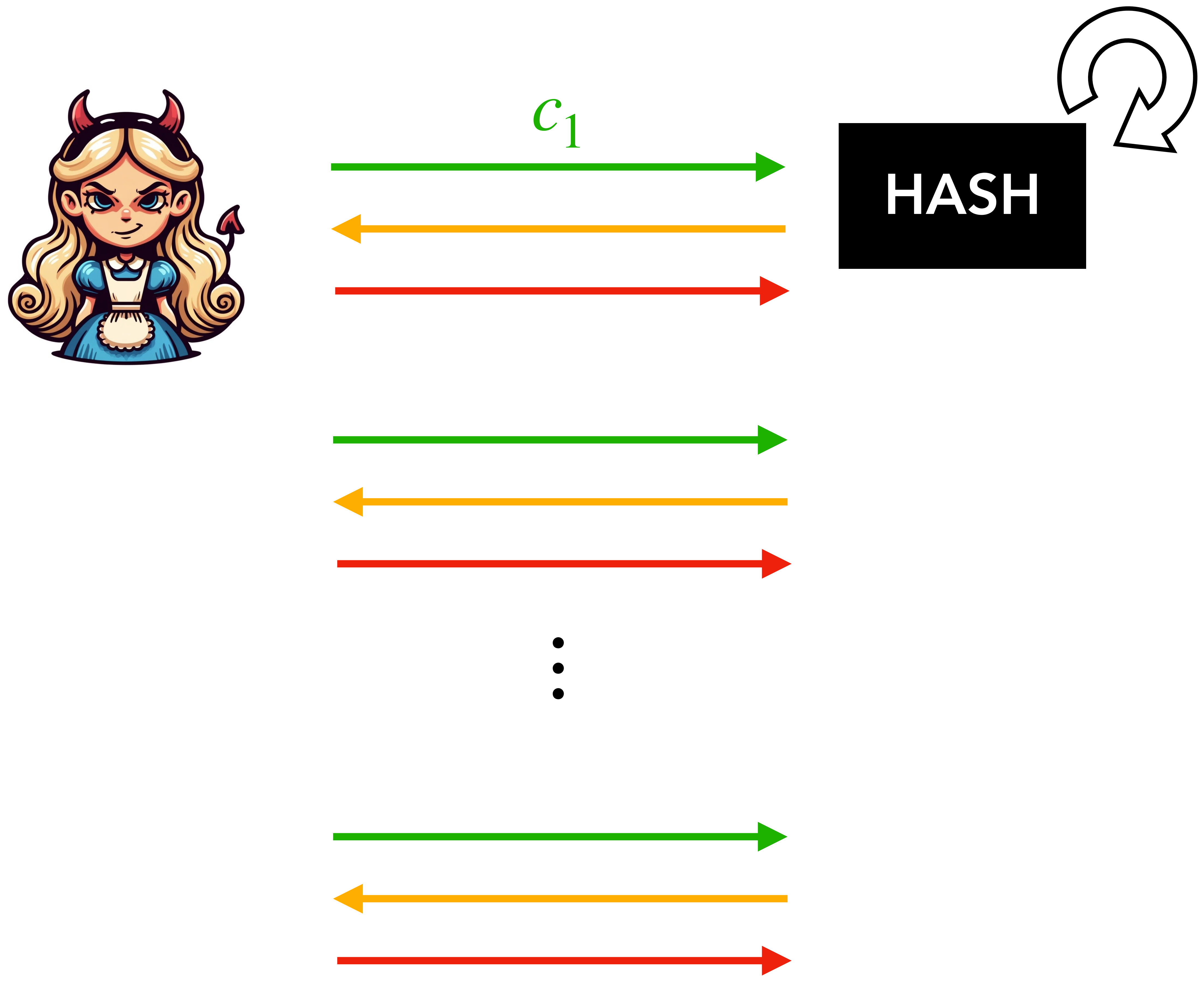
Sequential repetition



Parallel repetition

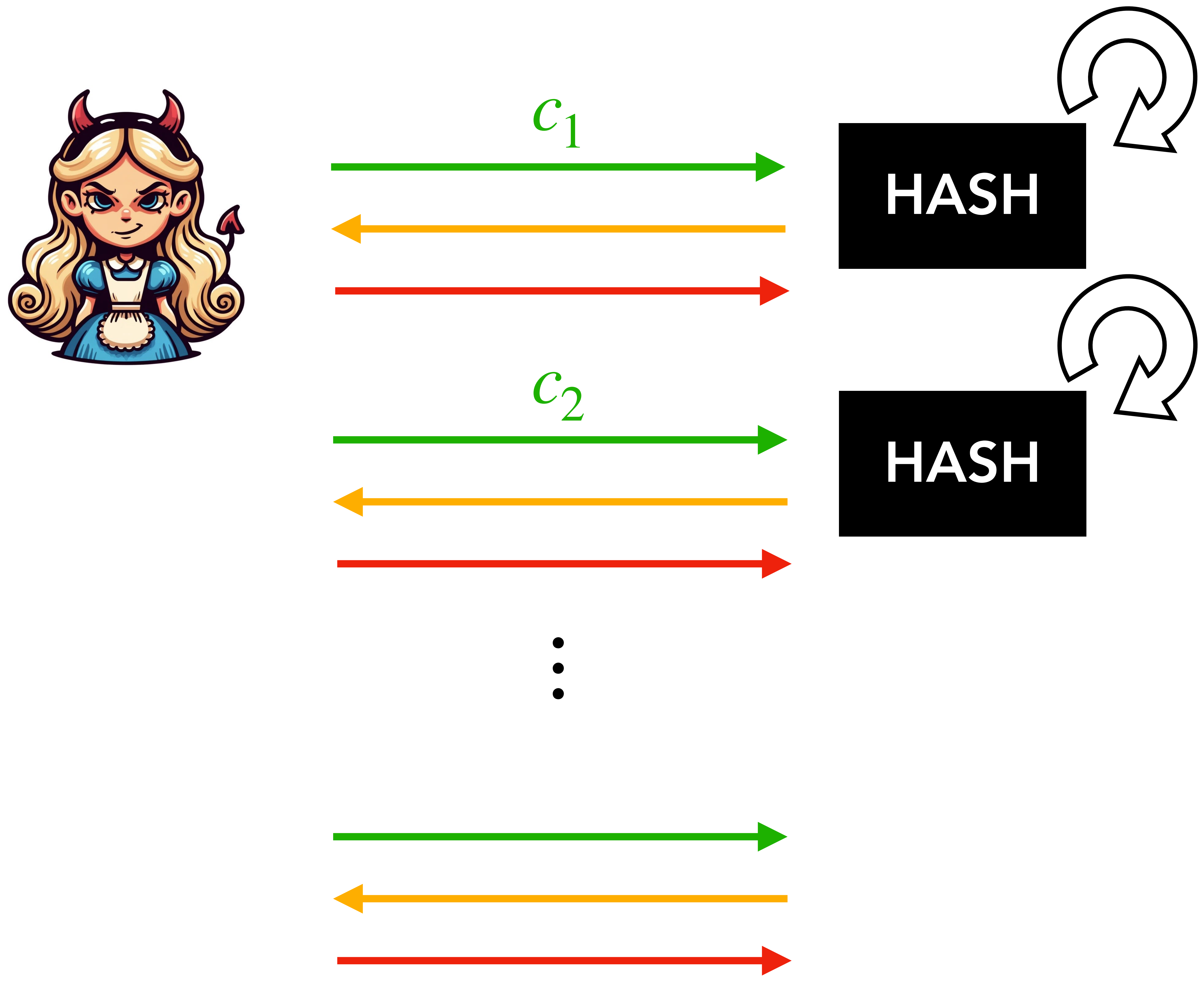
Q. With Fiat-Shamir, which one is better and why?

Answer



Try new c_1 until $q_1 = \text{Hash}(c_1)$
can be answered $\Rightarrow 1/\varepsilon$ trials

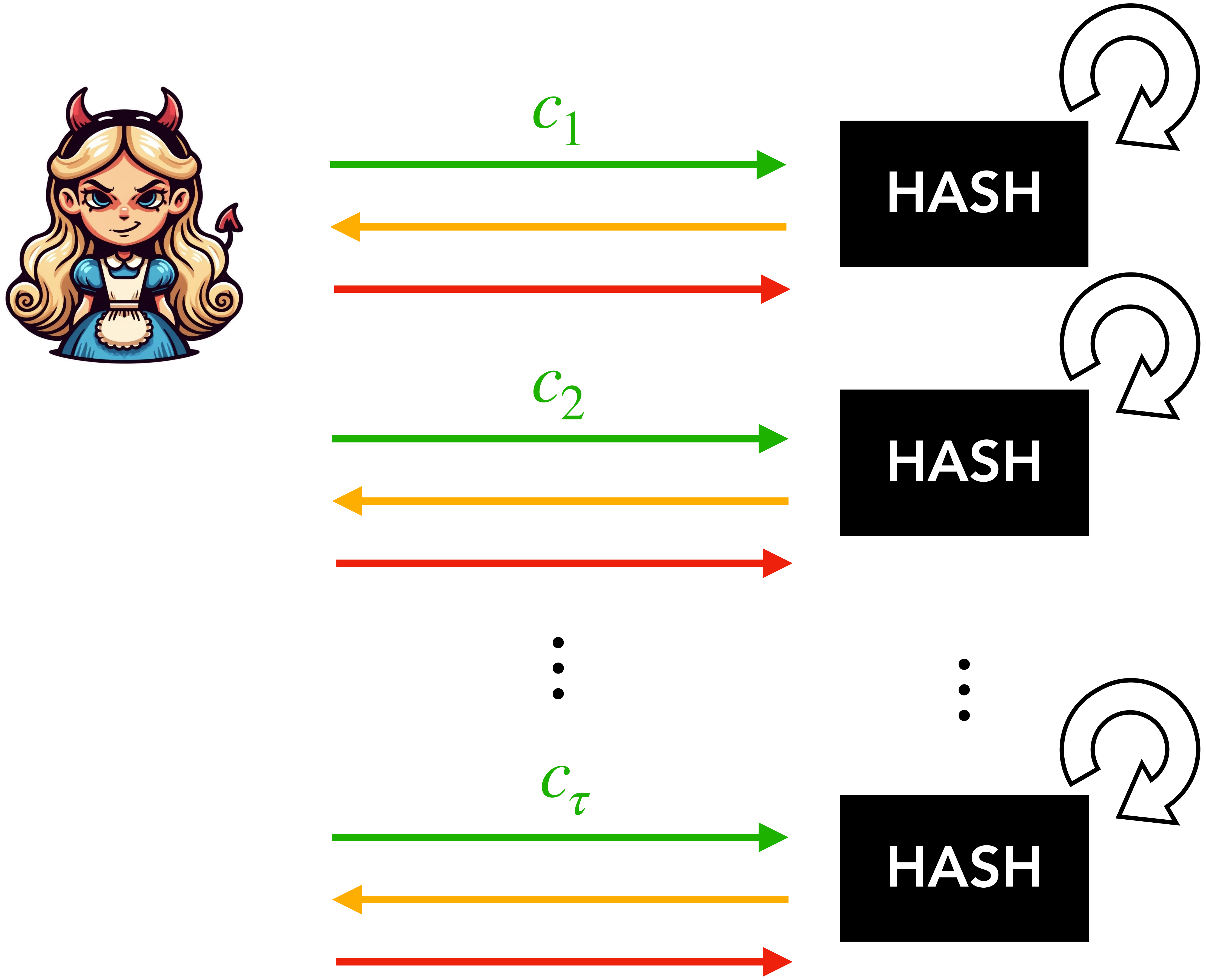
Answer



Try new c_1 until $q_1 = \text{Hash}(c_1)$
can be answered $\Rightarrow 1/\varepsilon$ trials

Try new c_2 until $q_2 = \text{Hash}(c_1, a_1, c_2)$
can be answered $\Rightarrow 1/\varepsilon$ trials

Answer

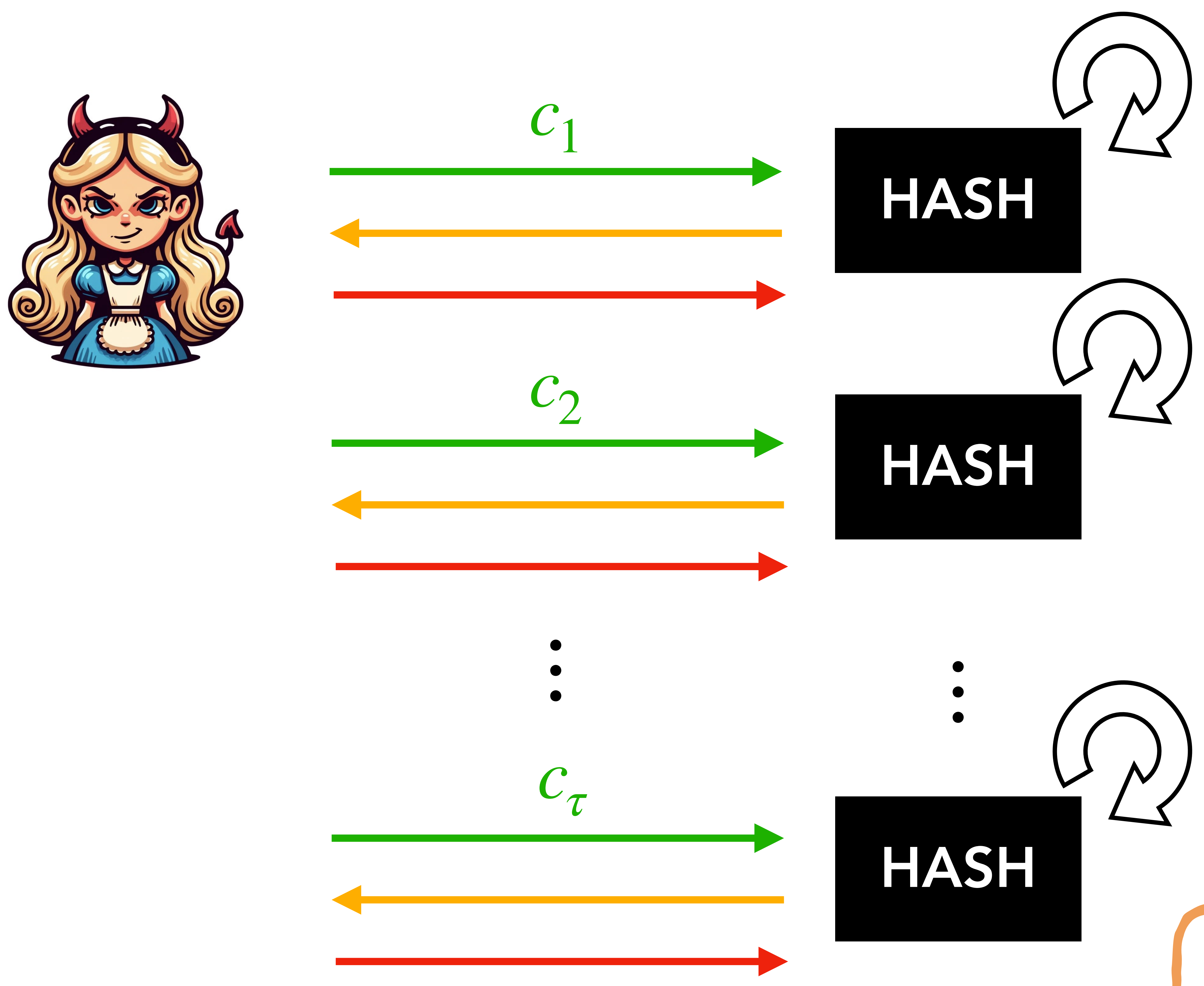


Try new c_1 until $q_1 = \text{Hash}(c_1)$
can be answered $\Rightarrow 1/\varepsilon$ trials

Try new c_2 until $q_2 = \text{Hash}(c_1, a_1, c_2)$
can be answered $\Rightarrow 1/\varepsilon$ trials

Try new c_τ until $q_\tau = \text{Hash}(c_1, a_1, \ldots, c_\tau)$
can be answered $\Rightarrow 1/\varepsilon$ trials

Answer



Try new c_1 until $q_1 = \text{Hash}(c_1)$ can be answered $\Rightarrow 1/\varepsilon$ trials

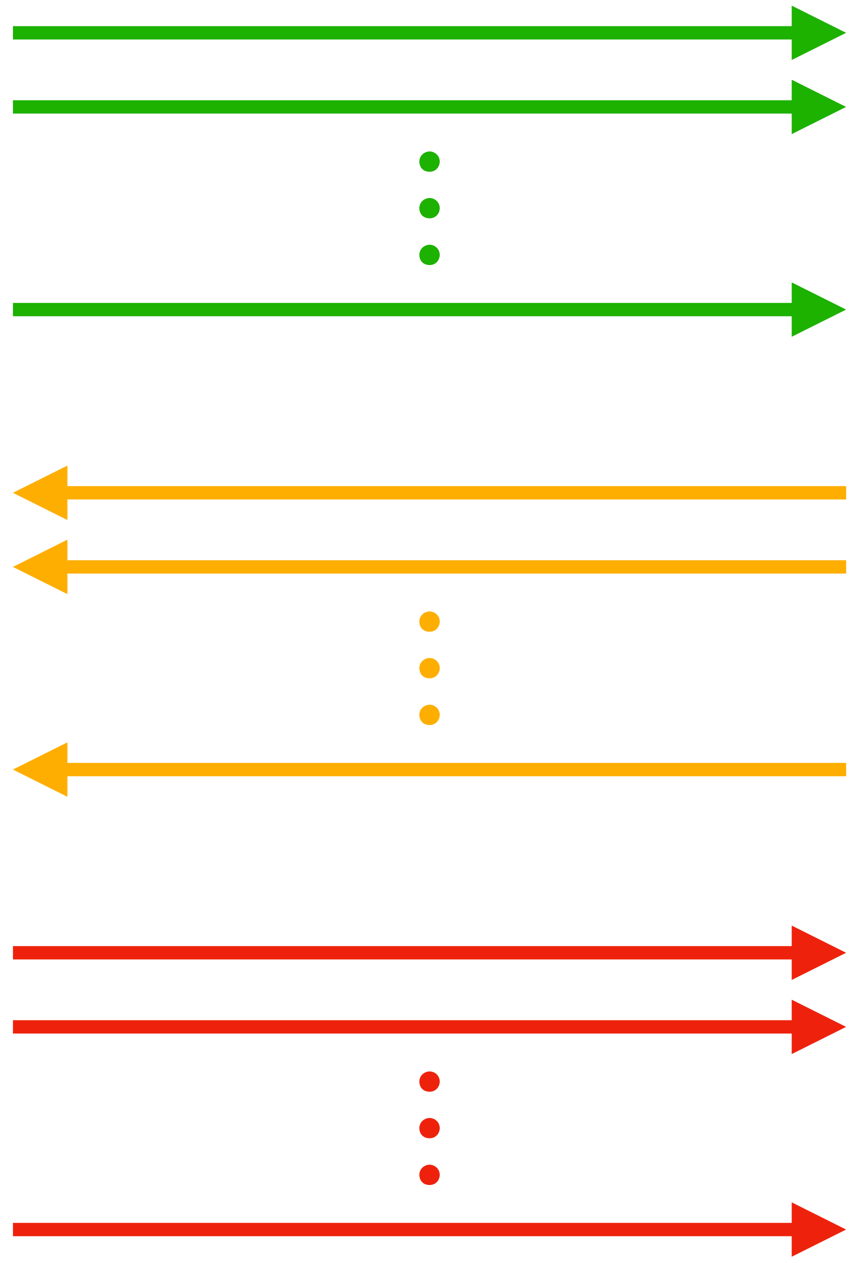
Try new c_2 until $q_2 = \text{Hash}(c_1, a_1, c_2)$ can be answered $\Rightarrow 1/\varepsilon$ trials

Try new c_τ until $q_\tau = \text{Hash}(c_1, a_1, \ldots, c_\tau)$ can be answered $\Rightarrow 1/\varepsilon$ trials

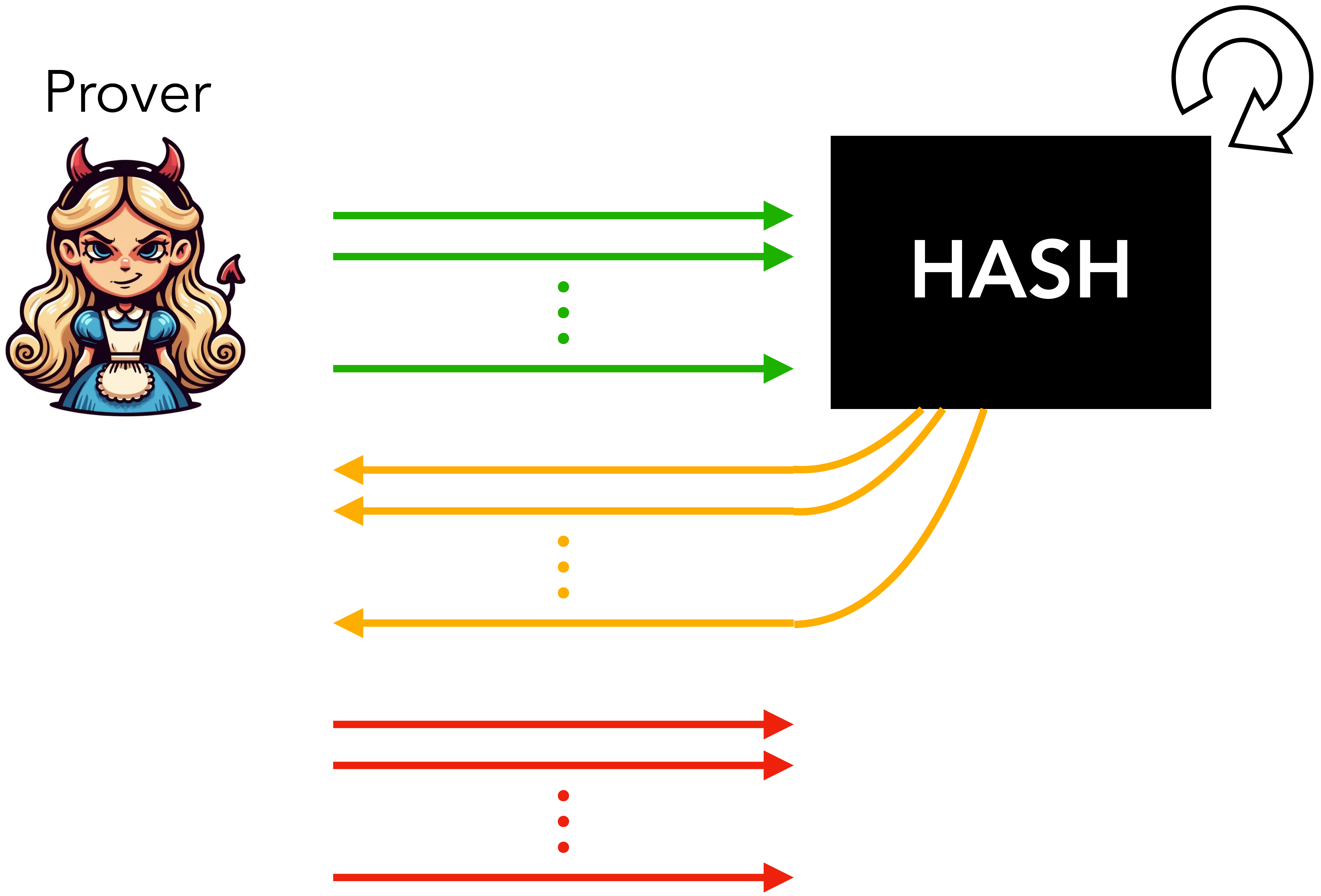
! Forge in time $\tau \cdot (1/\varepsilon) \Rightarrow$ sequential repetition is weak with Fiat Shamir.

Answer

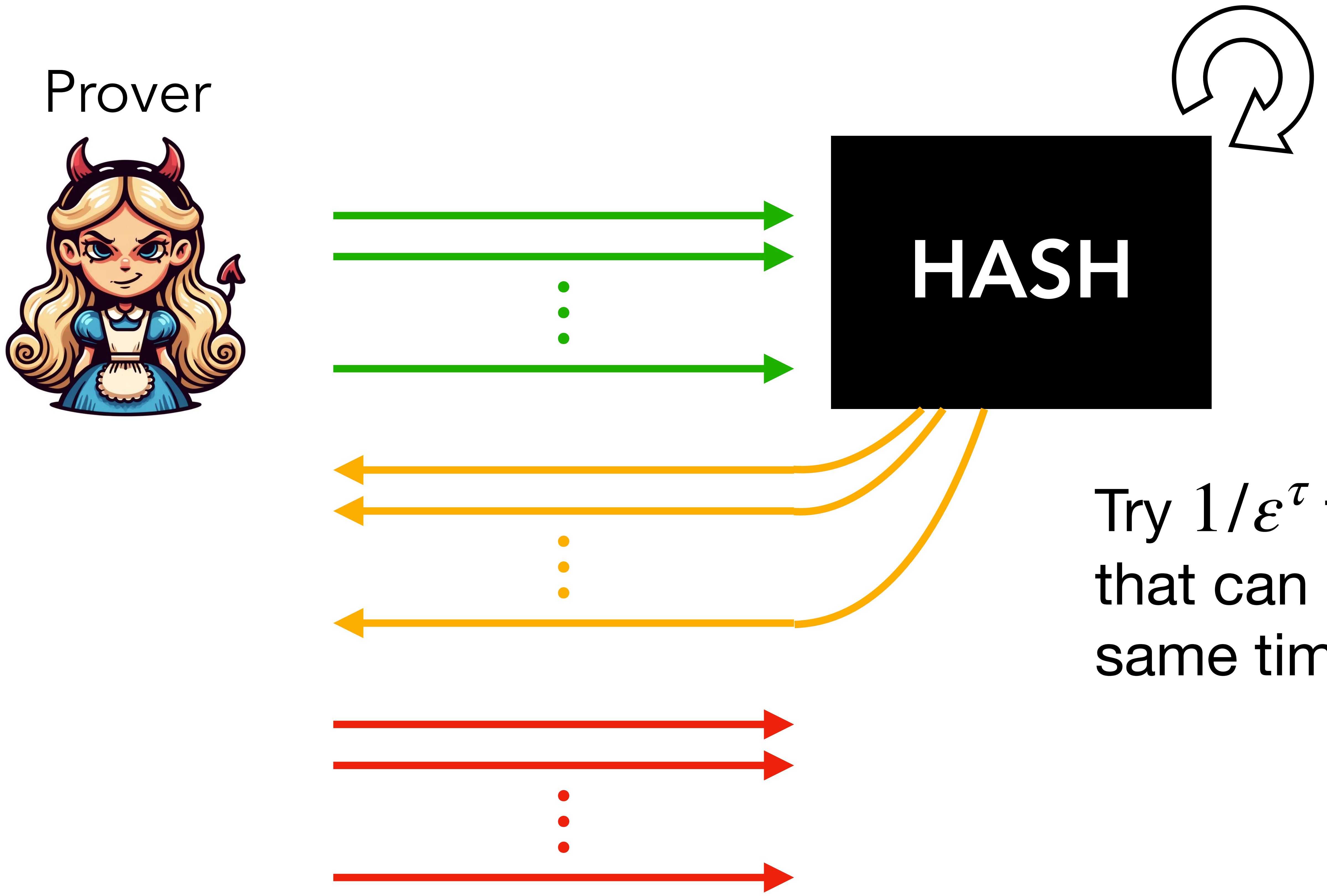
Prover



Answer

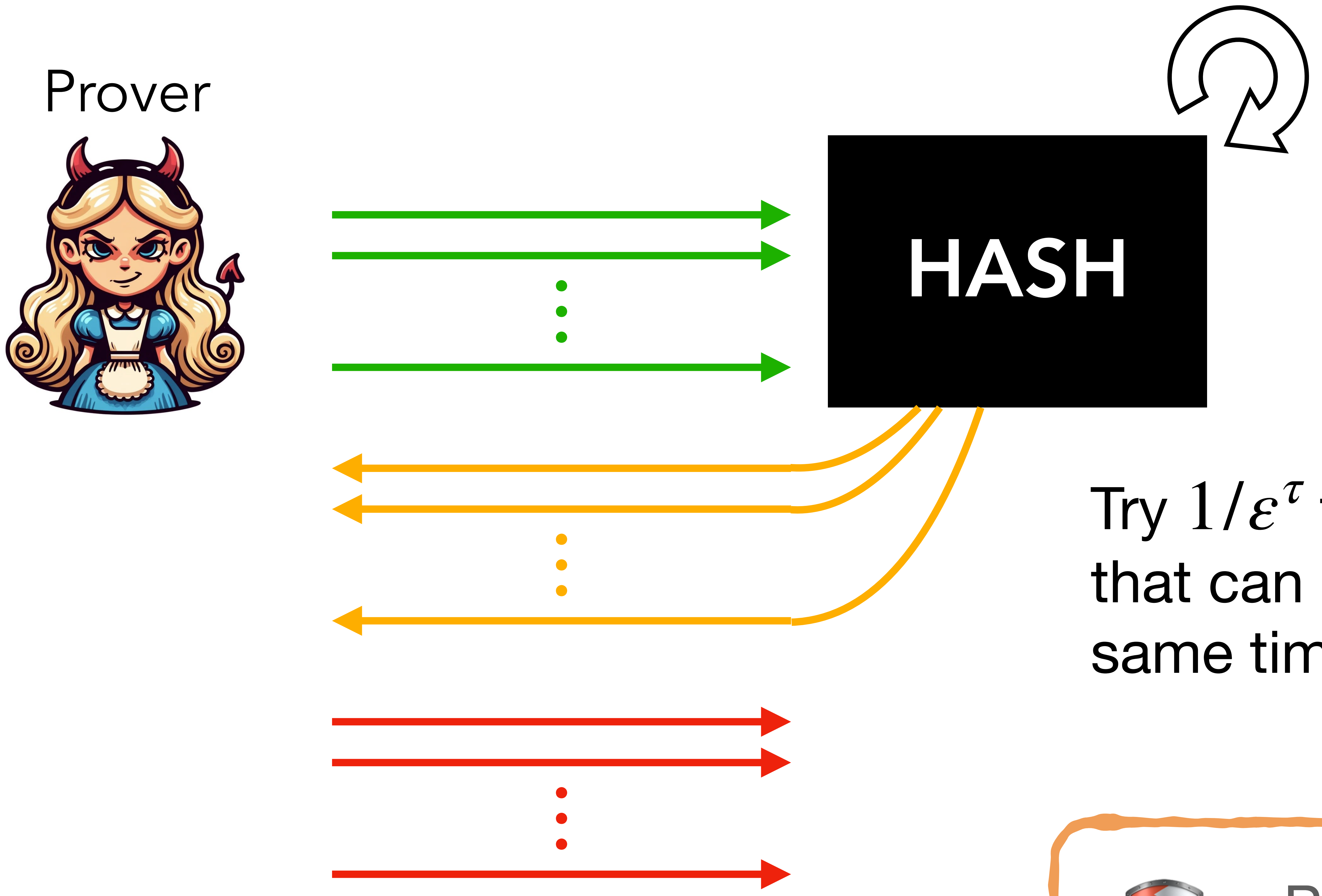


Answer



Try $1/\varepsilon^\tau$ times to get τ questions that can all be answered at the same time.

Answer



Try $1/\varepsilon^{\tau}$ times to get τ questions that can all be answered at the same time.

Parallel repetition is secure with Fiat Shamir.

ZK PoK + Fiat-Shamir = Signature

x

$\vec{c}$

$\vec{q}$

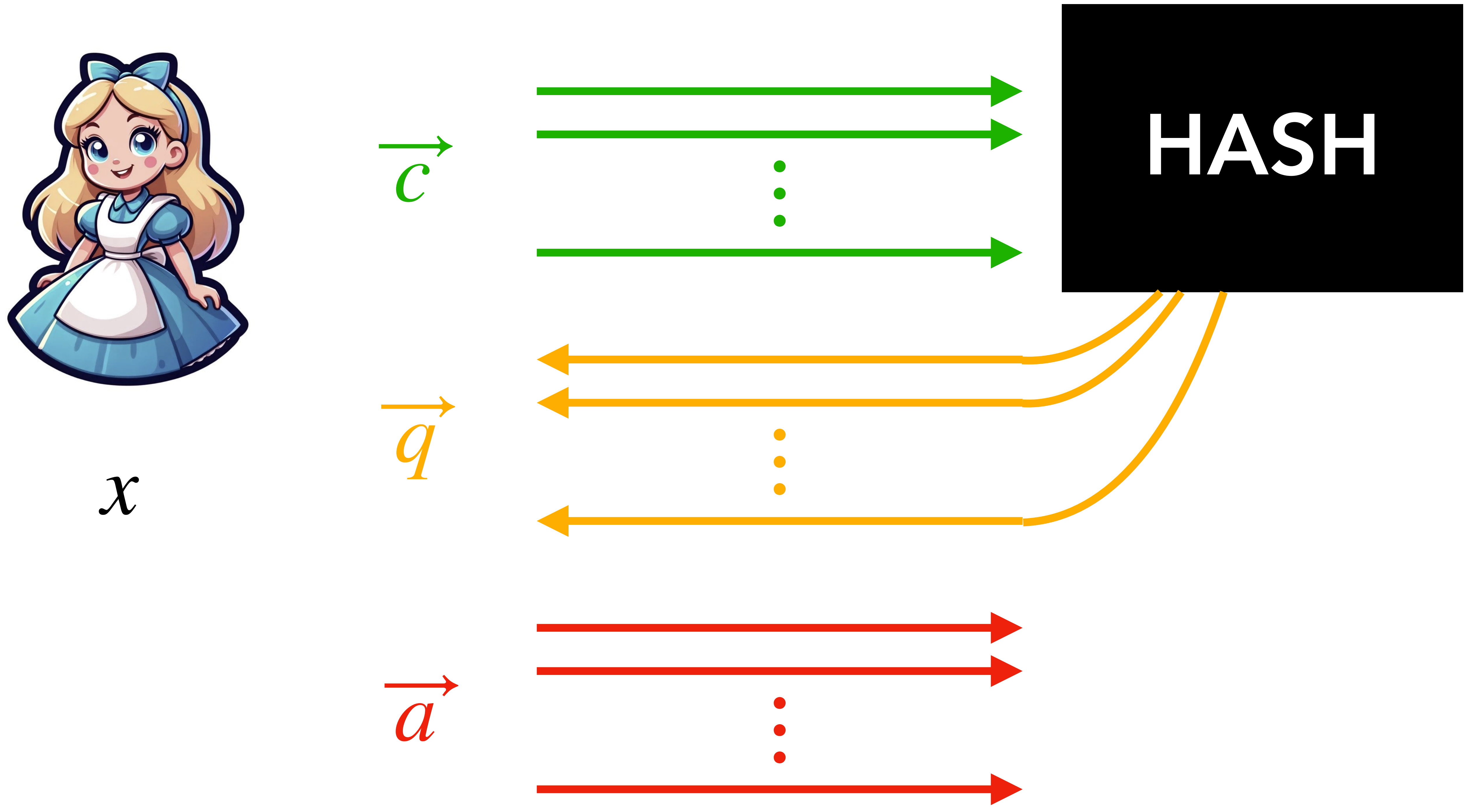
$\vec{a}$

$y = C(x)$

Verif Proof$(y, \vec{c}, \vec{q}, \vec{a}) \mapsto$ ✅ / ⛔

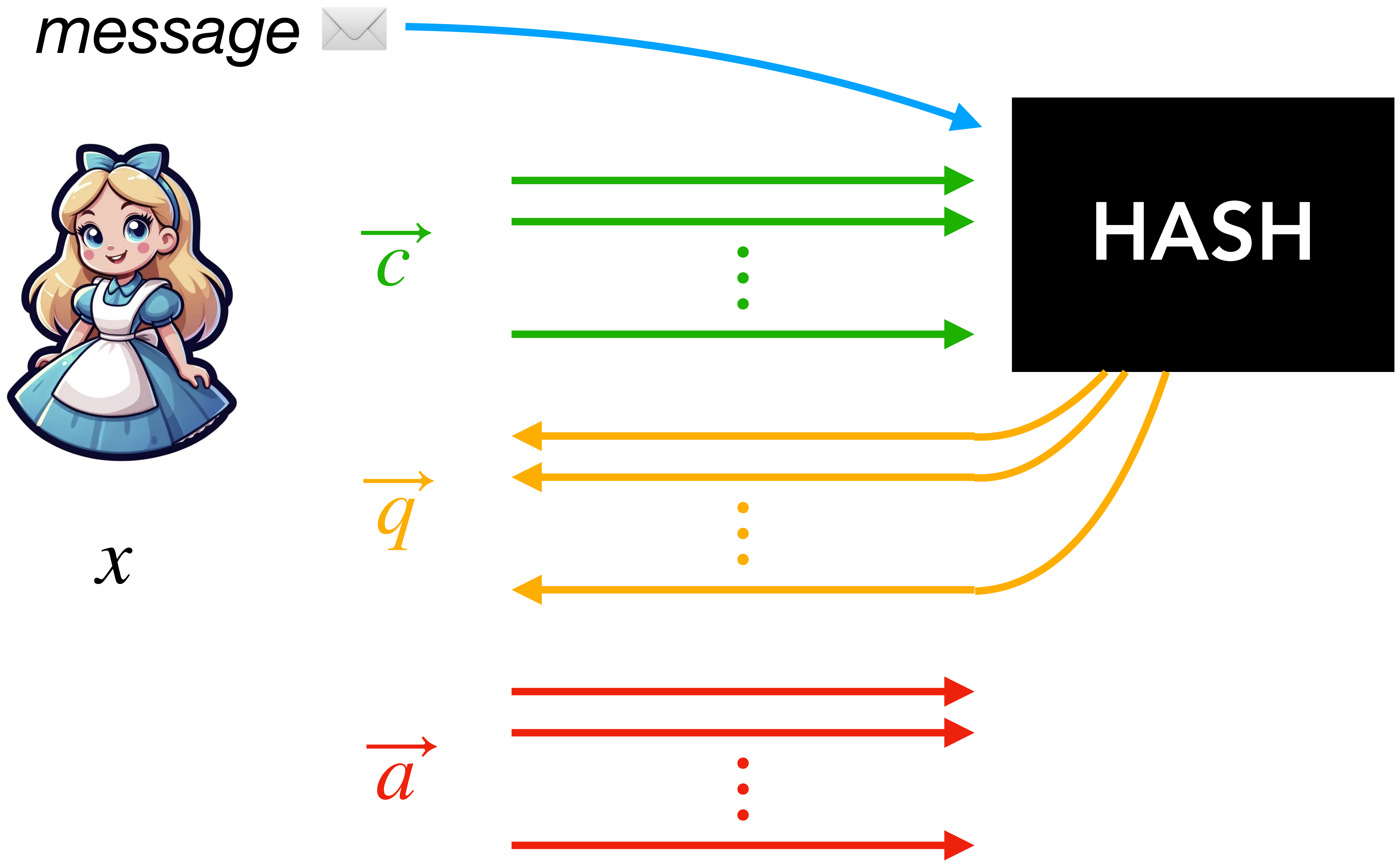
$\vec{a} =$ Answer$(x, \vec{c}, \vec{q})$

ZK PoK + Fiat-Shamir = Signature



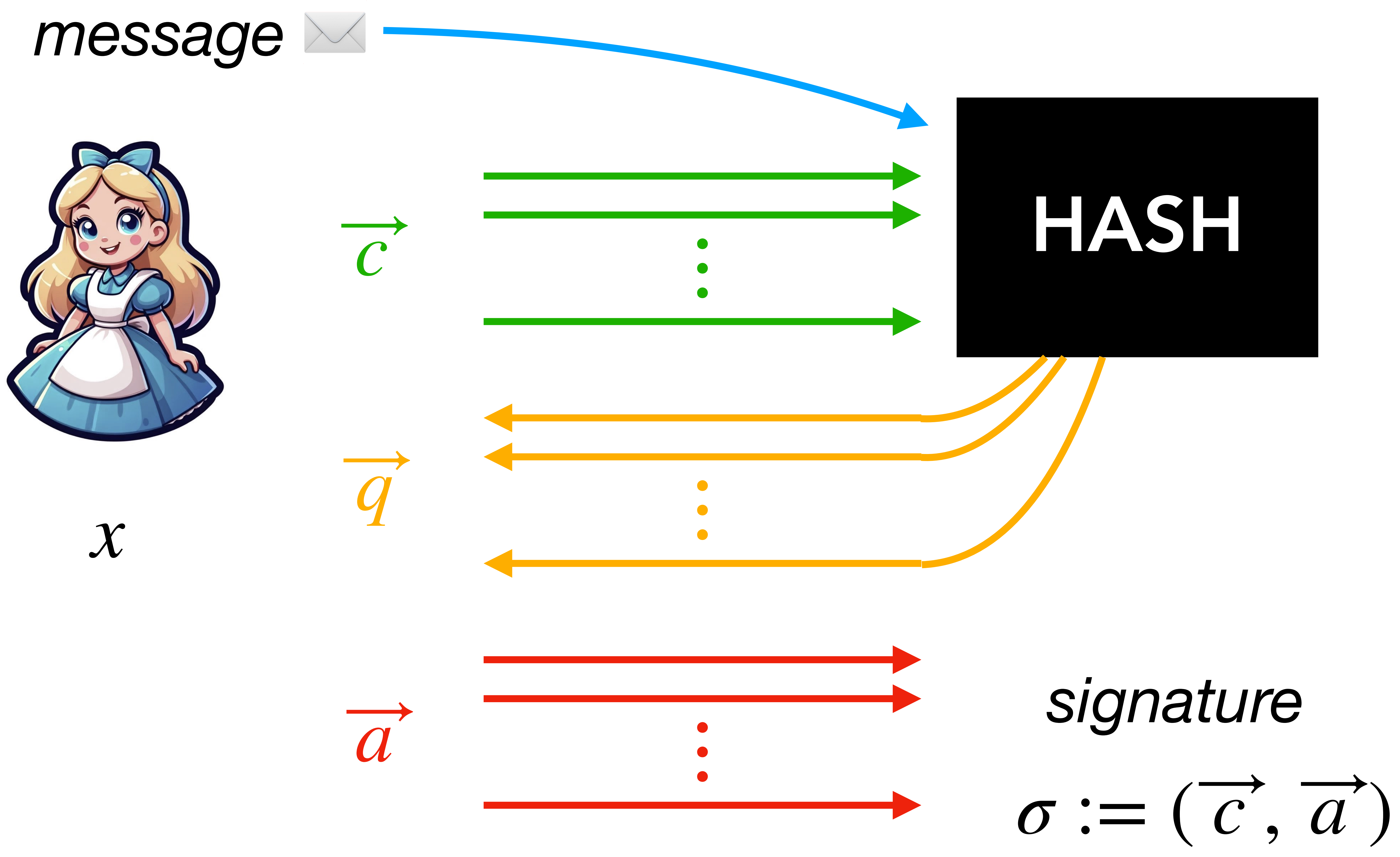
$$\vec{a} = \text{Answer}(x, \vec{c}, \vec{q})$$

ZK PoK + Fiat-Shamir = Signature



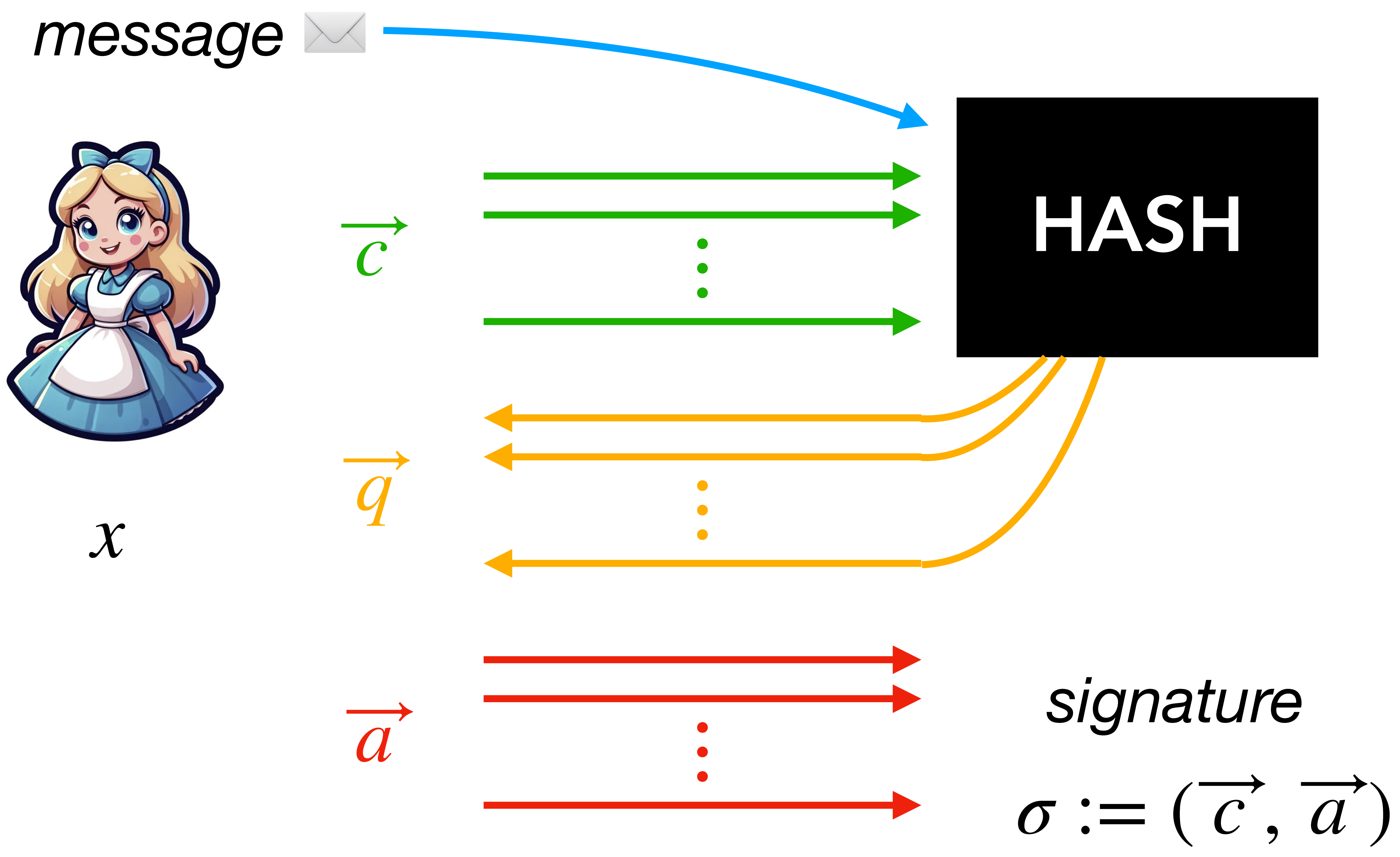
$$\vec{a} = \text{Answer}(x, \vec{c}, \vec{q})$$

ZK PoK + Fiat-Shamir = Signature



$$\vec{a} = \text{Answer}(x, \vec{c}, \vec{q})$$

ZK PoK + Fiat-Shamir = Signature



$\vec{a}$ = Answer$(x, \vec{c}, \vec{q})$

<u>Verif Sig (y, σ, msg)</u>:

1. $\vec{q}$ = Hash$(msg, \vec{c})$
2. Verif Proof$(y, \vec{c}, \vec{q}, \vec{a})$

Signature Security

- Security in the Random Oracle Model
  - EUF-CMA adversary $\Rightarrow$ algorithm to recover x

Signature Security

- Security in the Random Oracle Model
    - EUF-CMA adversary $\Rightarrow$ algorithm to recover x
- Zero Knowledge $\Rightarrow$ signatures do not leak information on x
    - ZK Simulator $\rightarrow$ Signature oracle in the EUF-CMA game

Signature Security

- Security in the Random Oracle Model
  - EUF-CMA adversary $\Rightarrow$ algorithm to recover x
- Zero Knowledge $\Rightarrow$ signatures do not leak information on x
  - ZK Simulator $\rightarrow$ Signature oracle in the EUF-CMA game
- Knowledge Soundness $\Rightarrow$ x can be extracted from an EUF-CMA adversary
  - Extractor $\rightarrow$ Recovers x from forged signatures (1, 2, a few)

Introduction to the MPC-in-the-Head Paradigm

Secret Sharing

$$[\![x]\!] \;=\; ([\![x]\!]_1, \ldots, [\![x]\!]_N) \quad \in \mathbb{F}^N$$

Secret Sharing

$$[\![x]\!] \;=\; ([\![x]\!]_1, \ldots, [\![x]\!]_N) \quad \in \mathbb{F}^N$$

- Random generation: $[\![x]\!] \leftarrow \mathsf{Generate}(x, \$)$
- Deterministic reconstruction: $x = \mathsf{Reconstruct}([\![x]\!])$

Secret Sharing

$$[\![x]\!] = ([\![x]\!]_1, \ldots, [\![x]\!]_N) \in \mathbb{F}^N$$

- Random generation: $[\![x]\!] \leftarrow \mathsf{Generate}(x, \$)$
- Deterministic reconstruction: $x = \mathsf{Reconstruct}([\![x]\!])$
- Privacy: $[\![x]\!]$ is ℓ-private

  $\Leftrightarrow$ any set of ℓ shares $\{[\![x]\!]_i\}$ is statistically independent of x

Secret Sharing

$$[\![x]\!] \;=\; ([\![x]\!]_1, \ldots, [\![x]\!]_N) \quad \in \mathbb{F}^N$$

- Random generation: $[\![x]\!] \leftarrow \mathsf{Generate}(x, \$)$
- Deterministic reconstruction: $x = \mathsf{Reconstruct}([\![x]\!])$
- Privacy: $[\![x]\!]$ is ℓ-private

$\Leftrightarrow$ any set of ℓ shares $\{[\![x]\!]_i\}$ is statistically independent of x

$\Leftrightarrow$ any set of ℓ shares $\{[\![x]\!]_i\}$ can be perfectly simulated w/o x

Secret Sharing

Example: additive secret sharing

- Reconstruction:

$$x = \sum_{i=1}^{N} [\![x]\!]_i$$

- Generation:

$$[\![x]\!]_1, \ldots, [\![x]\!]_{N-1} \leftarrow \$ \,, \qquad [\![x]\!]_N = x - \sum_{i=1}^{N-1} [\![x]\!]_i$$

Question 6

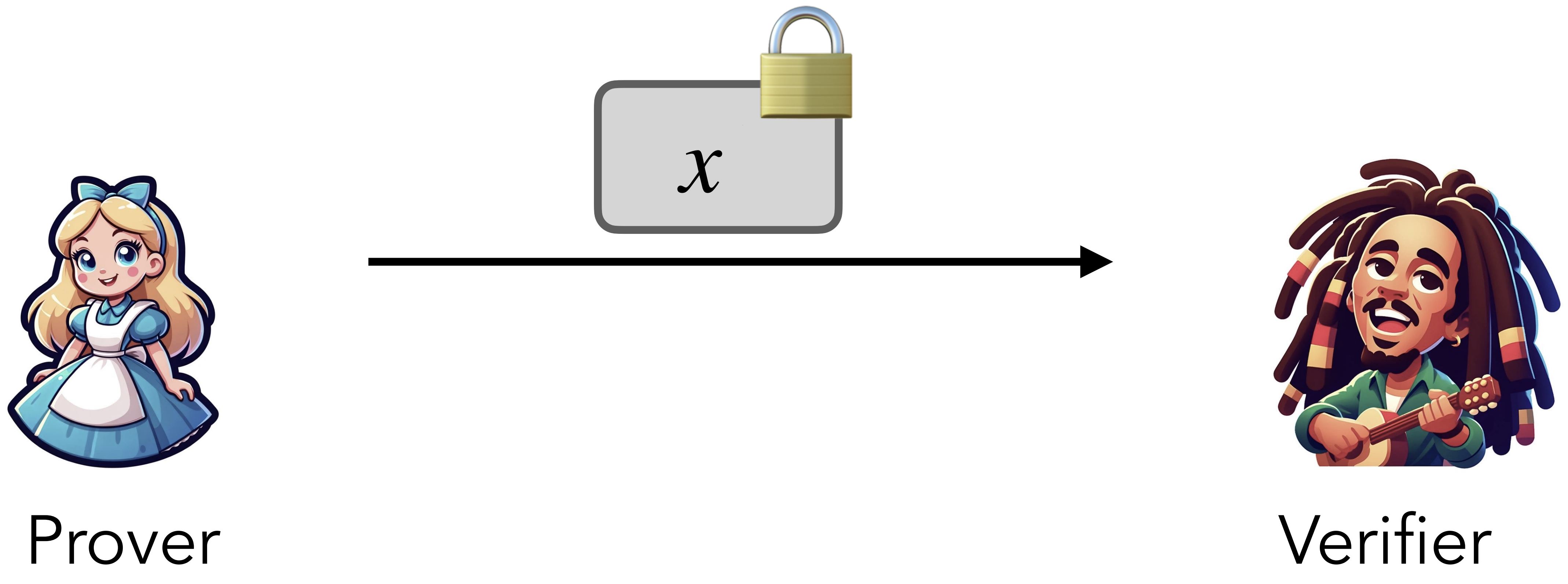
Q. Additive sharing is ℓ-private for which ℓ ?

Question 6

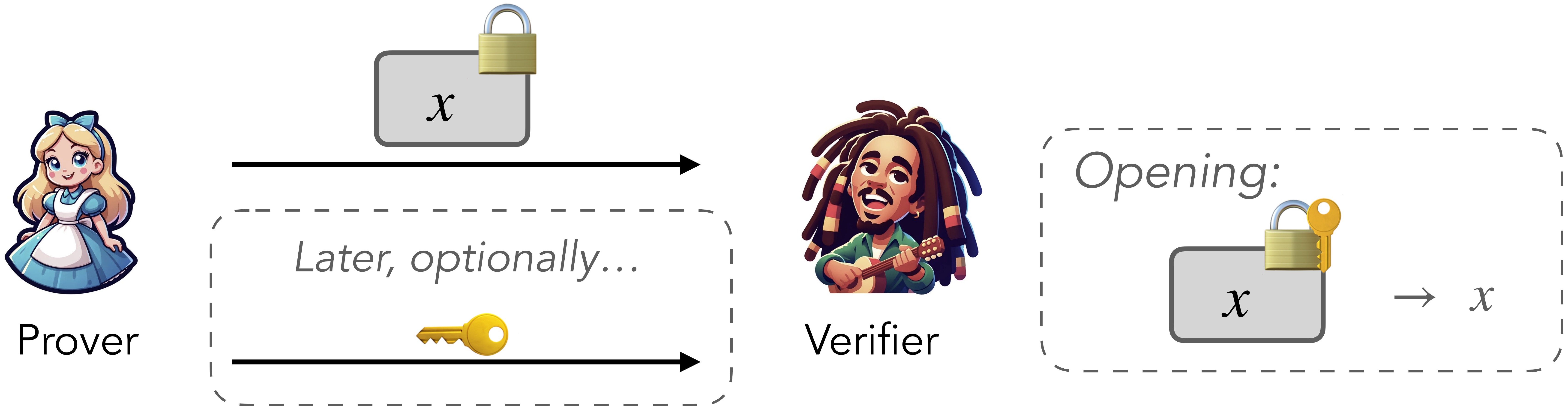
Q. Additive sharing is ℓ-private for which ℓ ?

A. Additive sharing is $(N-1)$-private.

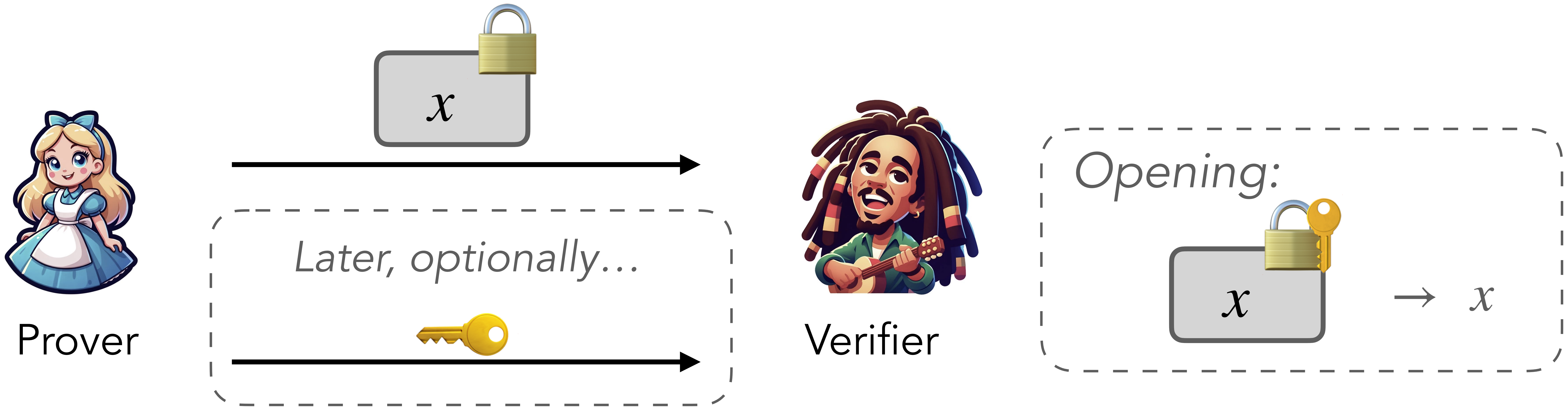
Commitment Scheme

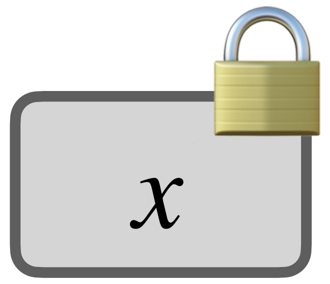


Commitment Scheme

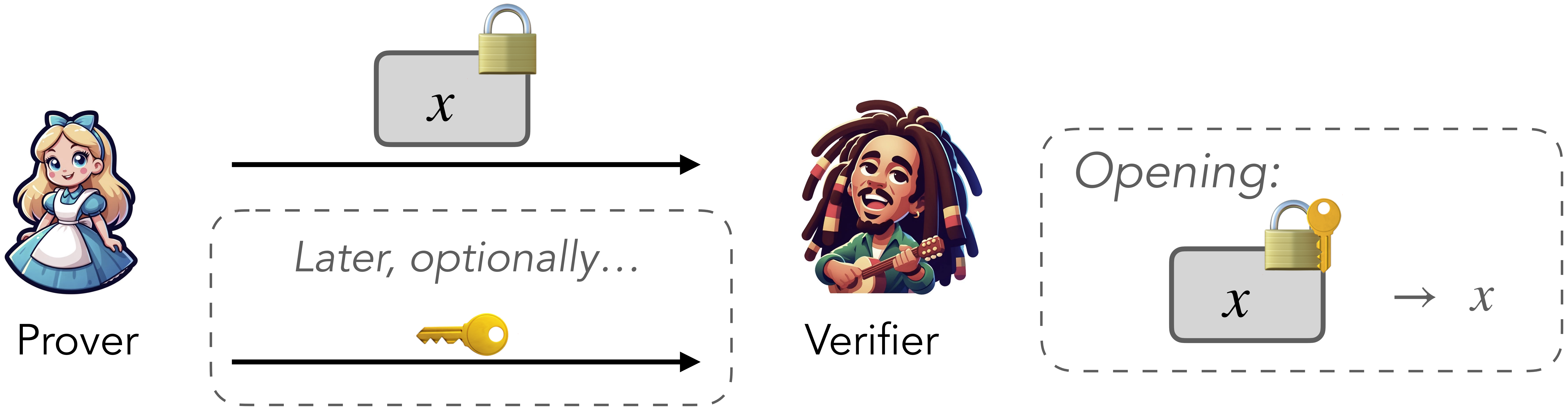


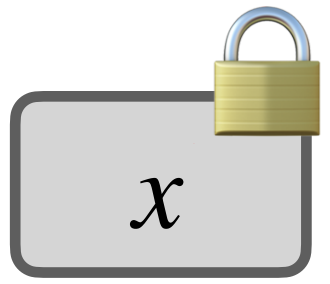
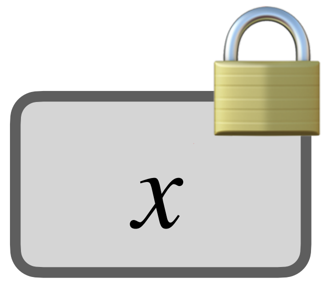
Commitment Scheme



- **Binding:** no way  can be opened to $x' \neq x$

Commitment Scheme



- **Binding:** no way  can be opened to $x' \neq x$
- **Hiding:**  does not reveal information about x (without )

Question 7

Question 7

Q. How to construct a simple binding and hiding commitment scheme using symmetric cryptography?

Question 7

Q. How to construct a simple binding and hiding commitment scheme using symmetric cryptography?

A. **Hash commitment:**

$$\boxed{x}^{🔒} := \mathrm{Hash}(x \,\|\, \rho) \quad \text{with} \quad \rho \leftarrow \$ \qquad 🔑 := (x, \rho)$$

Question 7

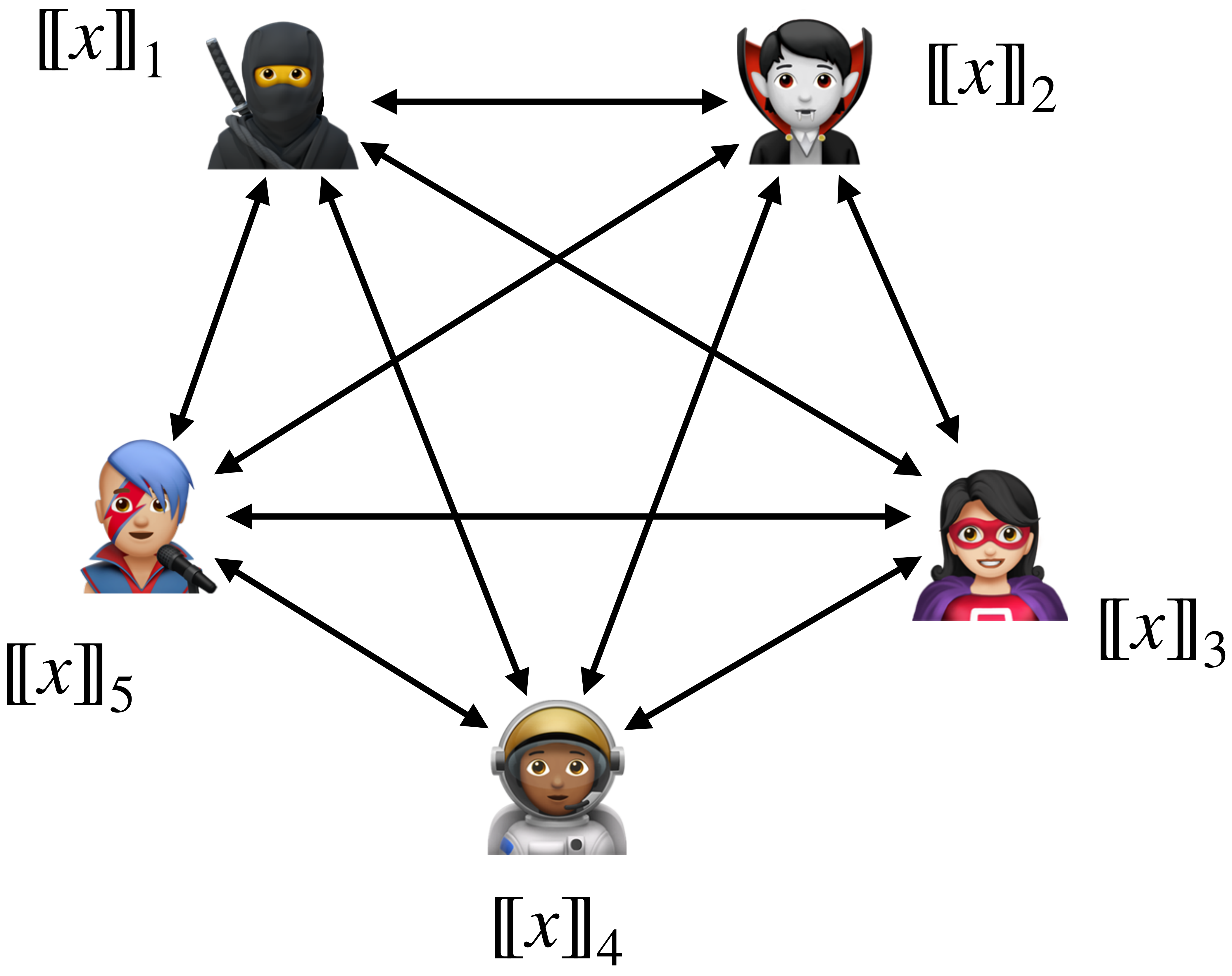
Q. How to construct a simple binding and hiding commitment scheme using symmetric cryptography?

A. **Hash commitment:**

$\boxed{x}$🔒 $:= \mathrm{Hash}(x \,\|\, \rho)$ with $\rho \leftarrow \$$ 🔑 $:= (x, \rho)$

- Biding by collision resistance
- Hiding in the ROM

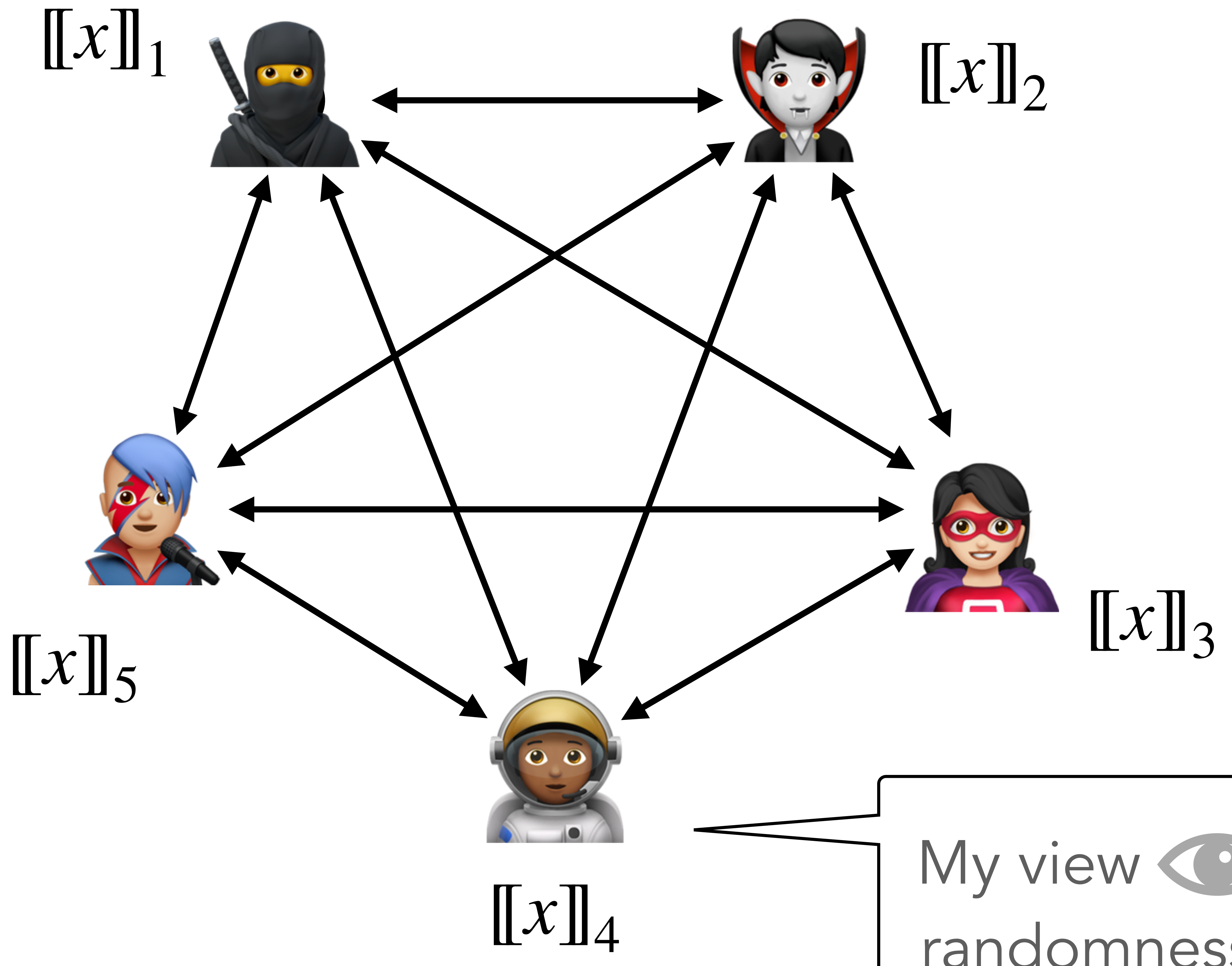
Multiparty Computation (MPC) Protocol



- **Input:** the parties receive a sharing $[\![x]\!]$
- **MPC:** the parties jointly compute

$$y = C(x)$$

Multiparty Computation (MPC) Protocol



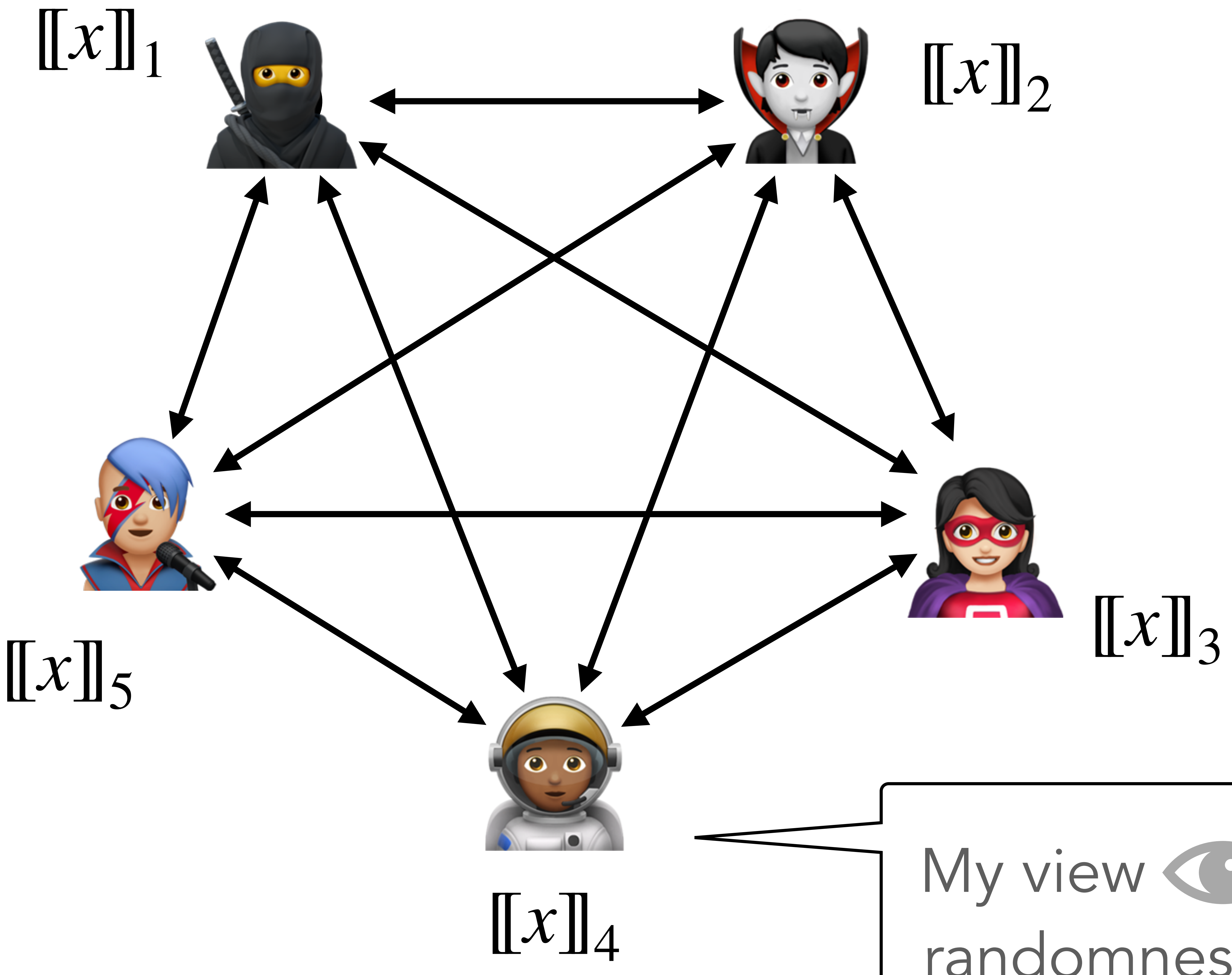
- **Input:** the parties receive a sharing $[\![x]\!]$
- **MPC:** the parties jointly compute

$$y = C(x)$$

- **ℓ-privacy:** the views of any ℓ parties reveal no information on x

My view 👁 = my input share, my internal randomness and all the messages I receive

Multiparty Computation (MPC) Protocol



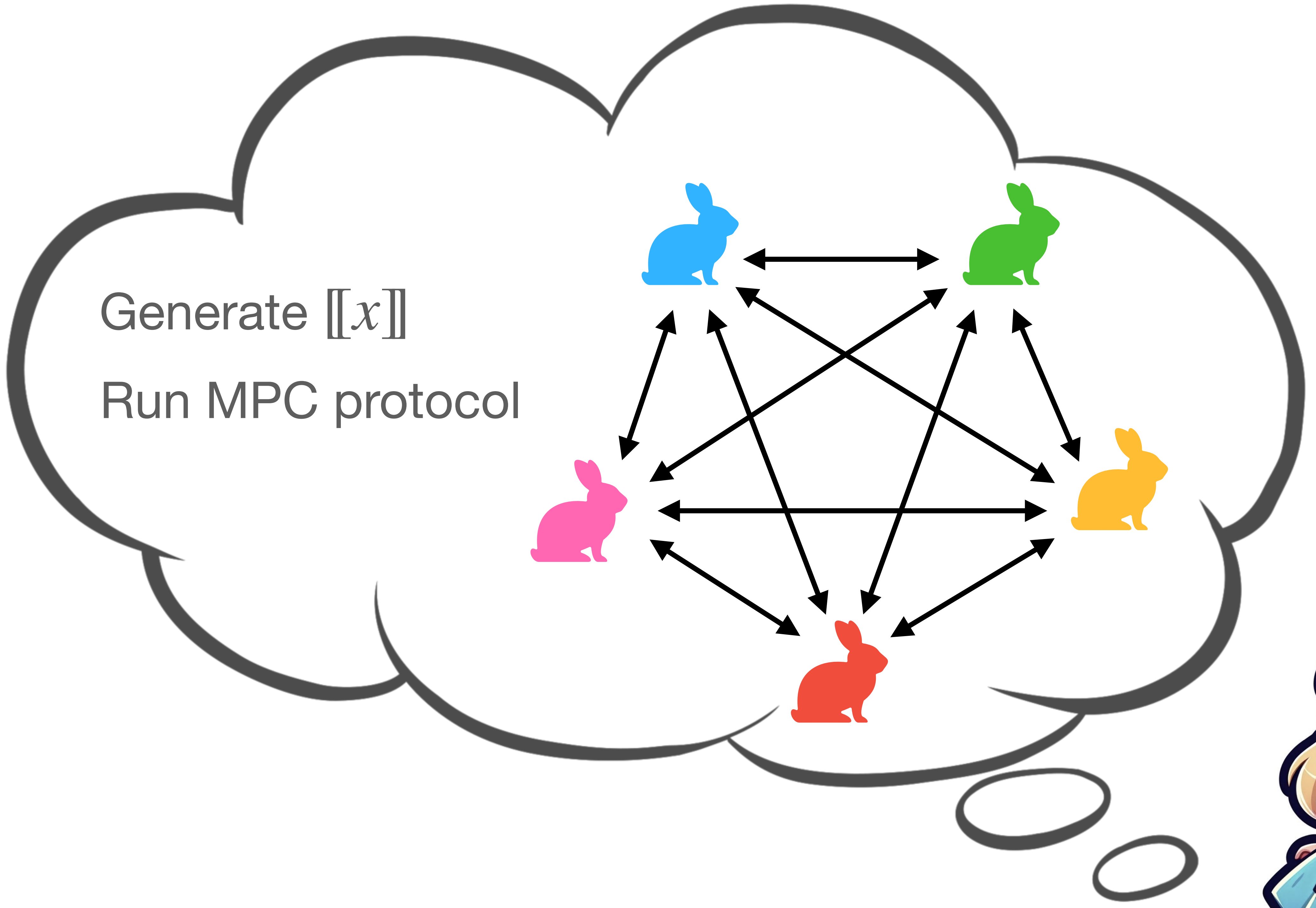
- **Input:** the parties receive a sharing $[\![x]\!]$
- **MPC:** the parties jointly compute

$$y = C(x)$$

- **ℓ-privacy:** the views of any ℓ parties reveal no information on x
- **Semi-honest model:** the parties follow the steps of the protocol

My view = my input share, my internal randomness and all the messages I receive

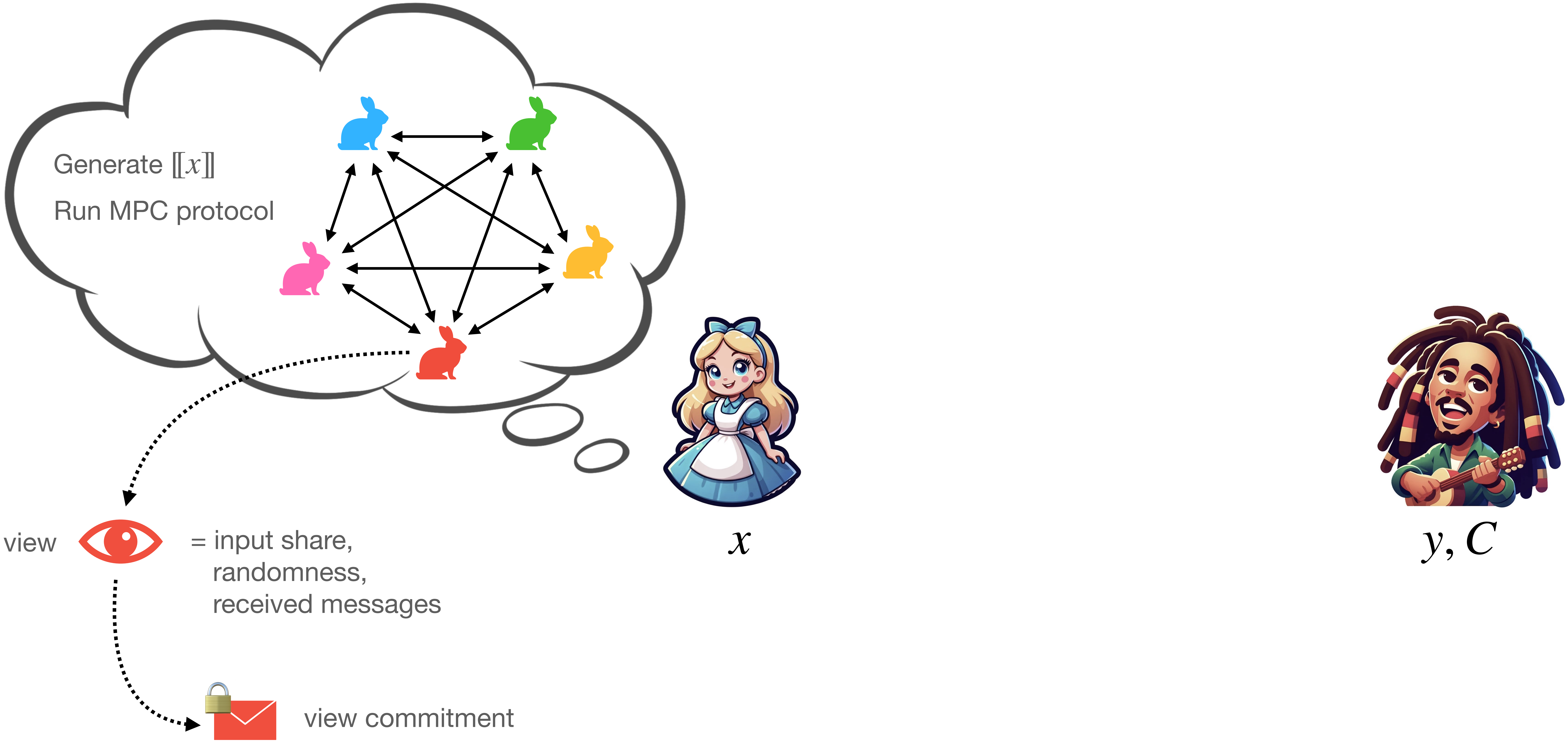
MPC in the Head



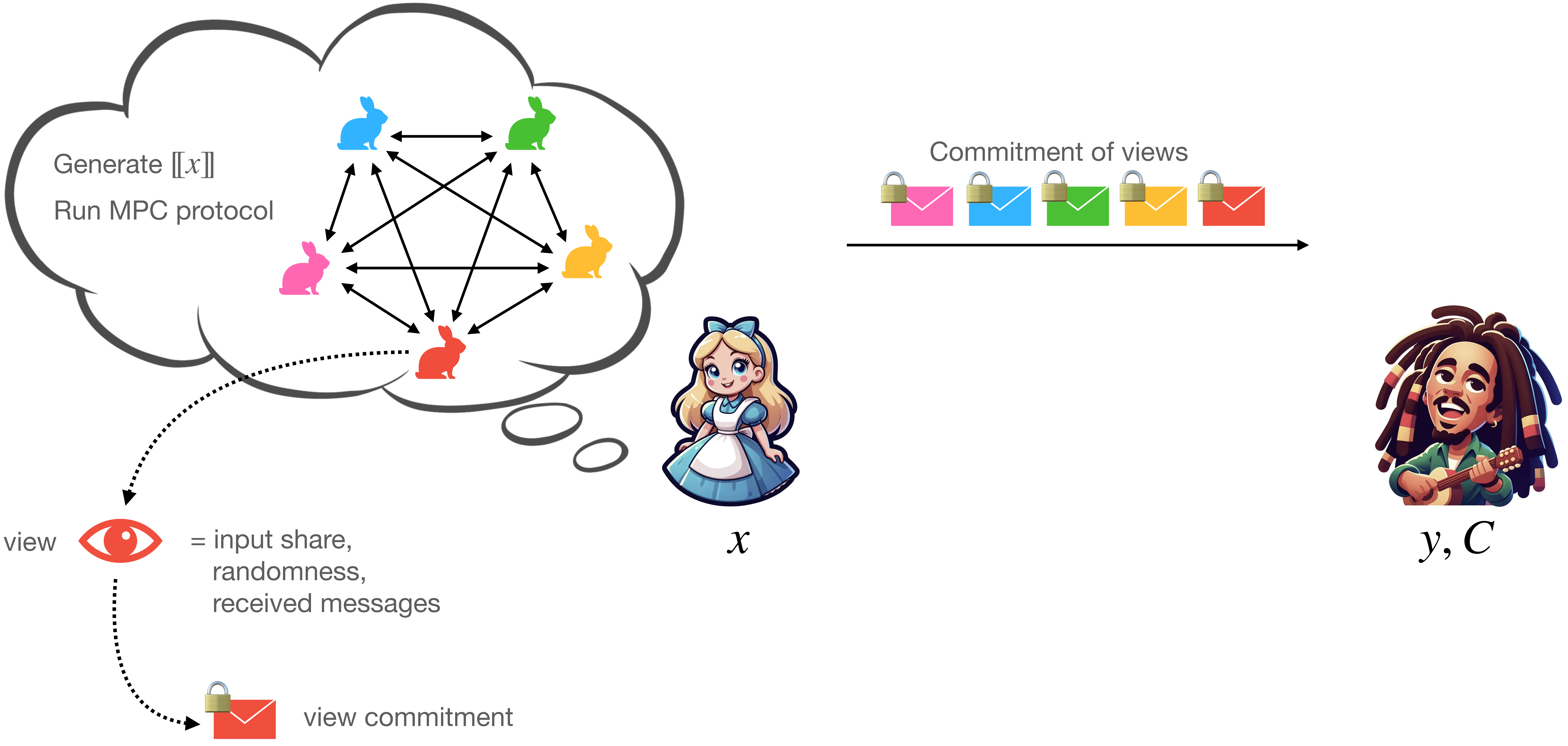
x

y, C

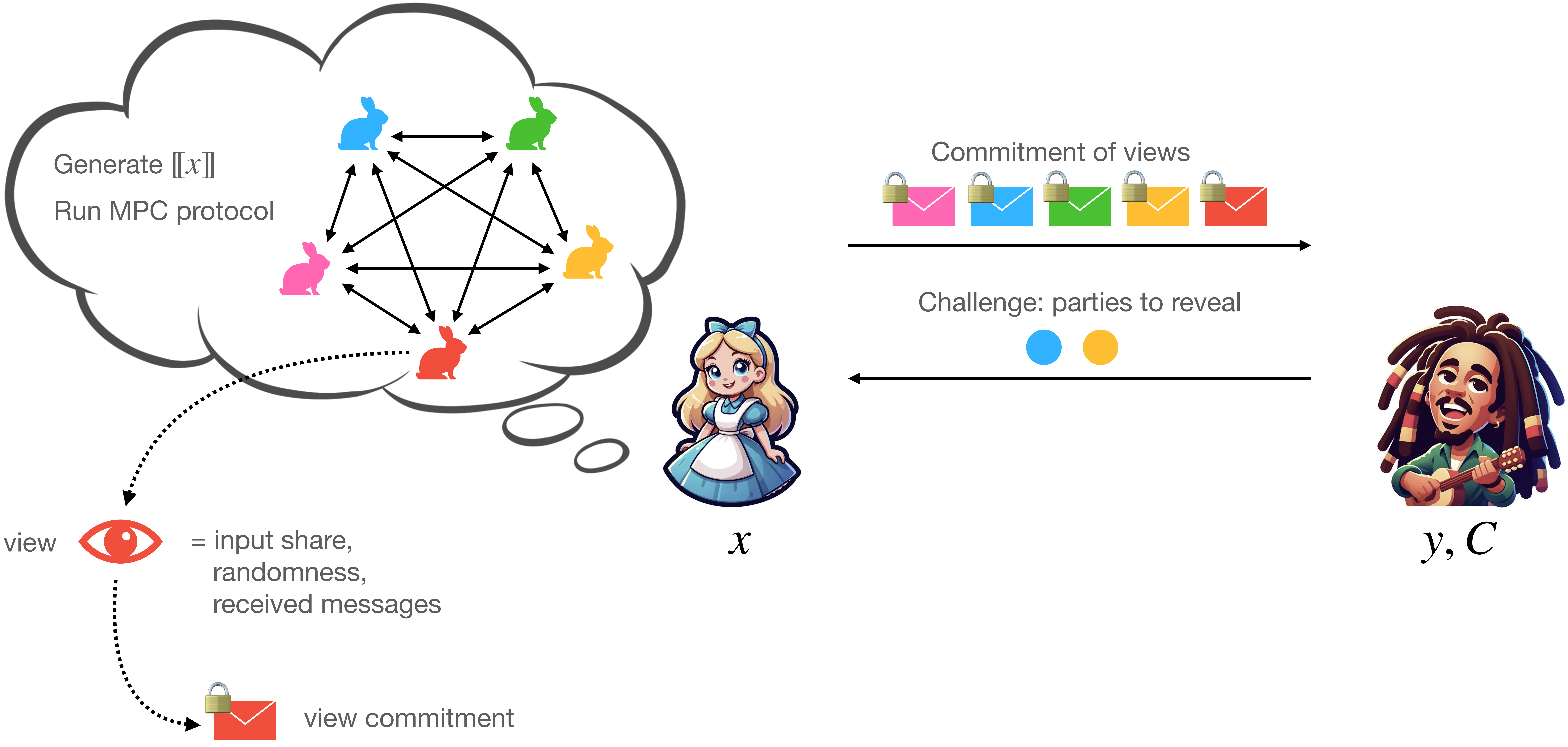
MPC in the Head



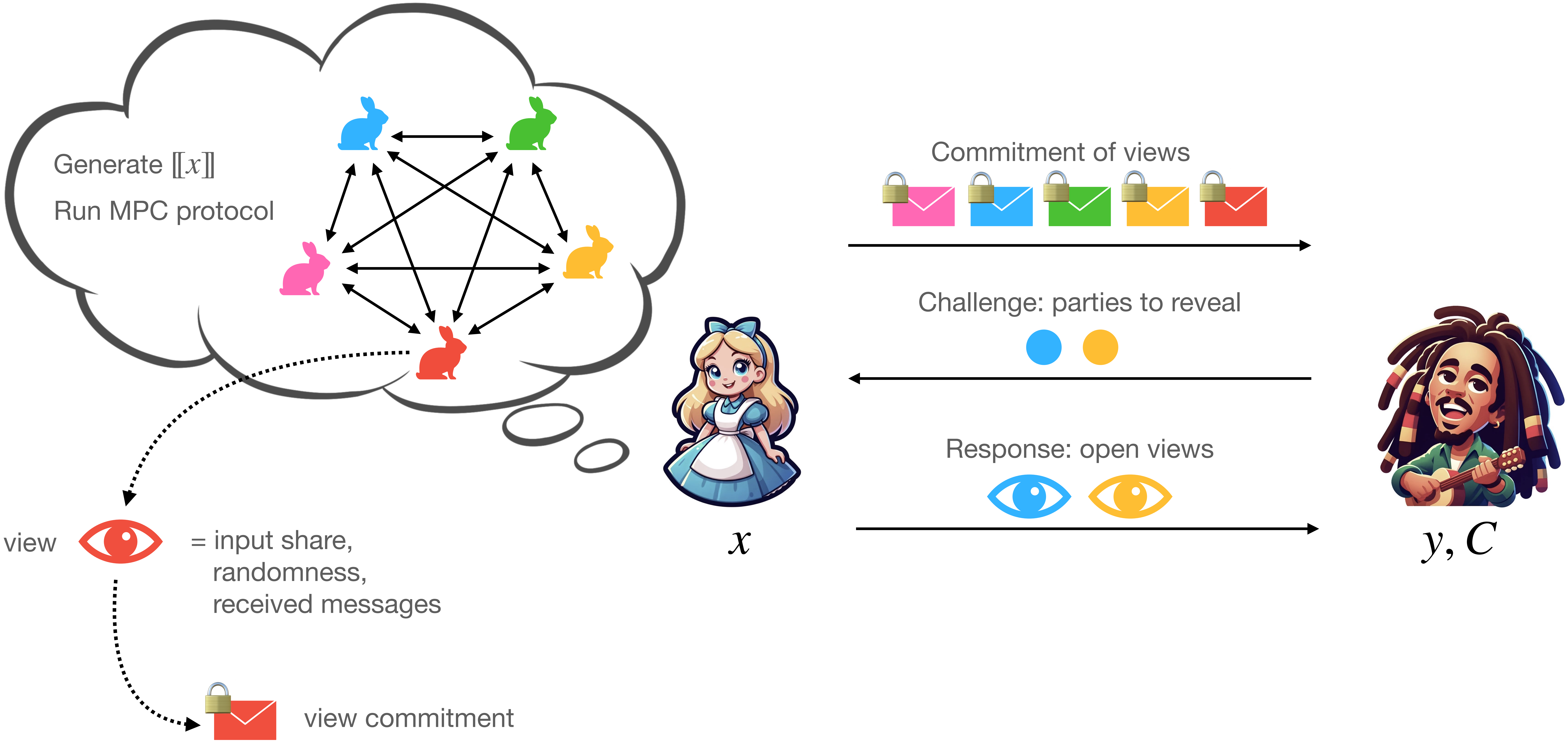
MPC in the Head



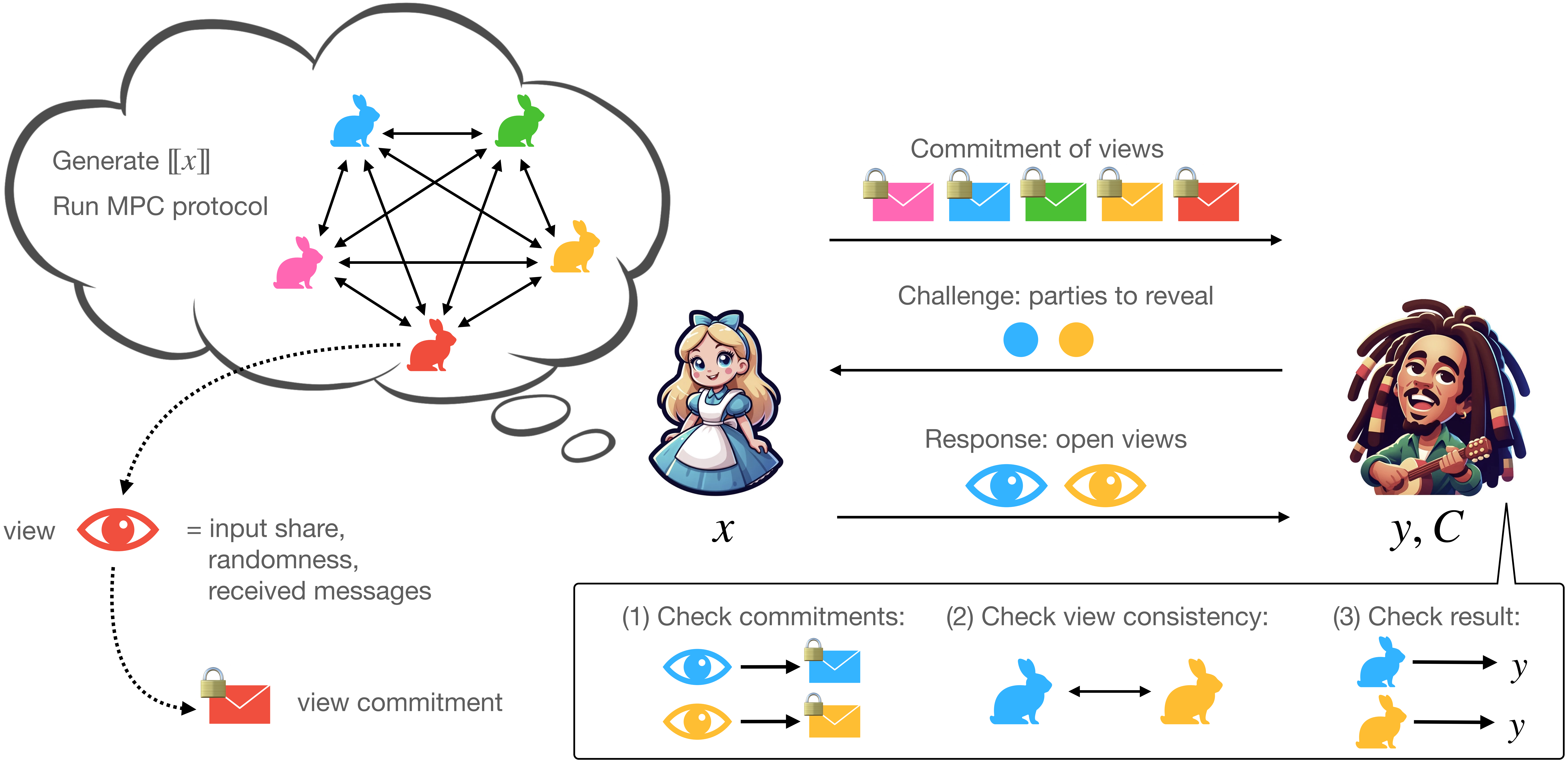
MPC in the Head



MPC in the Head

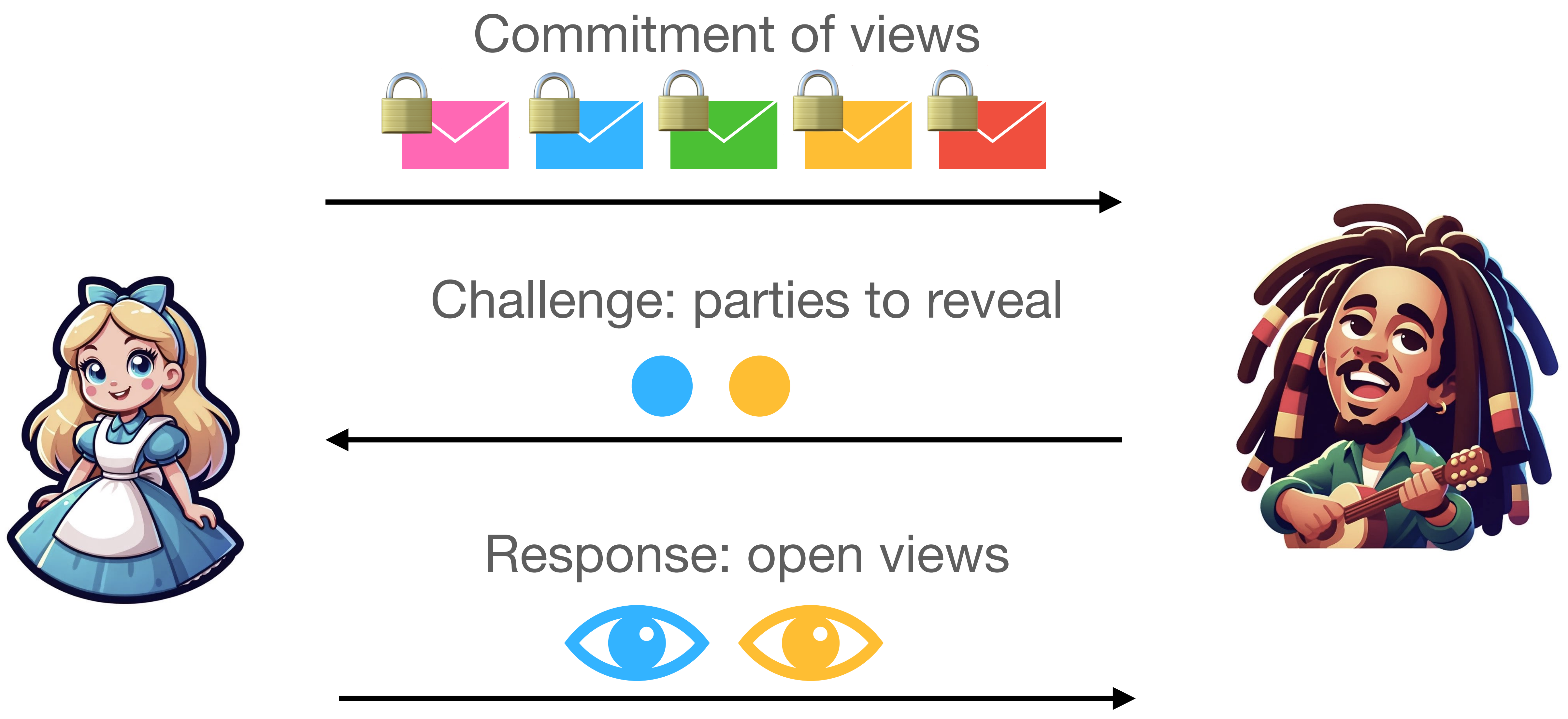


MPC in the Head



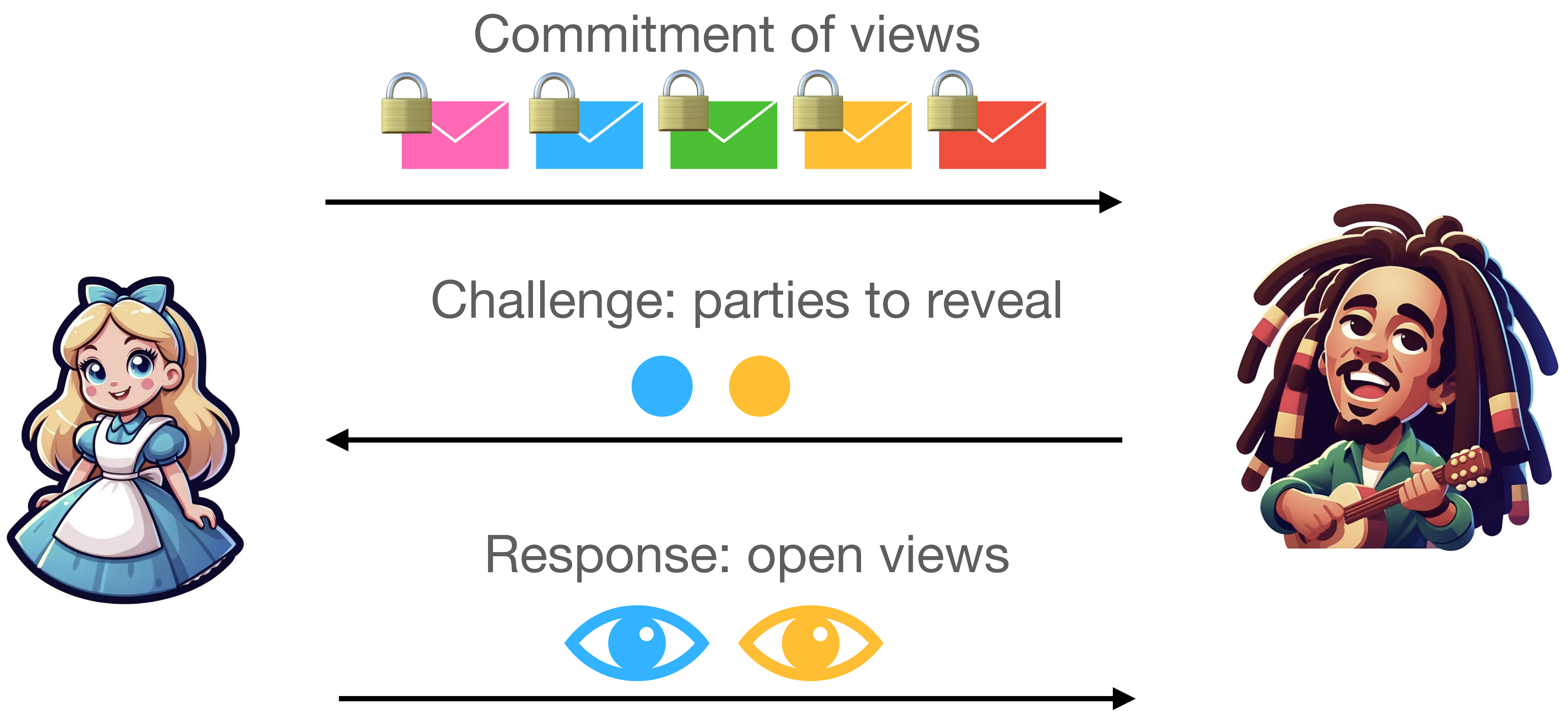
Question 8

Question 8



Q. This protocol is **zero-knowledge** if the MPC protocol is … ?

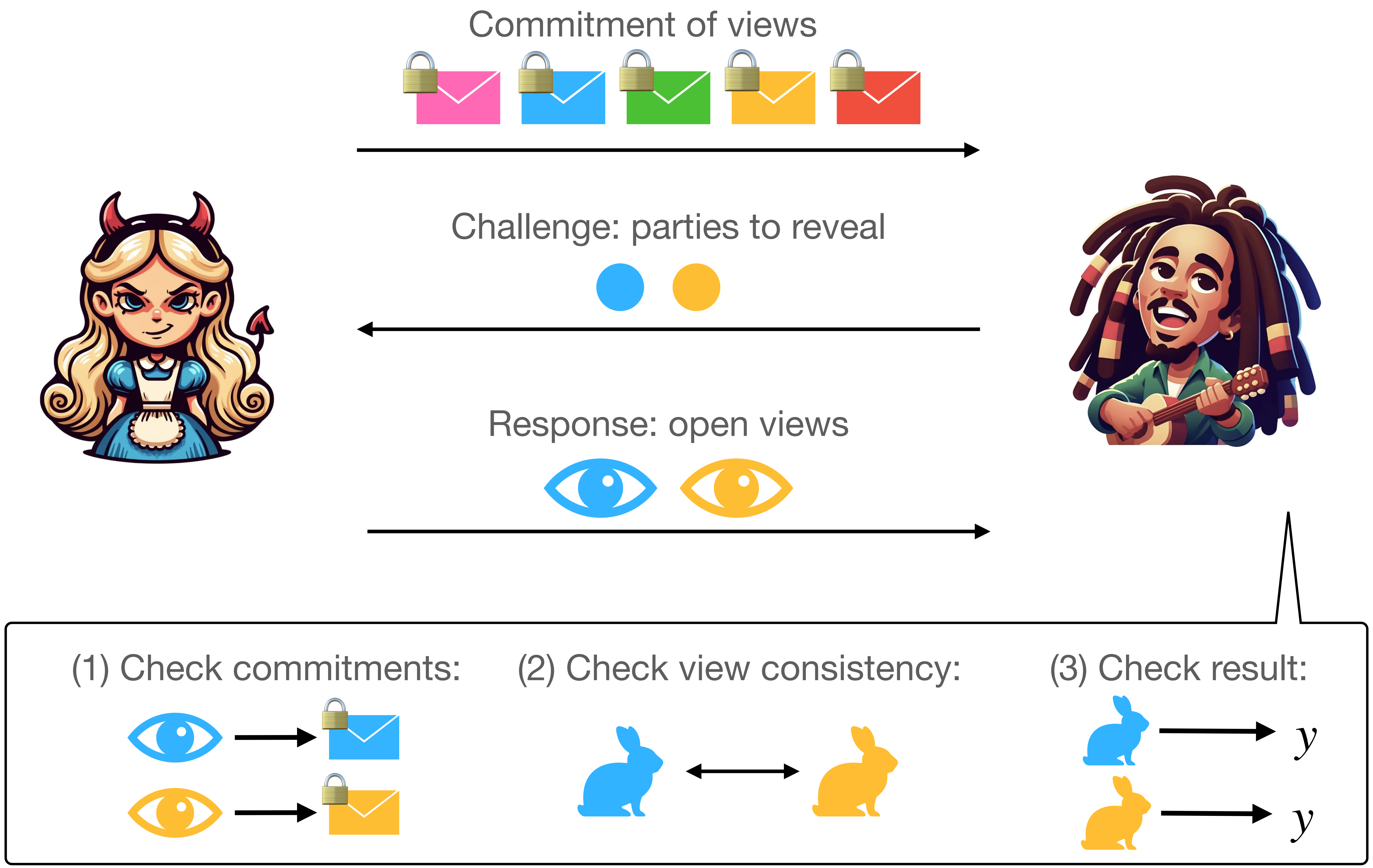
Question 8



Q. This protocol is **zero-knowledge** if the MPC protocol is … ?

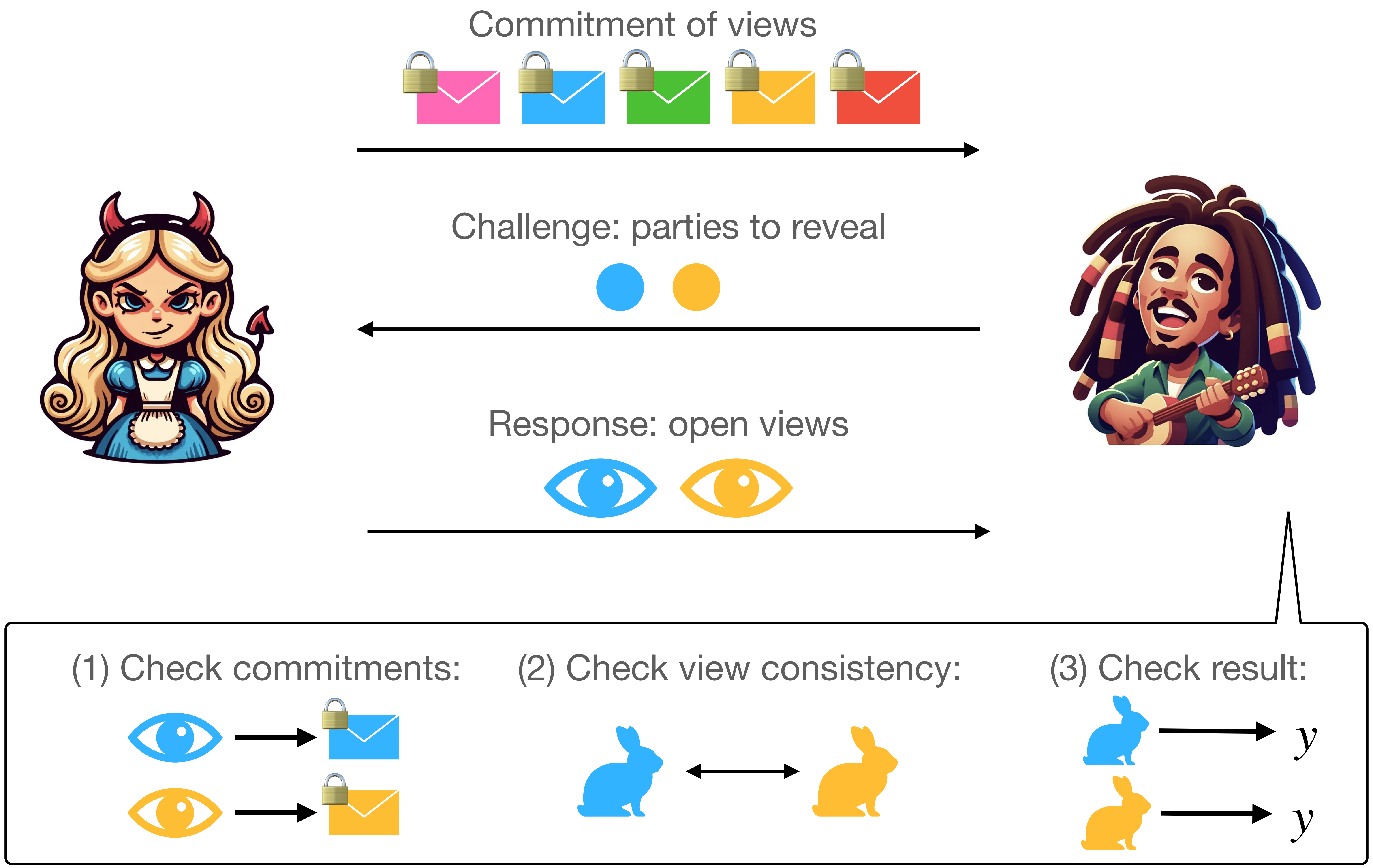
A. 2-private.

Soundness



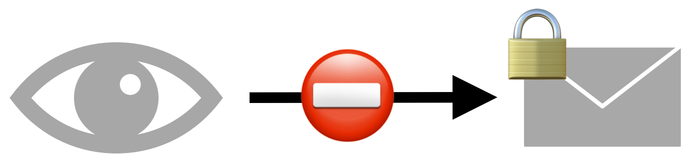
If  doesn't know x then parties receive $[\![\tilde{x}]\!]$ with $\tilde{x} \neq x$ and MPC$([\![\tilde{x}]\!]) \neq y$.

Soundness



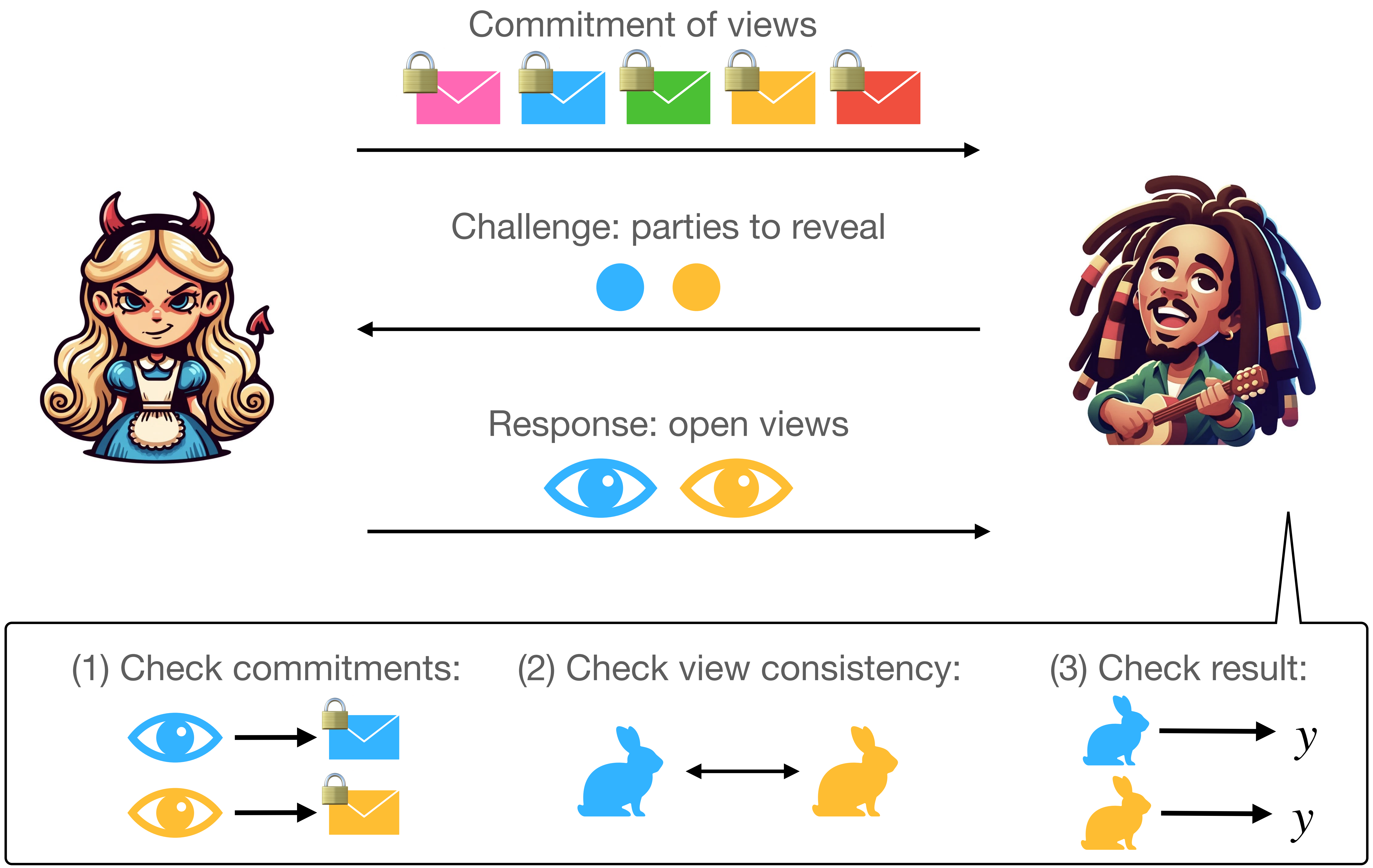
If  doesn't know x then parties receive $[\![\tilde{x}]\!]$ with $\tilde{x} \neq x$ and MPC$([\![\tilde{x}]\!]) \neq y$.

Therefore either

(1) 

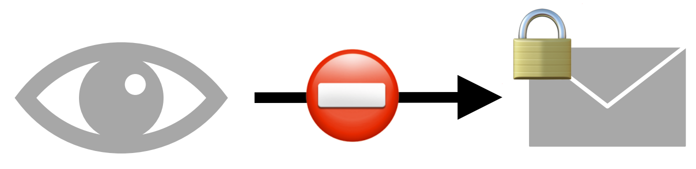
for some party

Soundness

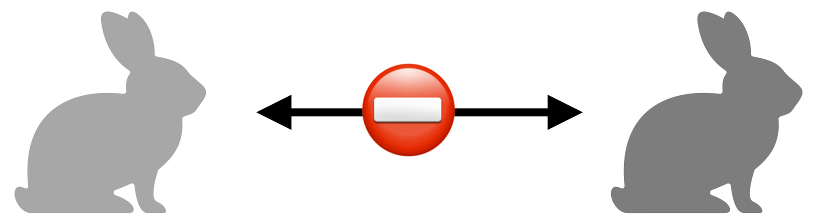


If  doesn't know x then parties receive $[\![\tilde{x}]\!]$ with $\tilde{x} \neq x$ and MPC($[\![\tilde{x}]\!]$) $\neq y$.

Therefore either

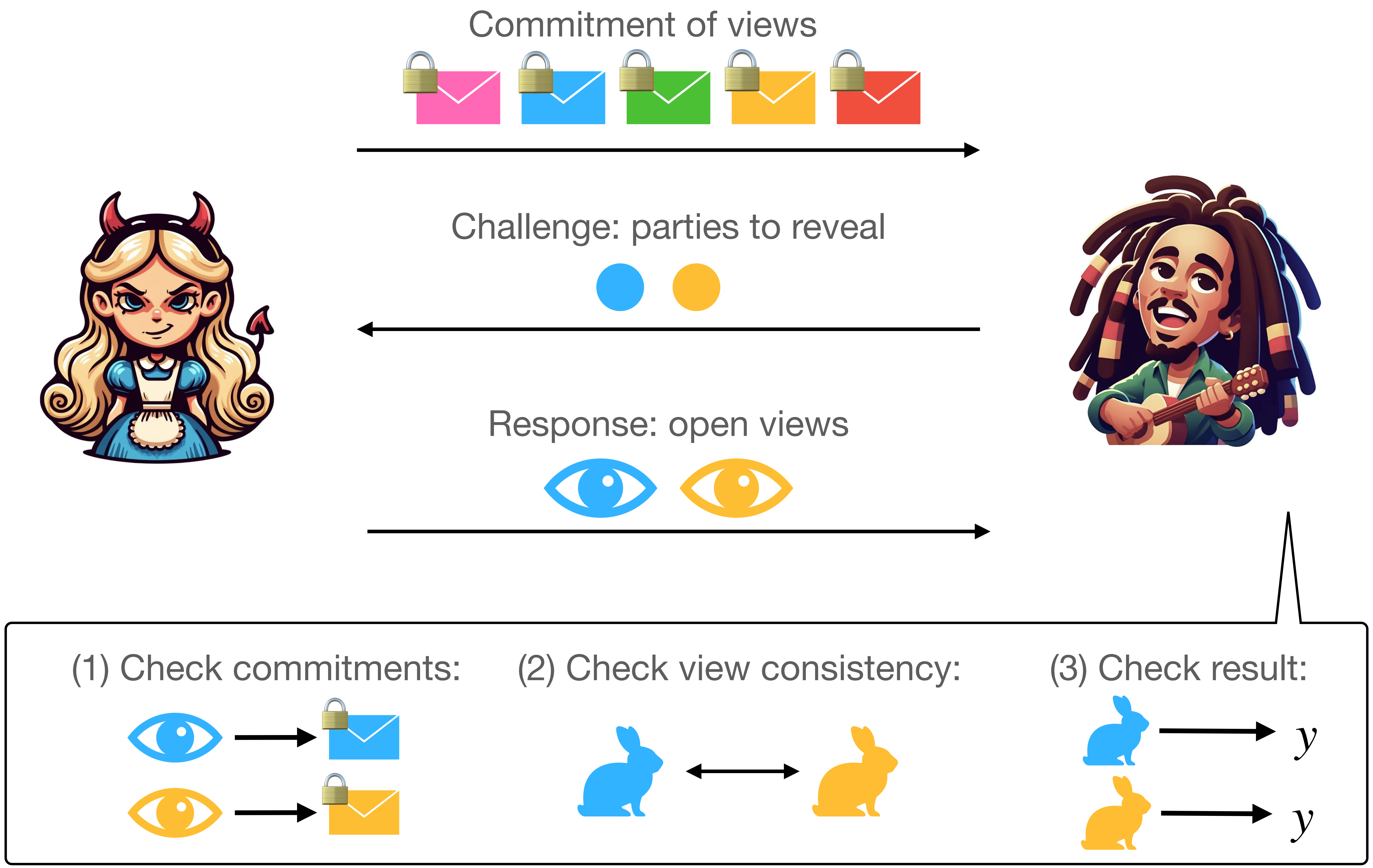
(1) 

for some party

(2) 

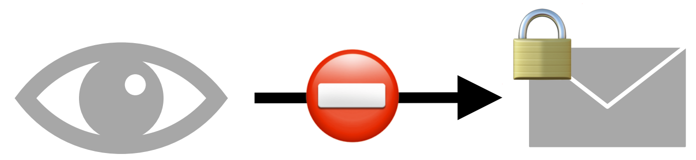
for two parties

Soundness

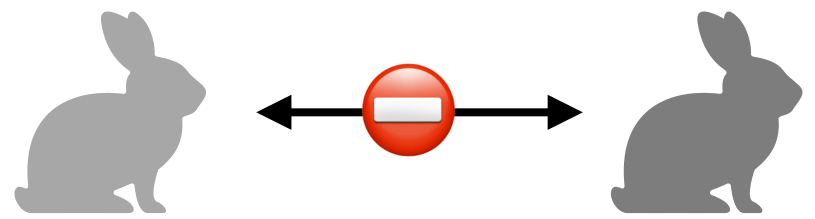


If  doesn't know x then parties receive $[\![\tilde{x}]\!]$ with $\tilde{x} \neq x$ and MPC($[\![\tilde{x}]\!]$) $\neq y$.

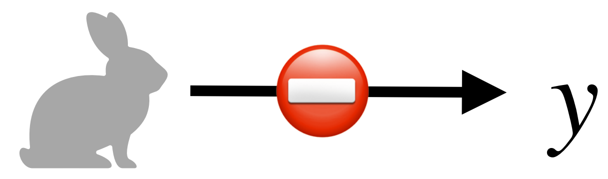
Therefore either

(1) 

for some party

(2) 

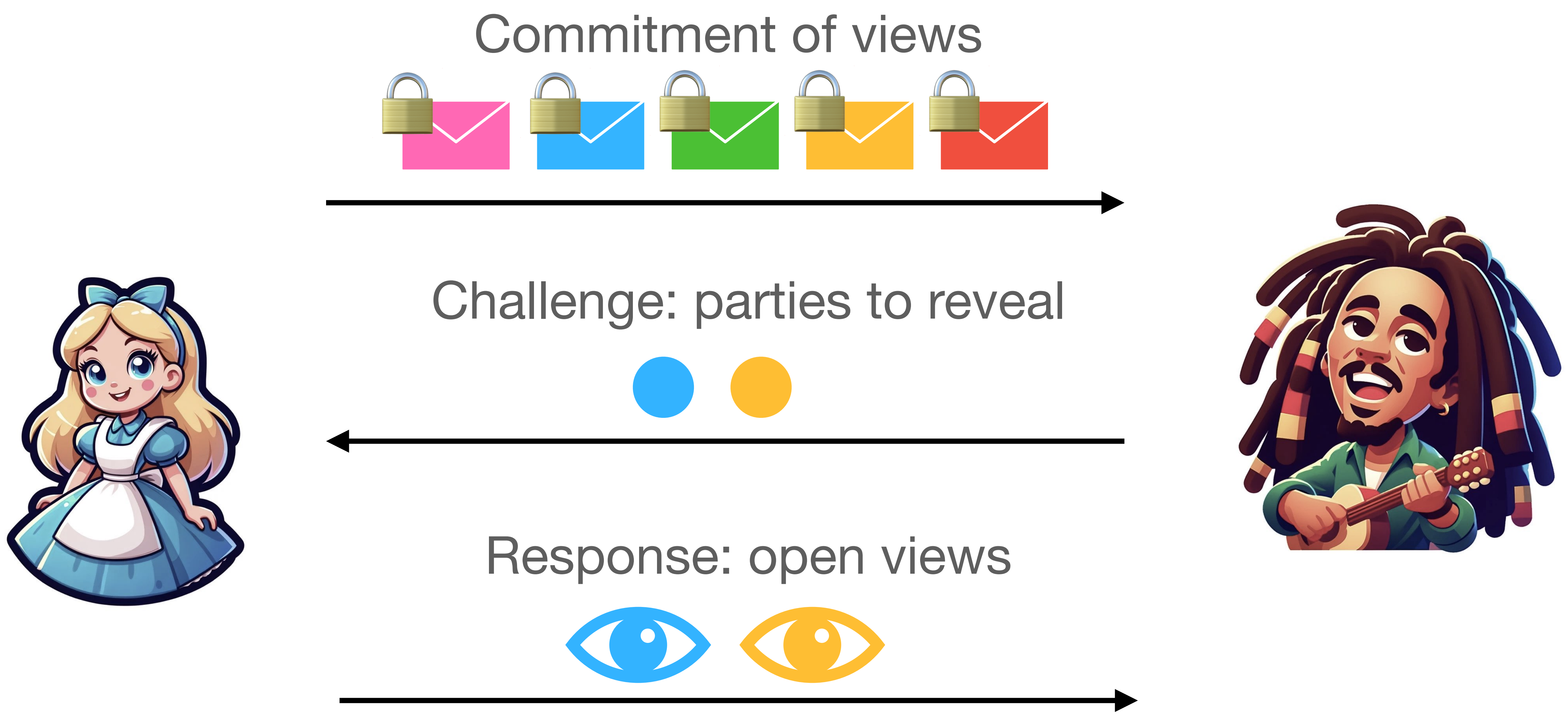
for two parties

(3) 

for some party

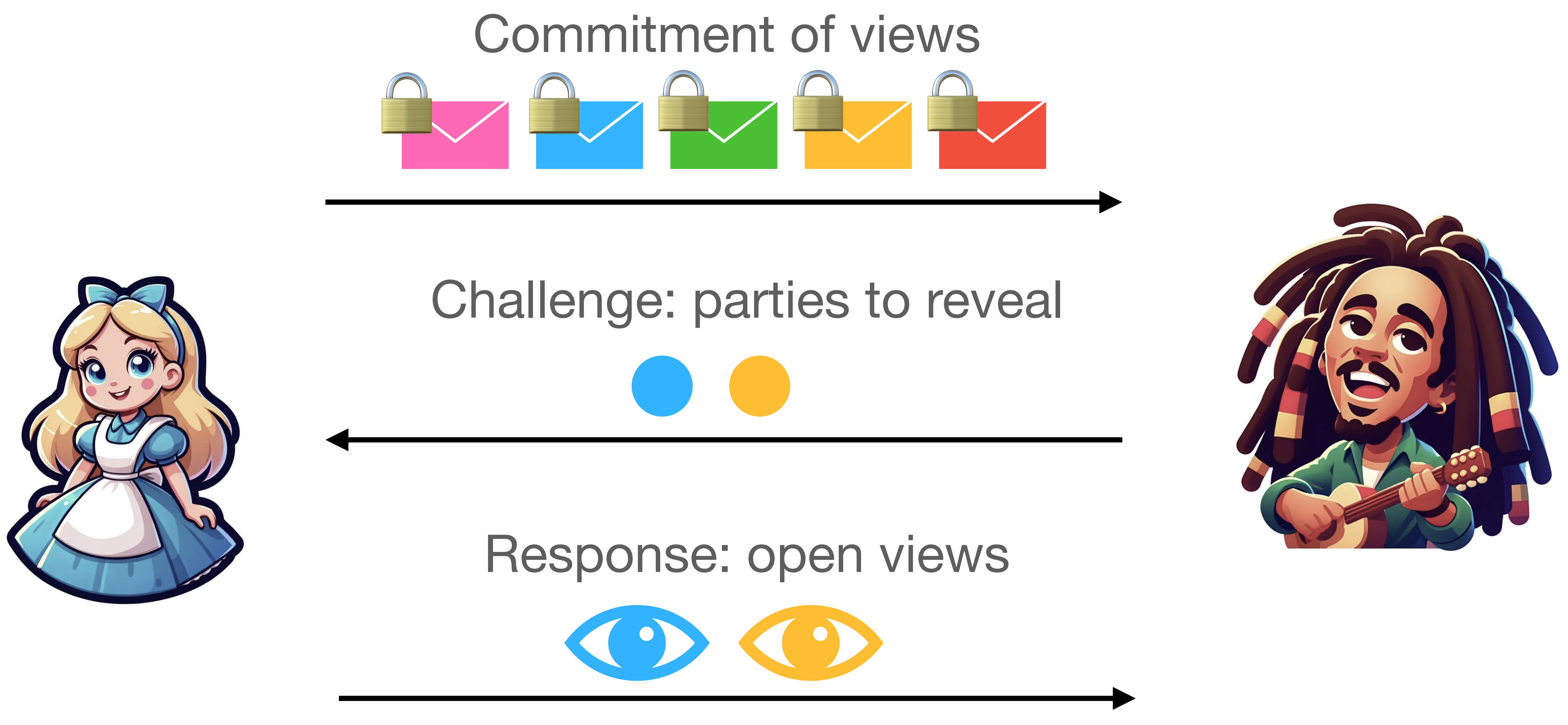
Question 9

Question 9

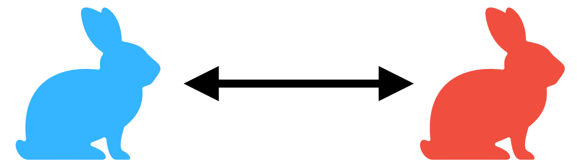


Q. What is the **soundness error** of this protocol?

Question 9



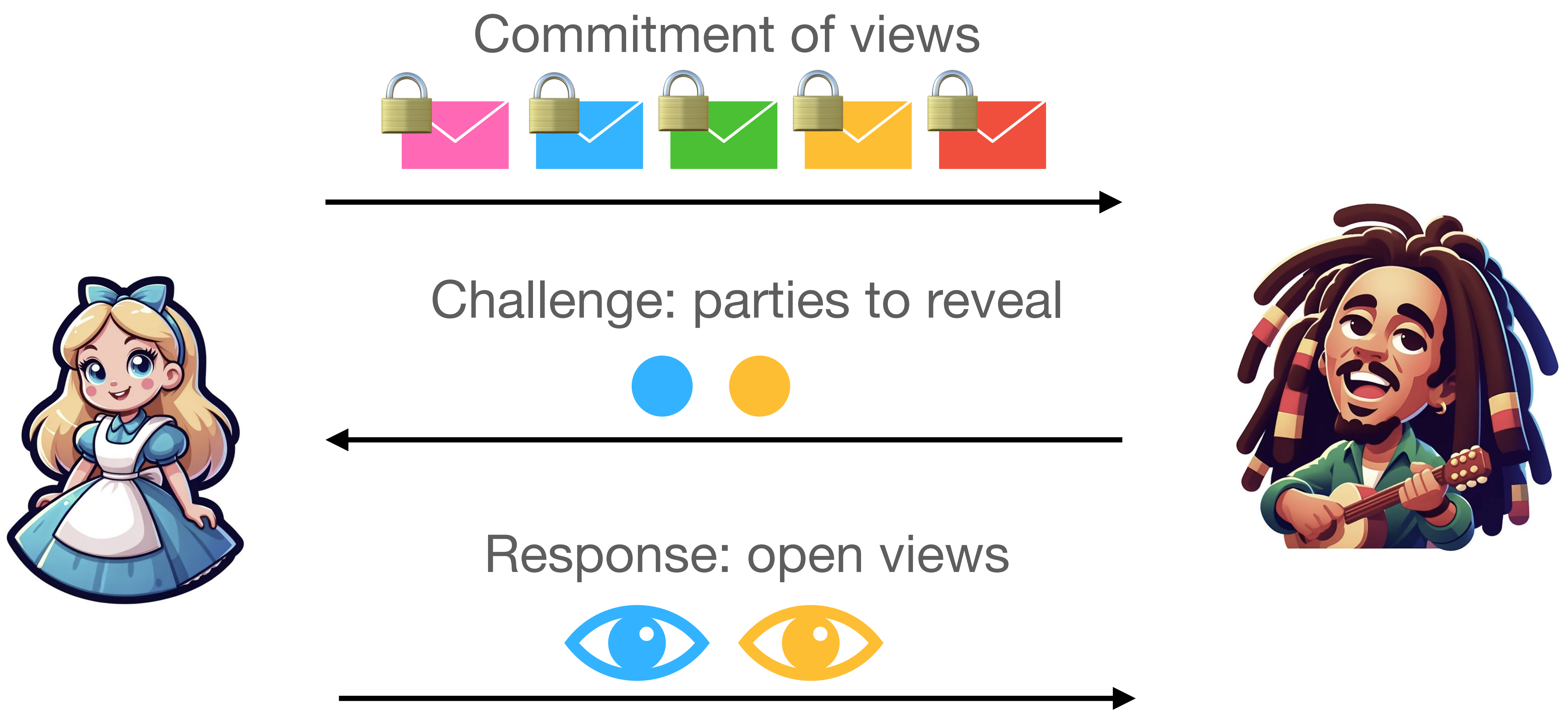
Q. What is the **soundness error** of this protocol?

A. If the prover cheat on a single message 

the verifier detects the cheat only if the challenge is

$$\text{Soundness error} = 1 - P[\text{detection}] = 1 - \frac{2}{N(N-1)}$$

Question 9



Q. What is the **soundness error** of this protocol?

A. If the prover cheat on a single message

the verifier detects the cheat only if the challenge is

Soundness error = $1 - P[\text{detection}] = 1 - \dfrac{2}{N(N-1)}$

We can do much better!
(See you tomorrow!)

MPC for Arithmetic Circuits

- Computation C composed of $(+)_{\mathbb{F}}$ and $(\times)_{\mathbb{F}}$

MPC for Arithmetic Circuits

- Computation C composed of $(+)_{\mathbb{F}}$ and $(\times)_{\mathbb{F}}$
- Additions → local computation

$$[\![x+y]\!] = \left([\![x]\!]_1 + [\![y]\!]_1,\ \ldots, [\![x]\!]_N + [\![y]\!]_N\right)$$

MPC for Arithmetic Circuits

- Computation C composed of $(+)_{\mathbb{F}}$ and $(\times)_{\mathbb{F}}$
- Additions → local computation

$$[\![x+y]\!] = \left([\![x]\!]_1 + [\![y]\!]_1,\ \ldots, [\![x]\!]_N + [\![y]\!]_N\right)$$

- Multiplications → require communication between parties

MPC for Arithmetic Circuits

- Computation C composed of $(+)_{\mathbb{F}}$ and $(\times)_{\mathbb{F}}$
- Additions → local computation

$$[\![x+y]\!] = \left([\![x]\!]_1 + [\![y]\!]_1,\ \ldots, [\![x]\!]_N + [\![y]\!]_N\right)$$

- Multiplications → require communication between parties
    - Common technique: using multiplication triples
    - Assume the parties have pre-generated/distributed random triples $[\![a]\!], [\![b]\!], [\![c]\!]$ such that $[\![c]\!] = [\![a \cdot b]\!]$

MPC for Arithmetic Circuits

- Multiplication of $[\![x]\!]$ and $[\![y]\!]$ using $[\![a]\!], [\![b]\!], [\![c]\!]$
  - Let $\alpha = x + a$ and $\beta = y + b$
  - We have

$$x \cdot y \;=\; (\alpha - a)(\beta - b) \;=\; \alpha\beta - \beta a - \alpha b + ab$$

  - Giving

$$[\![xy]\!] = \alpha\beta - \beta[\![a]\!] - \alpha[\![b]\!] + [\![c]\!]$$

MPC for Arithmetic Circuits

- Multiplication of $[\![x]\!]$ and $[\![y]\!]$ using $[\![a]\!], [\![b]\!], [\![c]\!]$
  - Let $\alpha = x + a$ and $\beta = y + b$
  - We have

$$x \cdot y \;=\; (\alpha - a)(\beta - b) \;=\; \alpha\beta - \beta a - \alpha b + ab$$

  - Giving

$$[\![xy]\!] = \alpha\beta - \beta[\![a]\!] - \alpha[\![b]\!] + [\![c]\!]$$

- Protocol:
  1. Parties locally compute $[\![\alpha]\!] = [\![x]\!] + [\![a]\!]$ and $[\![\beta]\!] = [\![y]\!] + [\![b]\!]$
  2. Parties broadcast $[\![\alpha]\!]$ and $[\![\beta]\!]$
  3. Parties reconstruct α and β and compute $[\![xy]\!]$ as above

MPC for Arithmetic Circuits

- Multiplication of $[\![x]\!]$ and $[\![y]\!]$ using $[\![a]\!], [\![b]\!], [\![c]\!]$
  - Let $\alpha = x + a$ and $\beta = y + b$
  - We have

$$x \cdot y \;=\; (\alpha - a)(\beta - b) \;=\; \alpha\beta - \beta a - \alpha b + ab$$

  - Giving

$$[\![xy]\!] = \alpha\beta - \beta[\![a]\!] - \alpha[\![b]\!] + [\![c]\!]$$

- Protocol:
  1. Parties locally compute $[\![\alpha]\!] = [\![x]\!] + [\![a]\!]$ and $[\![\beta]\!] = [\![y]\!] + [\![b]\!]$
  2. Parties broadcast $[\![\alpha]\!]$ and $[\![\beta]\!]$
  3. Parties reconstruct α and β and compute $[\![xy]\!]$ as above

😇 Compiling this protocol with MPCitH, we get a ZK PoK for $y = C(x)$.

MPC for Arithmetic Circuits

- Multiplication of $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$ using $\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket$
  - Let $\alpha = x + a$ and $\beta = y + b$
  - We have

$$x \cdot y \;=\; (\alpha - a)(\beta - b) \;=\; \alpha\beta - \beta a - \alpha b + ab$$

  - Giving

$$\llbracket xy \rrbracket = \alpha\beta - \beta\llbracket a \rrbracket - \alpha\llbracket b \rrbracket + \llbracket c \rrbracket$$

- Protocol:
  1. Parties locally compute $\llbracket \alpha \rrbracket = \llbracket x \rrbracket + \llbracket a \rrbracket$ and $\llbracket \beta \rrbracket = \llbracket y \rrbracket + \llbracket b \rrbracket$
  2. Parties broadcast $\llbracket \alpha \rrbracket$ and $\llbracket \beta \rrbracket$
  3. Parties reconstruct α and β and compute $\llbracket xy \rrbracket$ as above

😇 Compiling this protocol with MPCitH, we get a ZK PoK for $y = C(x)$.

🤔 Wait, what do you do for multiplication triples? (See you tomorrow!)