Composable Non-interactive Zero-knowledge Proofs in the Random Oracle Model

Yashvanth Kondi

AARHUS UNIVERSITY

Based on joint work with abhi shelat (Asiacrypt ’22)

In this talk…

- Zero-knowledge proofs (of knowledge)
  — Understand and use their security guarantees

- A taste for how they are designed and analysed
  — Provably secure composition
  — Random Oracle Model

- [Ks 22] Uncover a gap in the literature that was glossed over as folklore—turns out to permit a new kind of attack
  Briefly discussion on how we fix it

Quick Disclaimer

- **What will be covered**:
  Intuitive abstract idea of how to construct composition-safe ZK, how our attack works

- **What won't be touched**:
  Formalism of definitions, concrete instantiations, efficiency (this is to help understanding, not to hand-wave; please ask if something is unclear!)

Composable Non-interactive Zero-knowledge Proofs in the Random Oracle Model

Zero-knowledge Proofs

- Very powerful cryptographic primitive, introduced by [Goldwasser Micali Rackoff 85]

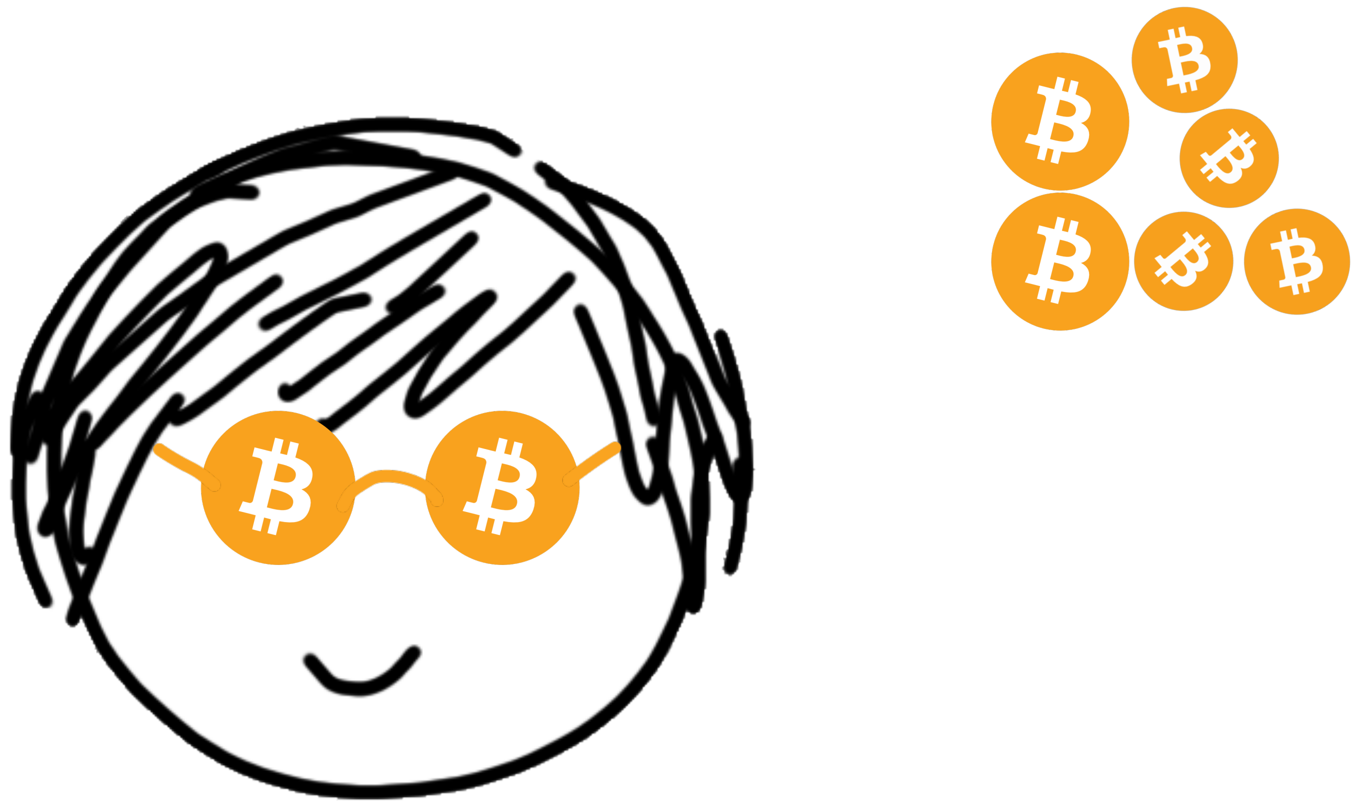
- Intuition: Prover convinces a Verifier of a statement, without revealing “why” it’s true.

  - Prover typically needs to use some secret information

  - Verifier obtains no useful information about Prover’s secrets

Zero-knowledge Proofs

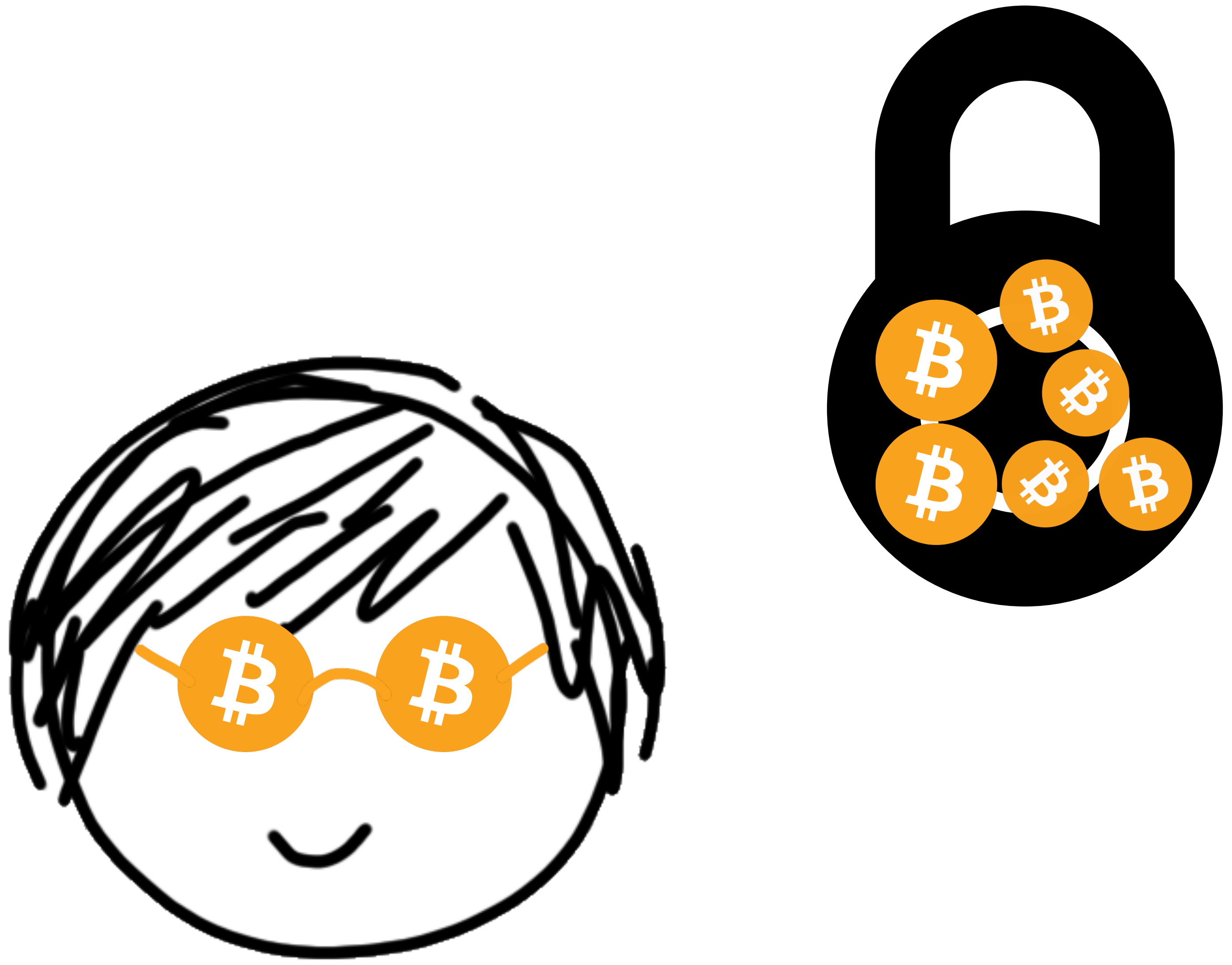
- Simple application: proof of possession (key ownership)



₿ob

Zero-knowledge Proofs

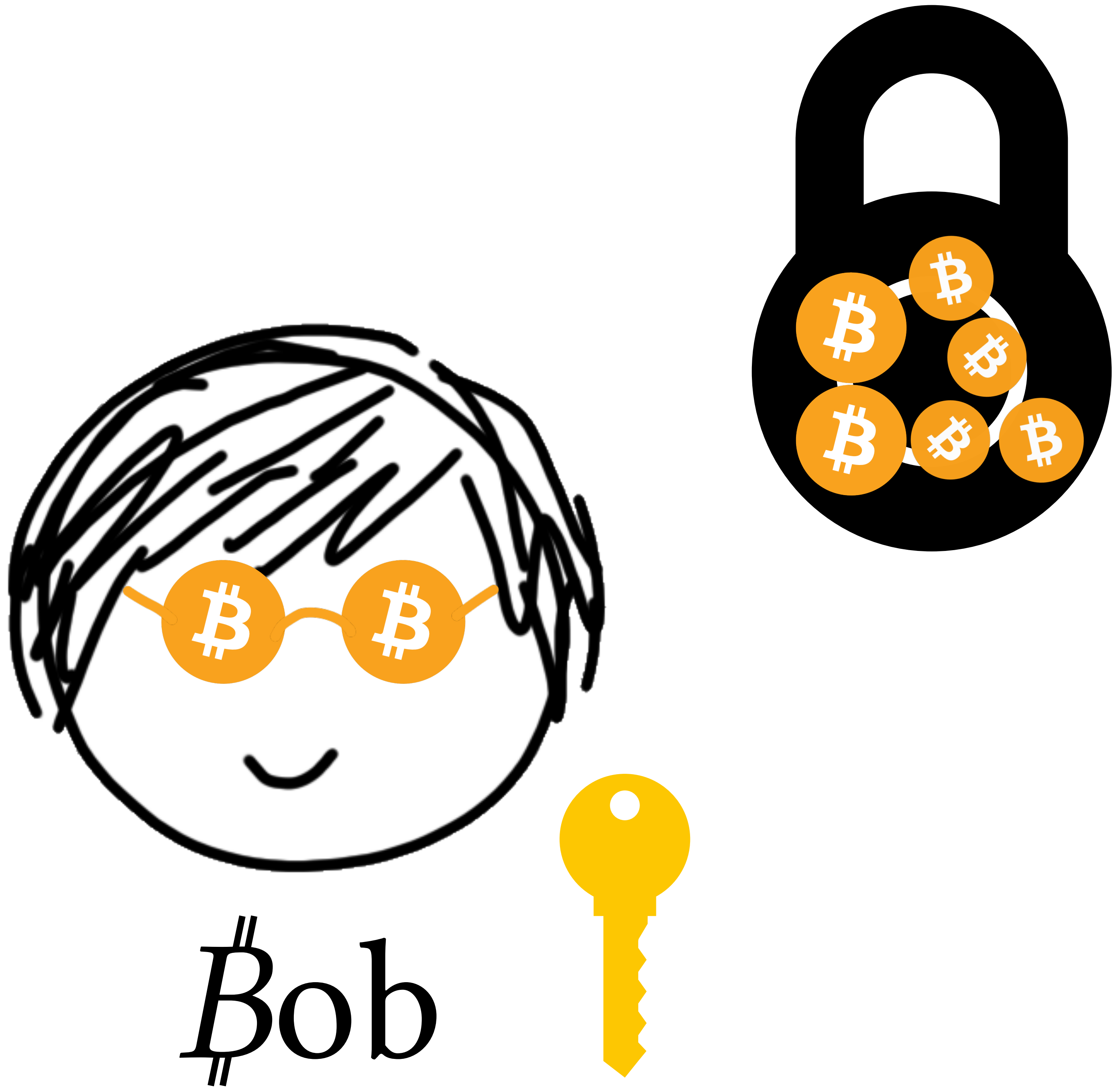
- Simple application: proof of possession (key ownership)



₿ob

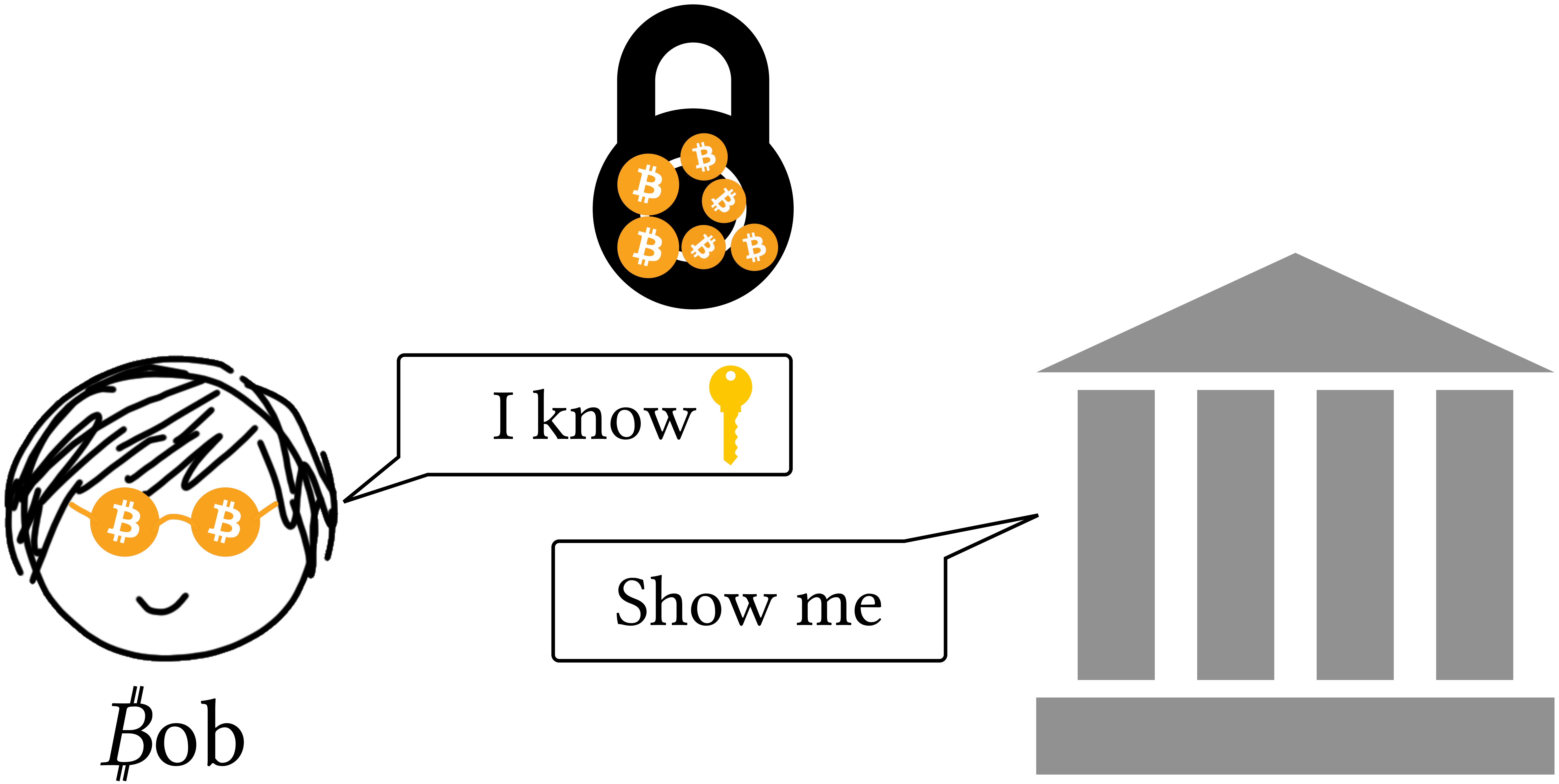
Zero-knowledge Proofs

- Simple application: proof of possession (key ownership)



Zero-knowledge Proofs

- Simple application: proof of possession (key ownership)



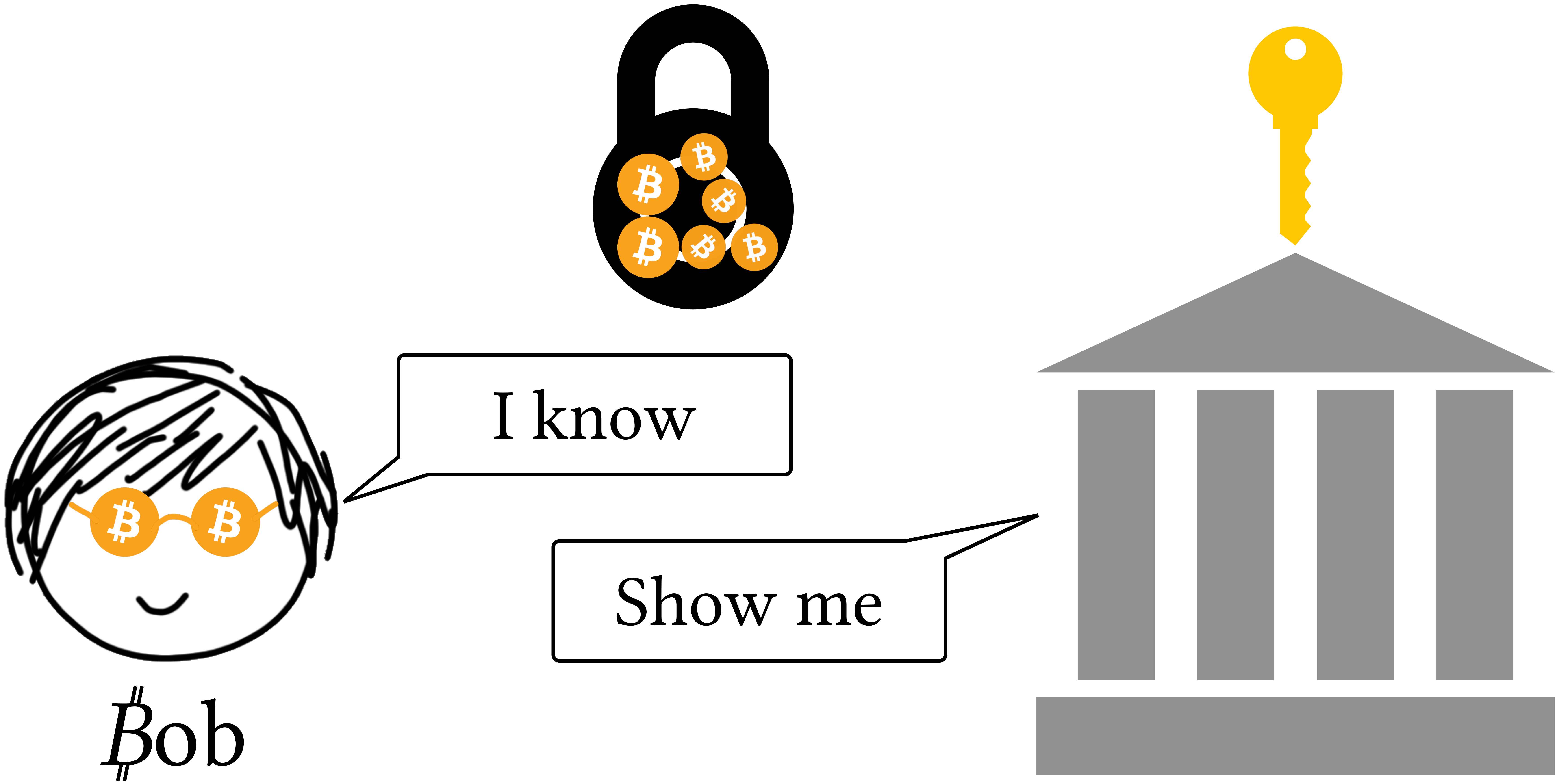
I know 🔑

Show me

Ƀob

Zero-knowledge Proofs

- Simple application: proof of possession (key ownership)



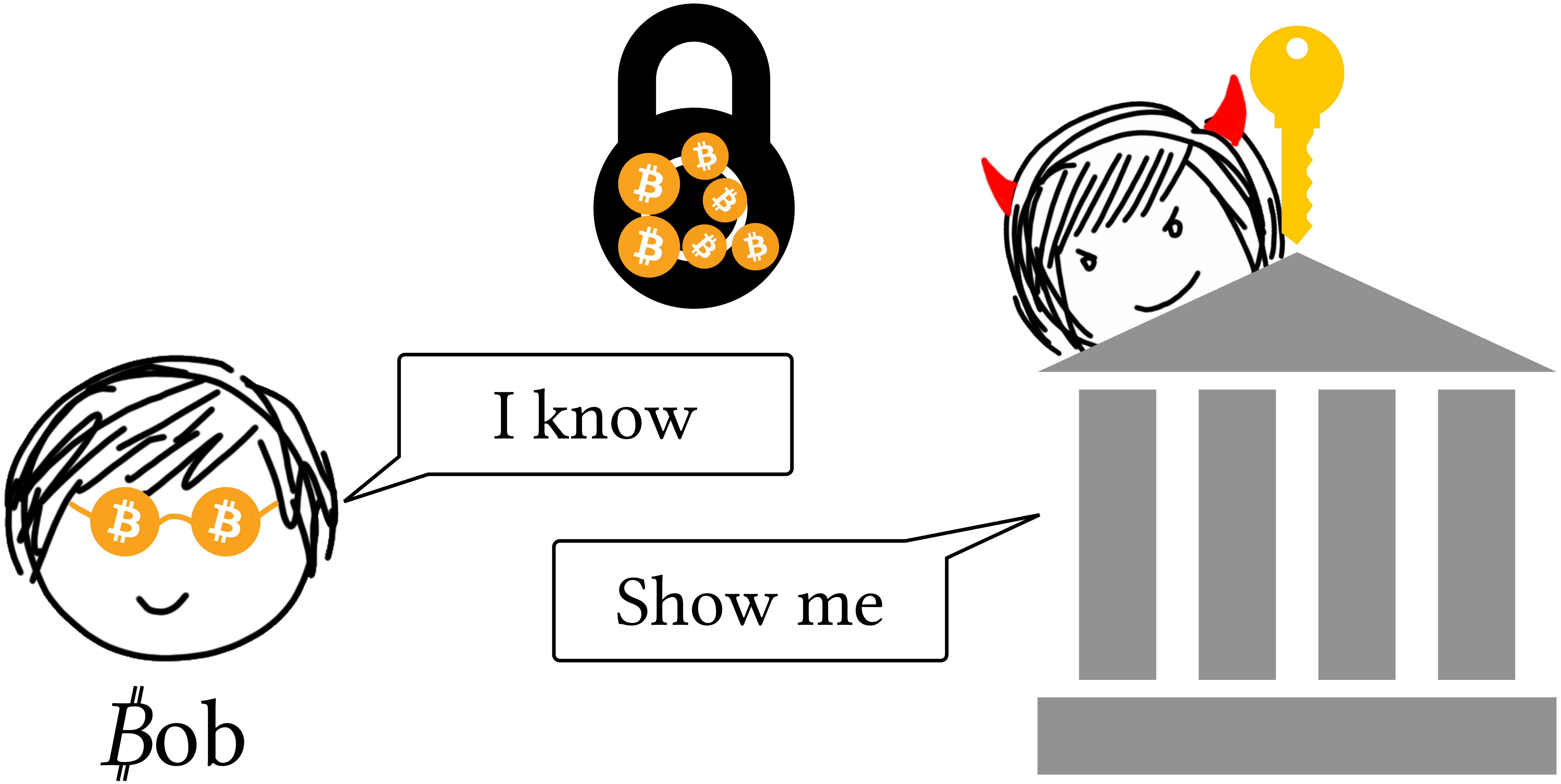
I know

Show me

₿ob

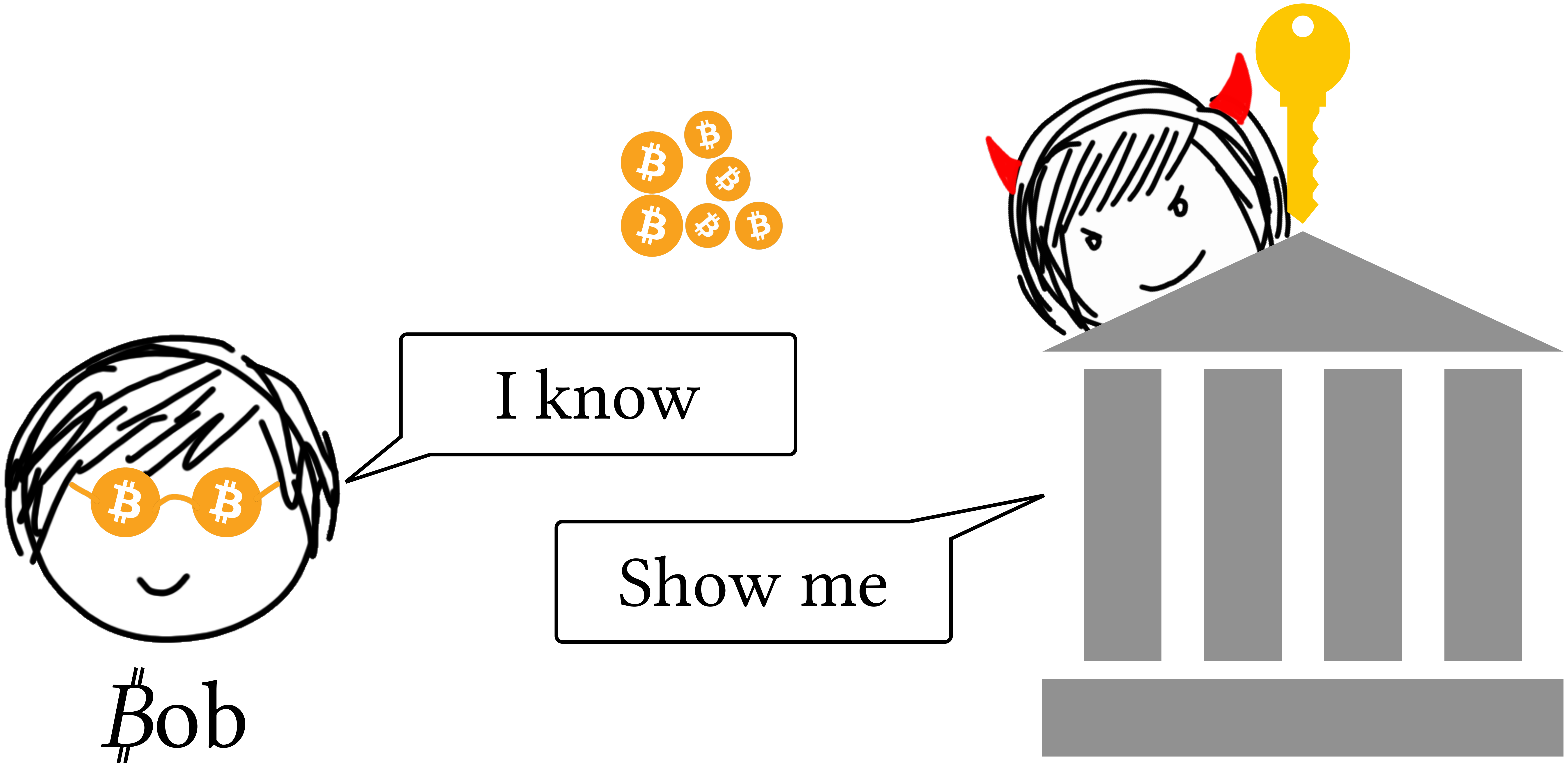
Zero-knowledge Proofs

- Simple application: proof of possession (key ownership)



Zero-knowledge Proofs

- Simple application: proof of possession (key ownership)



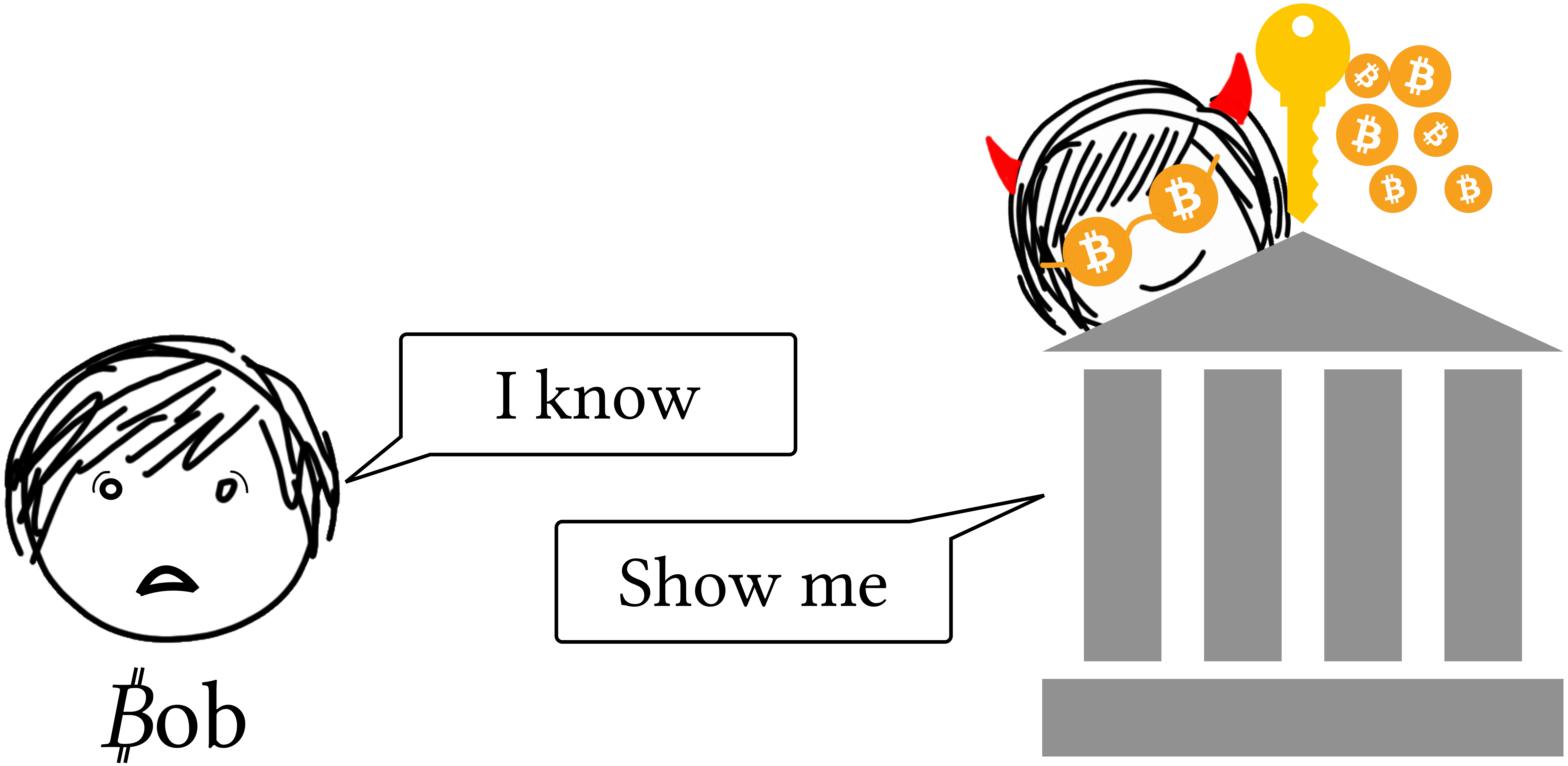
I know

Show me

Ƀob

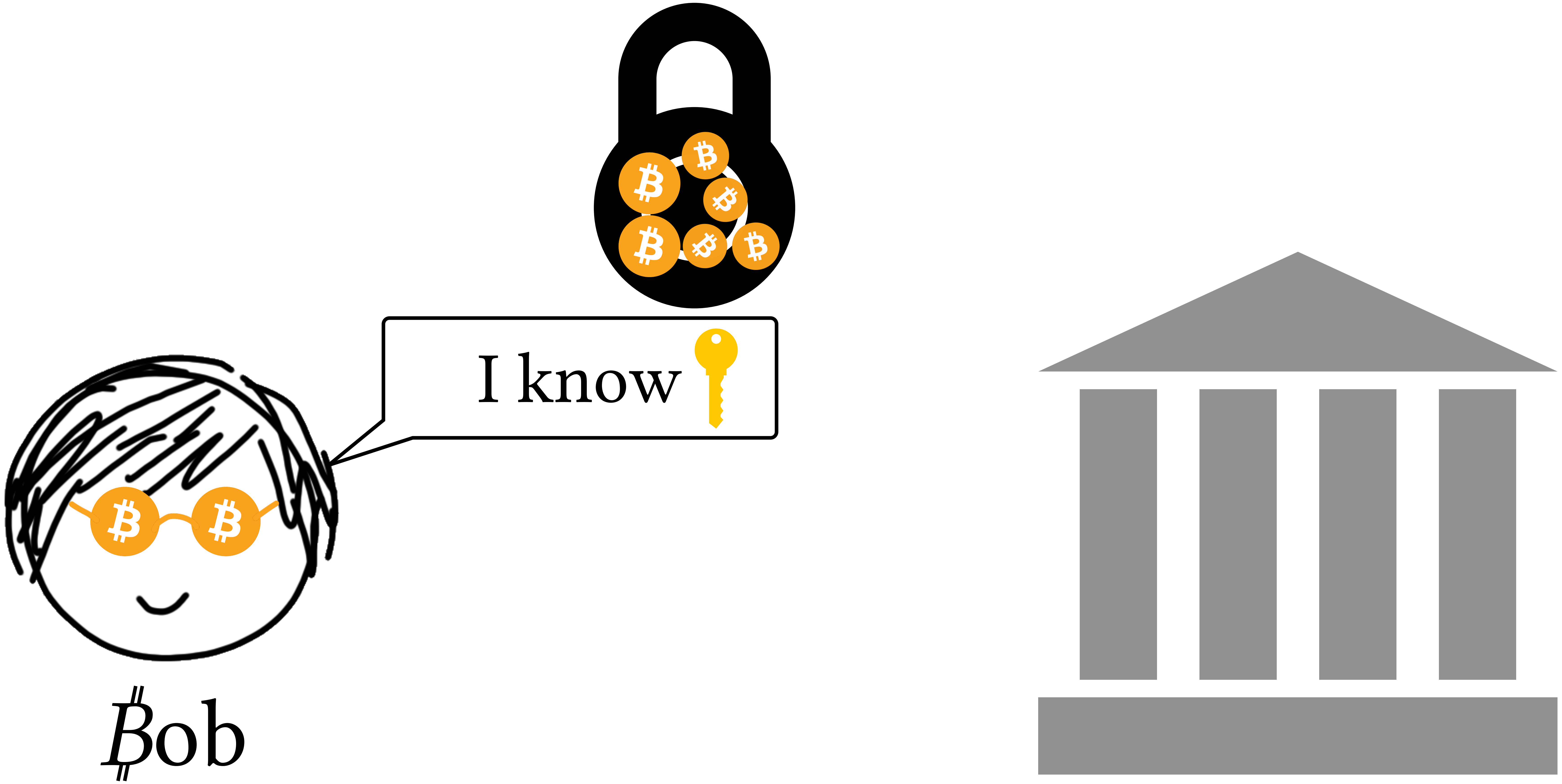
Zero-knowledge Proofs

- Simple application: proof of possession (key ownership)



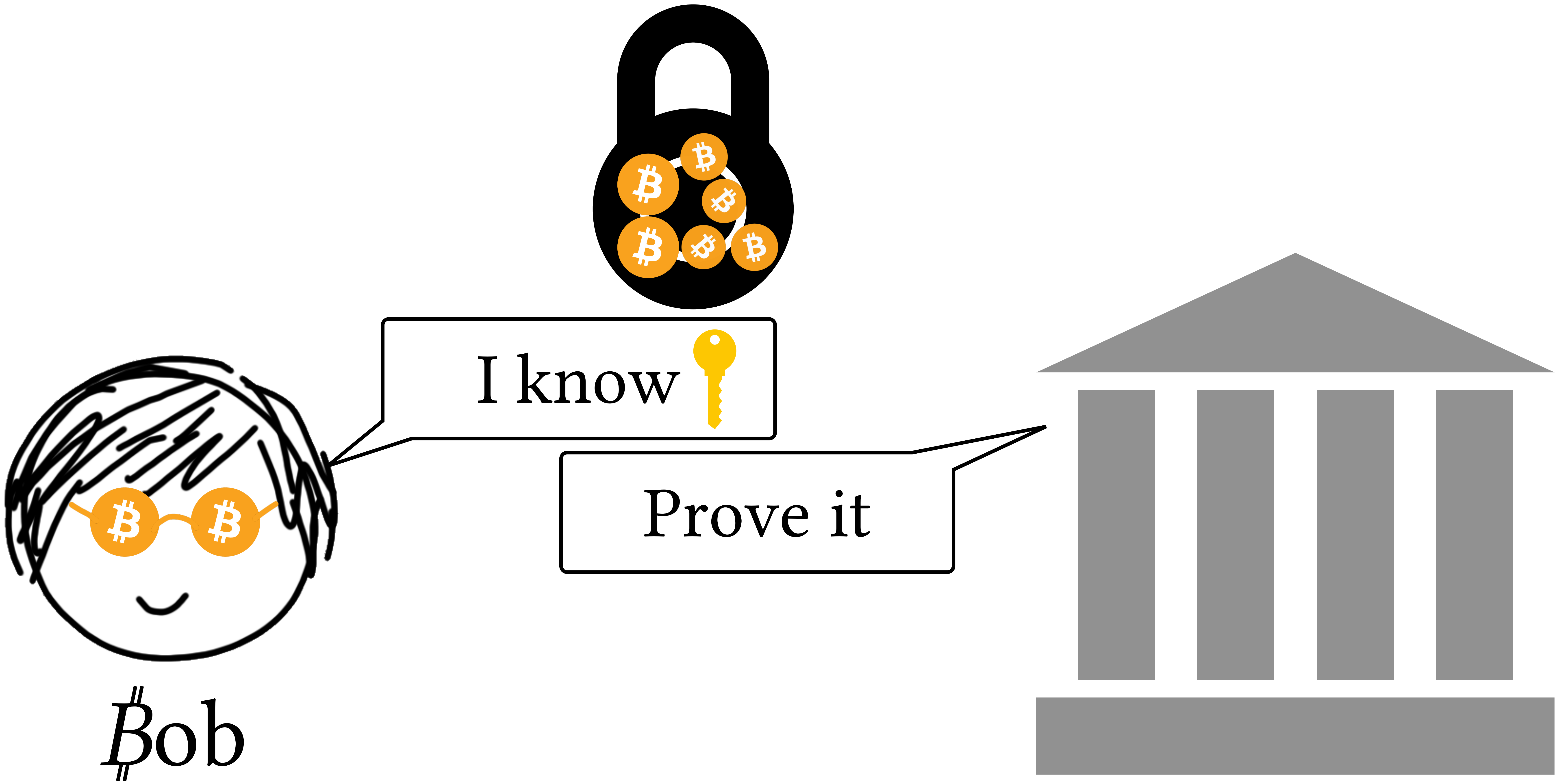
Zero-knowledge Proofs

- Simple application: proof of possession (key ownership)



Zero-knowledge Proofs

- Simple application: proof of possession (key ownership)



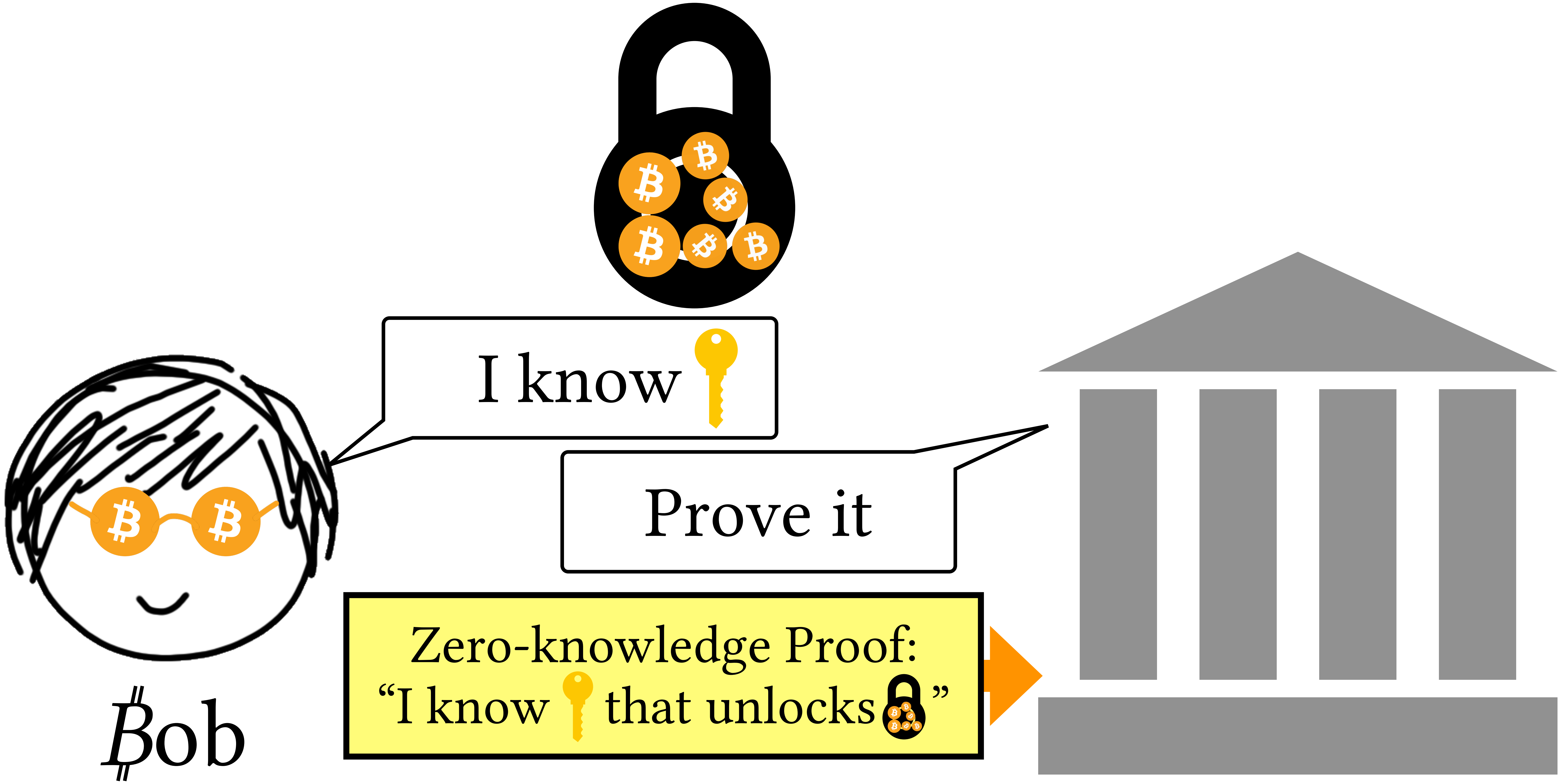
I know

Prove it

₿ob

Zero-knowledge Proofs

- Simple application: proof of possession (key ownership)



I know 🔑

Prove it

Zero-knowledge Proof:
"I know 🔑 that unlocks 🔒"

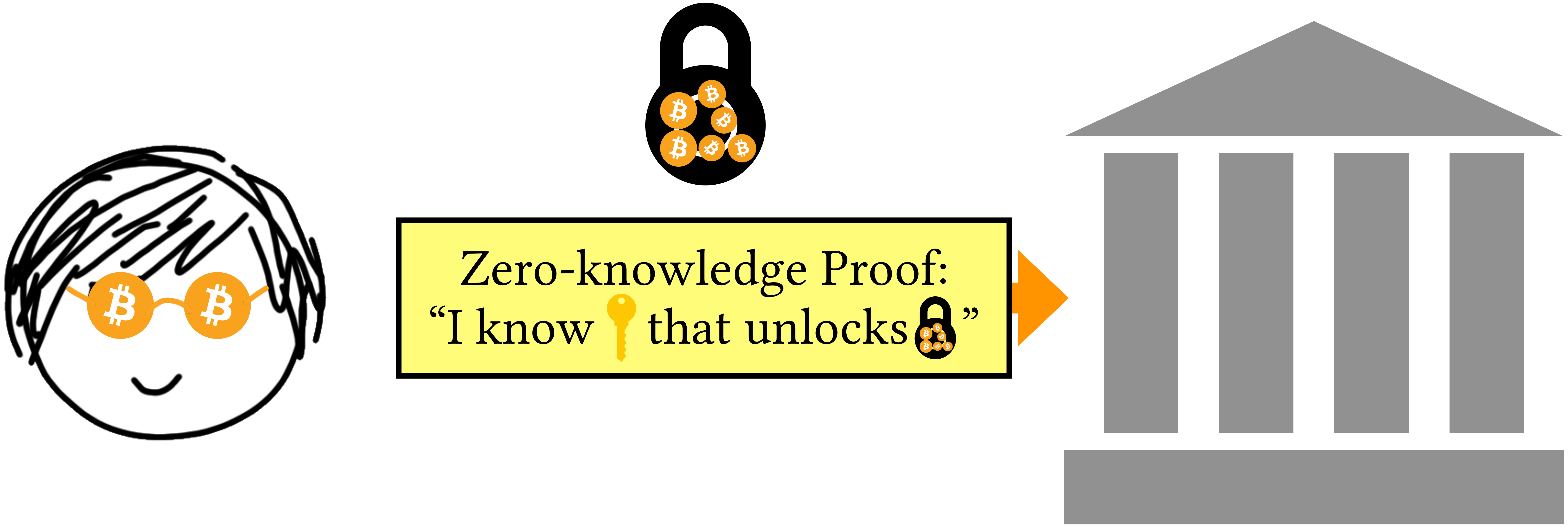
₿ob

Defining Zero-knowledge Proofs

- ZK is intuitive: No information about the key should be leaked by the proof
- But what does it mean to "know" something?
- "Proof of Knowledge" is formalized by an "extractor" Ext

Defining Zero-knowledge Proofs

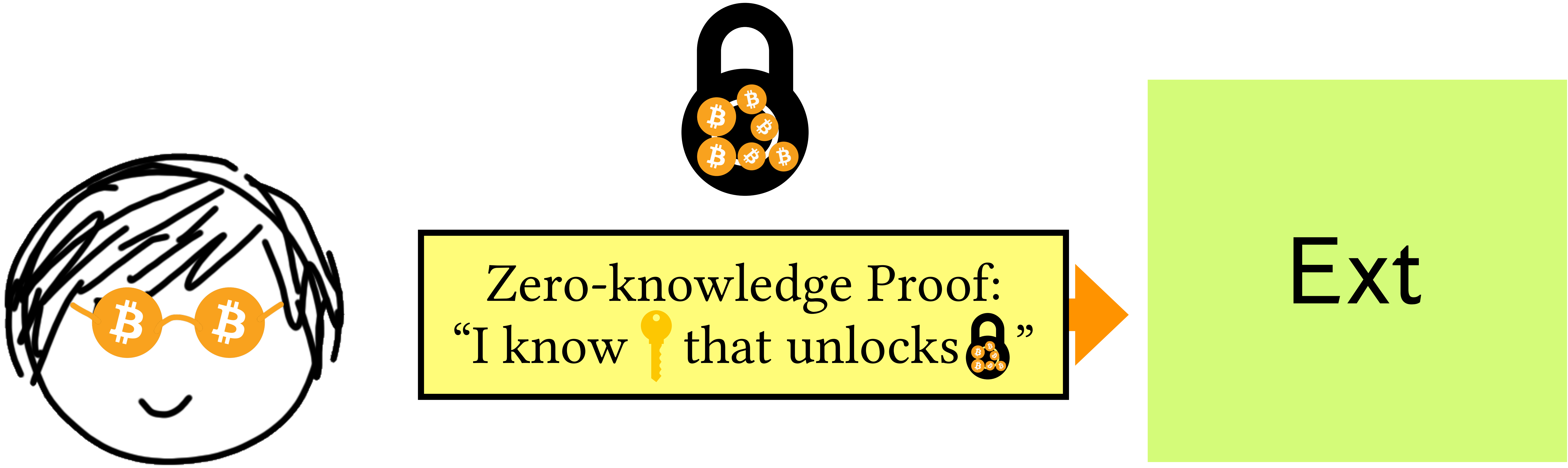
- ZK is intuitive: No information about the key should be leaked by the proof
- But what does it mean to “know” something?
- “Proof of Knowledge” is formalized by an “extractor” Ext



Zero-knowledge Proof:
“I know 🔑 that unlocks 🔒”

Defining Zero-knowledge Proofs

- ZK is intuitive: No information about the key should be leaked by the proof
- But what does it mean to “know” something?
- “Proof of Knowledge” is formalized by an “extractor” Ext

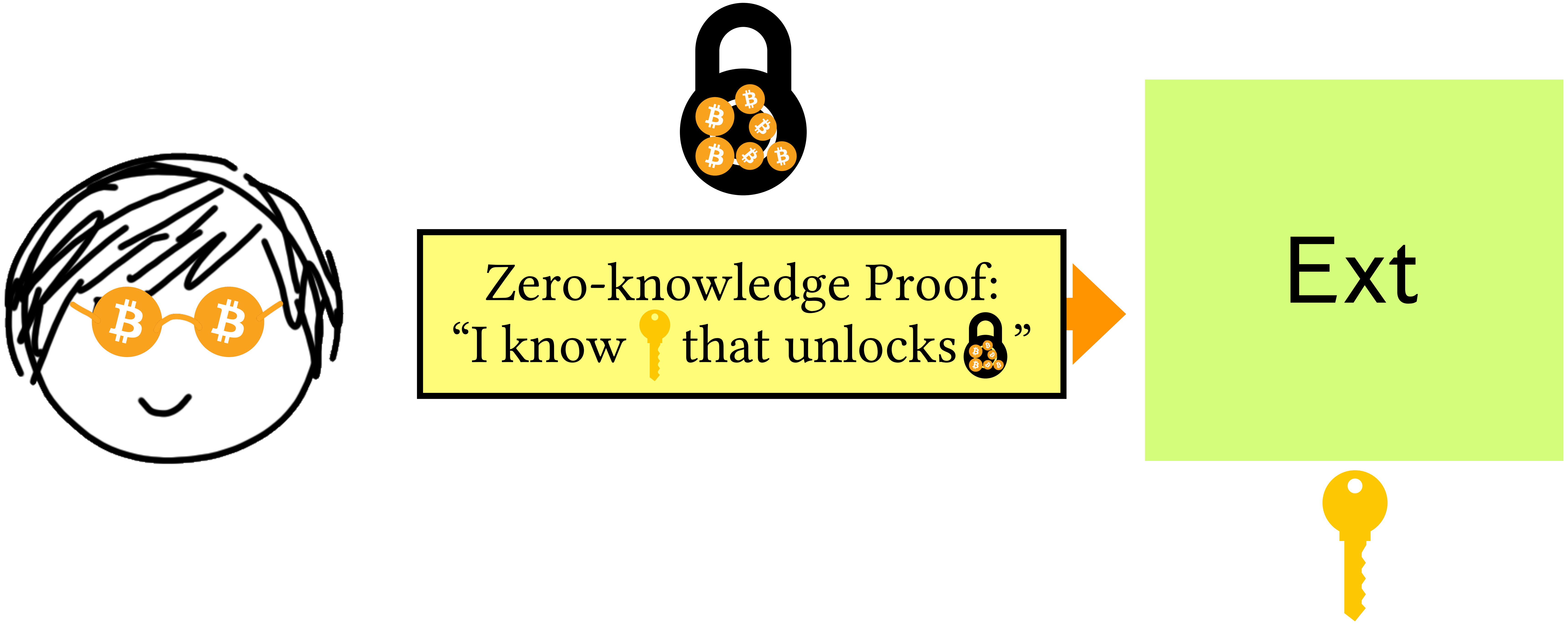


Zero-knowledge Proof:
“I know 🔑 that unlocks 🔒”

Ext

Defining Zero-knowledge Proofs

- ZK is intuitive: No information about the key should be leaked by the proof
- But what does it mean to "know" something?
- "Proof of Knowledge" is formalized by an "extractor" Ext



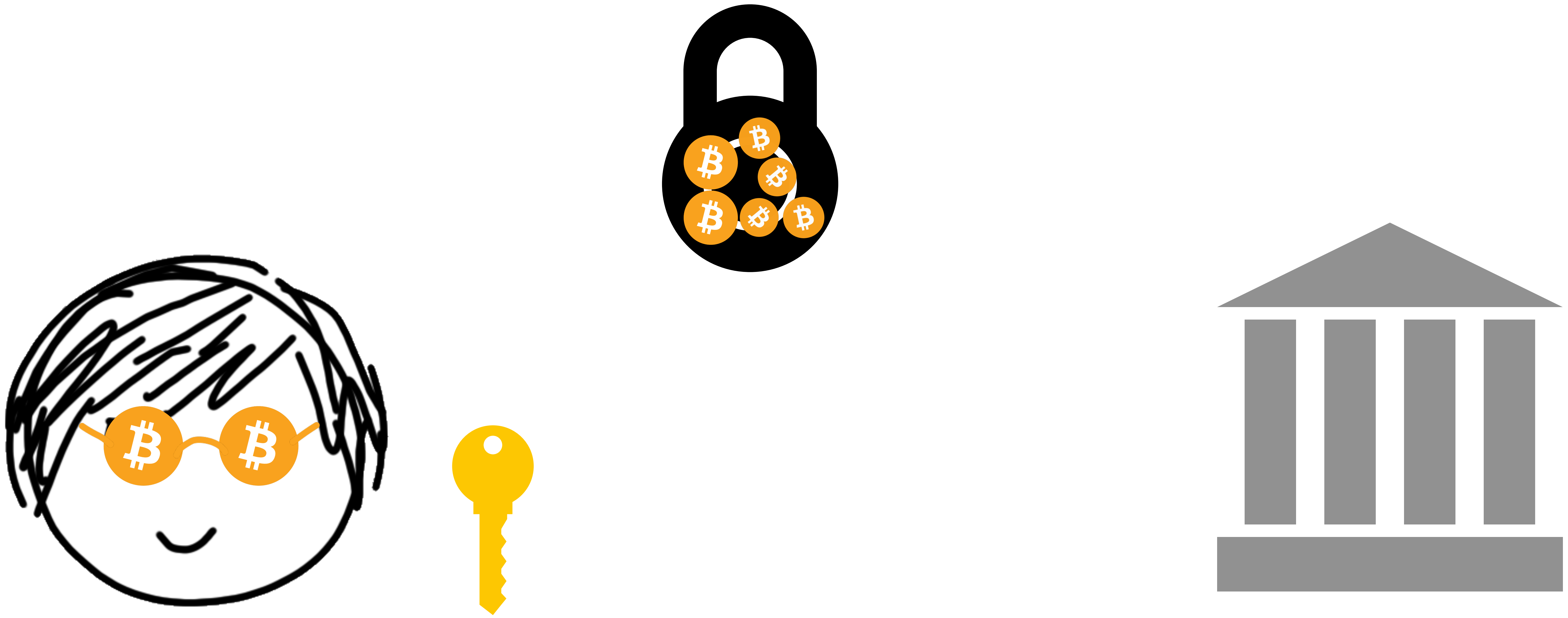
Zero-knowledge Proof:
"I know 🔑 that unlocks 🔒"

Why is Ext special?

- Clearly, Ext must not be an algorithm that just anybody can run
- Ext has carefully chosen special privileges:
  - Powerful enough to accomplish extraction
  - Still meaningful as a security claim
- We will look at a certain type of ZK proof to build intuition

Σ Protocols

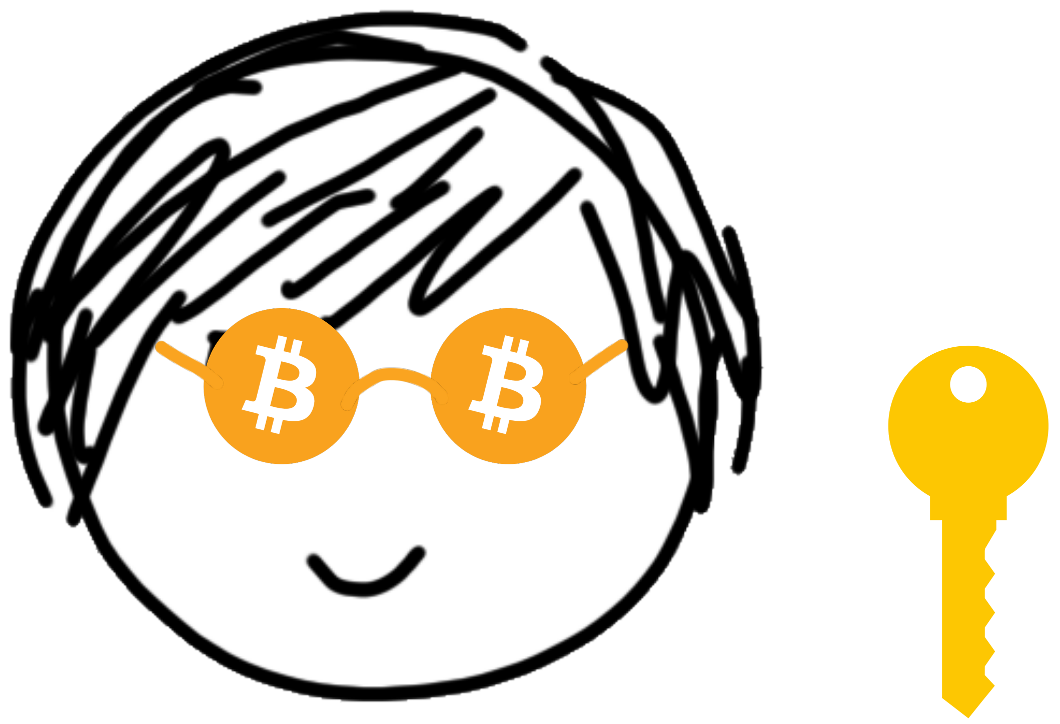
[Damgård 02]



Σ Protocols

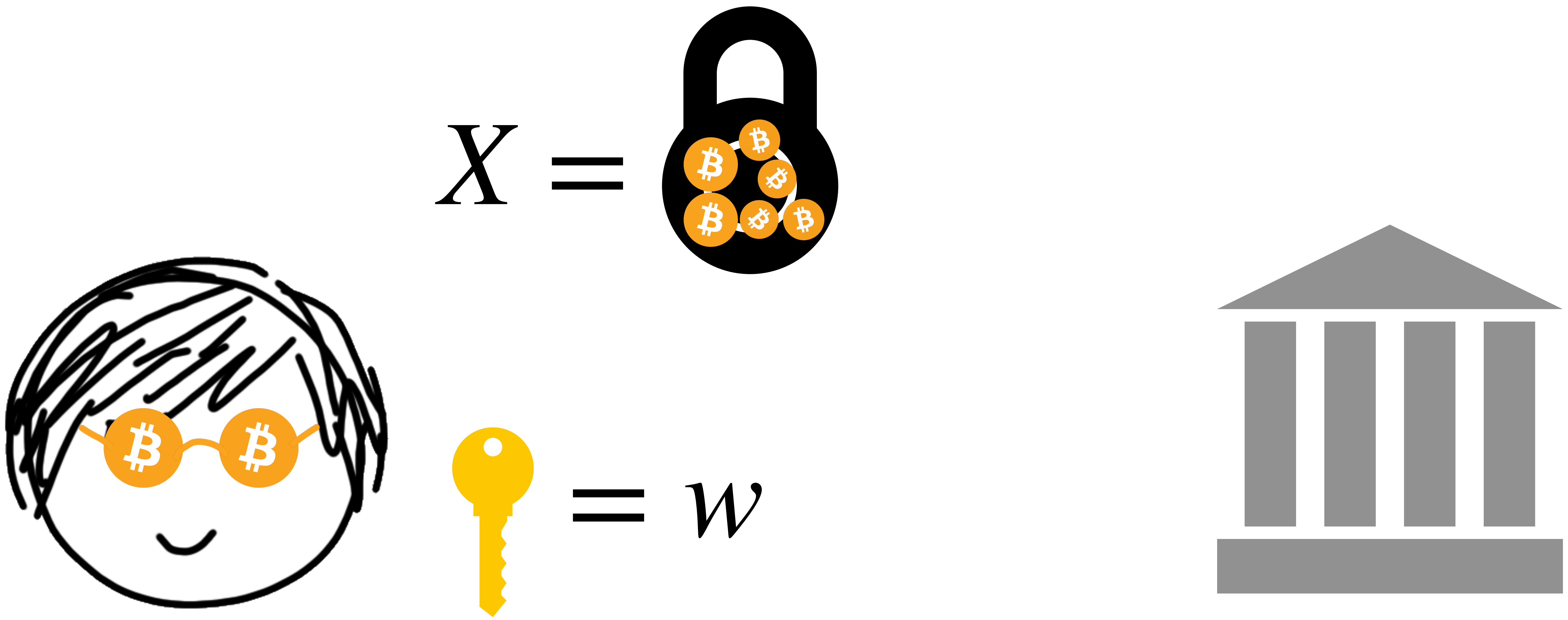
[Damgård 02]

$X =$



Σ Protocols

[Damgård 02]

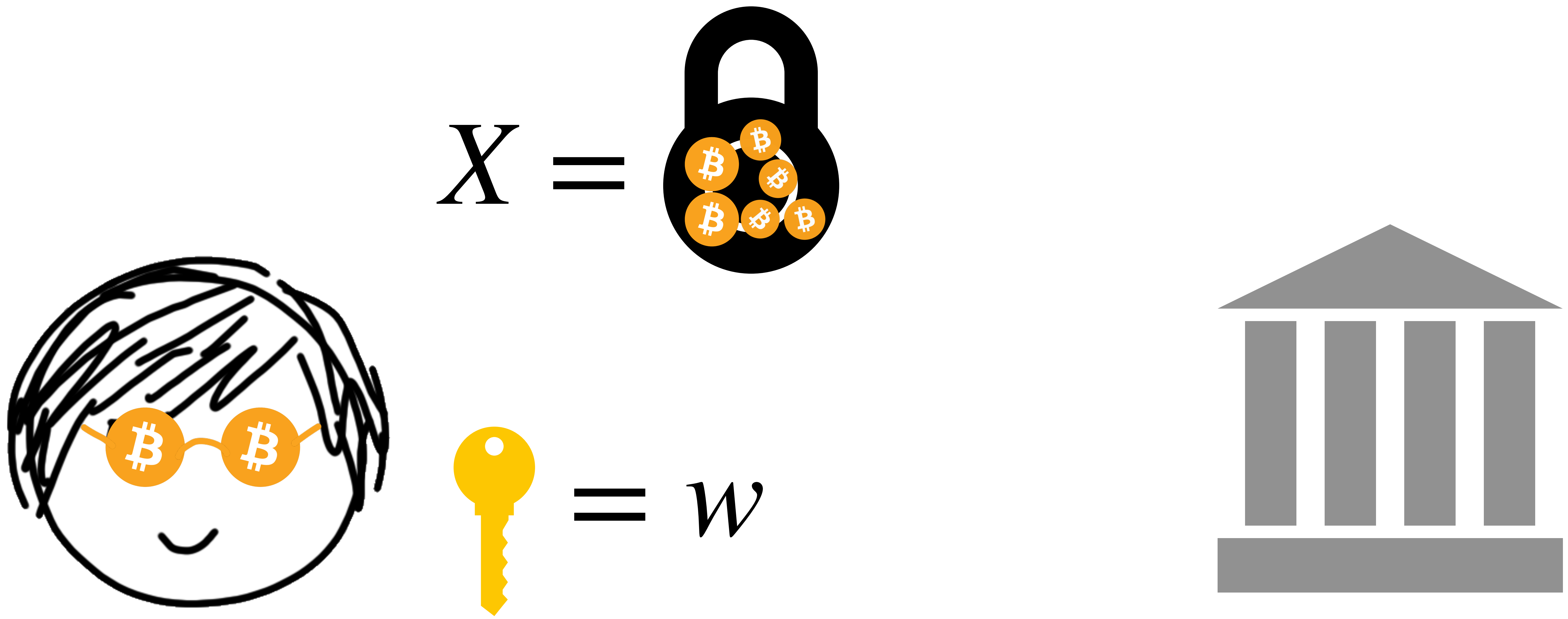


$X =$ 🔒

🔑 $= w$

Σ Protocols

[Damgård 02]



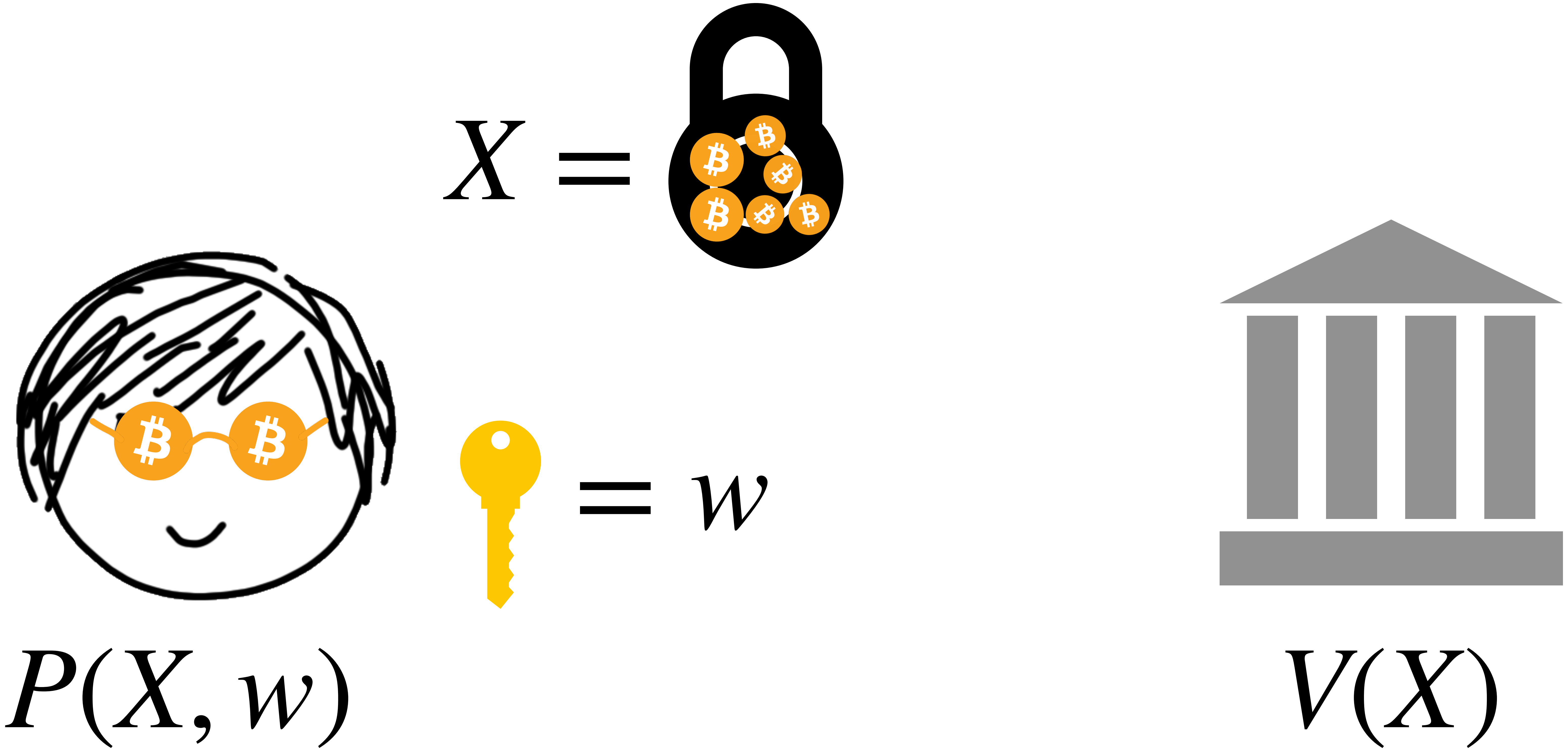
$X =$

$= w$

$P(X, w)$

Σ Protocols

[Damgård 02]



$X =$

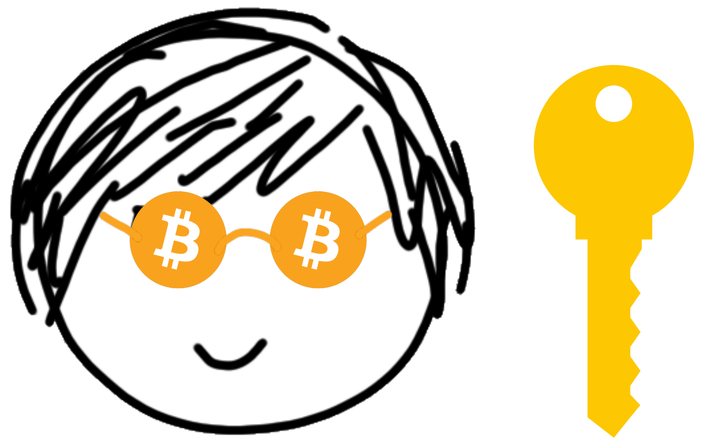
$= w$

$P(X, w)$

$V(X)$

Σ Protocols

[Damgård 02]



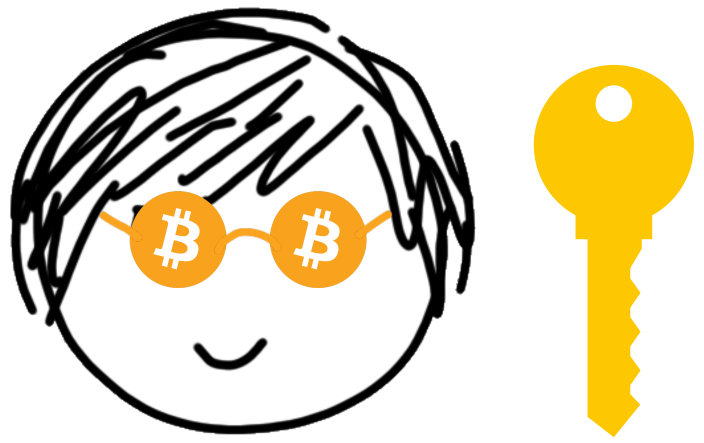
$X =$

$P(X, w)$

$V(X)$

Σ Protocols

[Damgård 02]



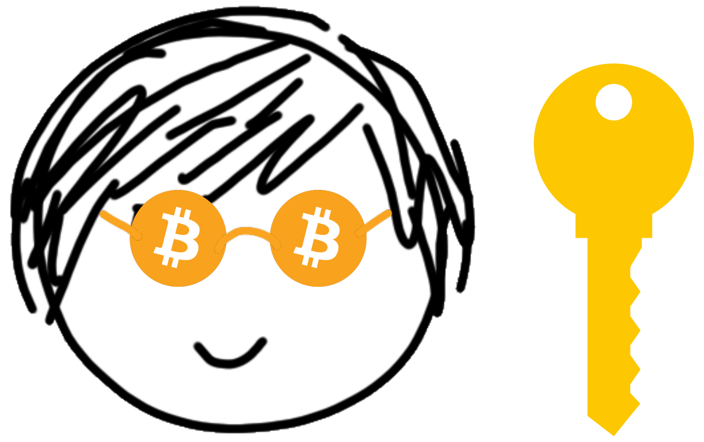
$P(X, w)$

$X =$

$V(X)$

Σ Protocols

[Damgård 02]



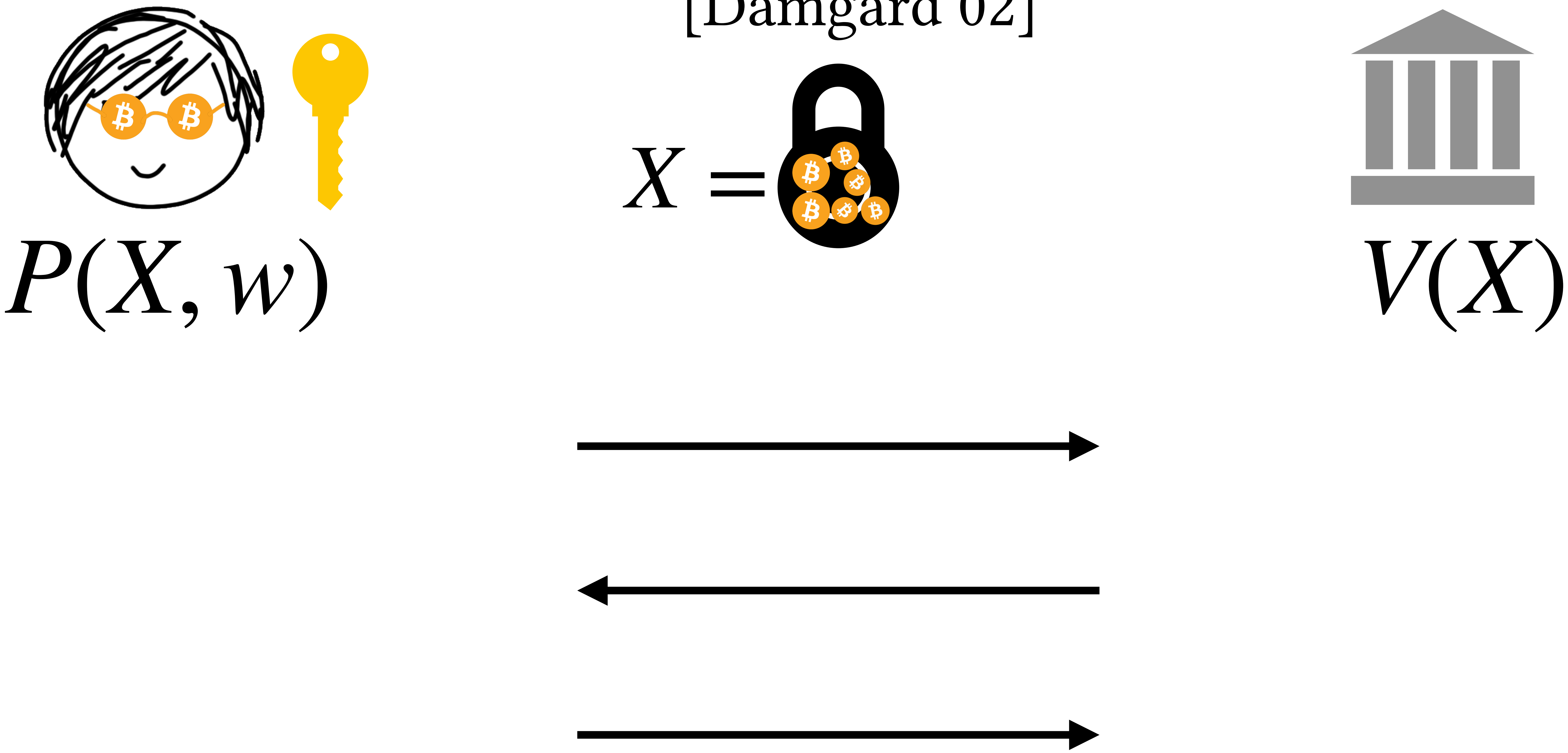
$X =$

$P(X, w)$

$V(X)$

Σ Protocols

[Damgård 02]



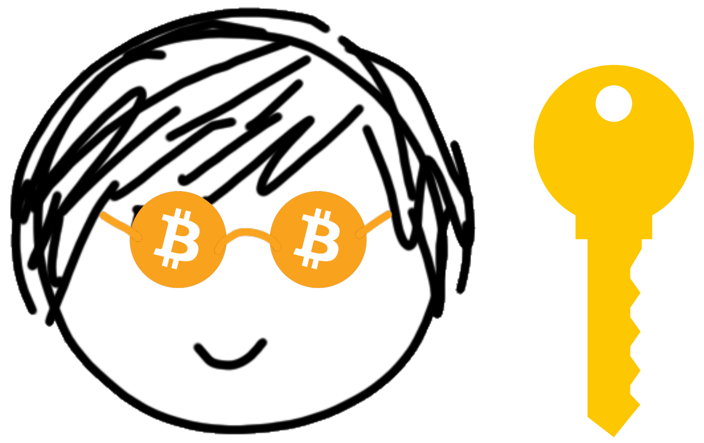
$P(X, w)$

$X =$

$V(X)$

Σ Protocols

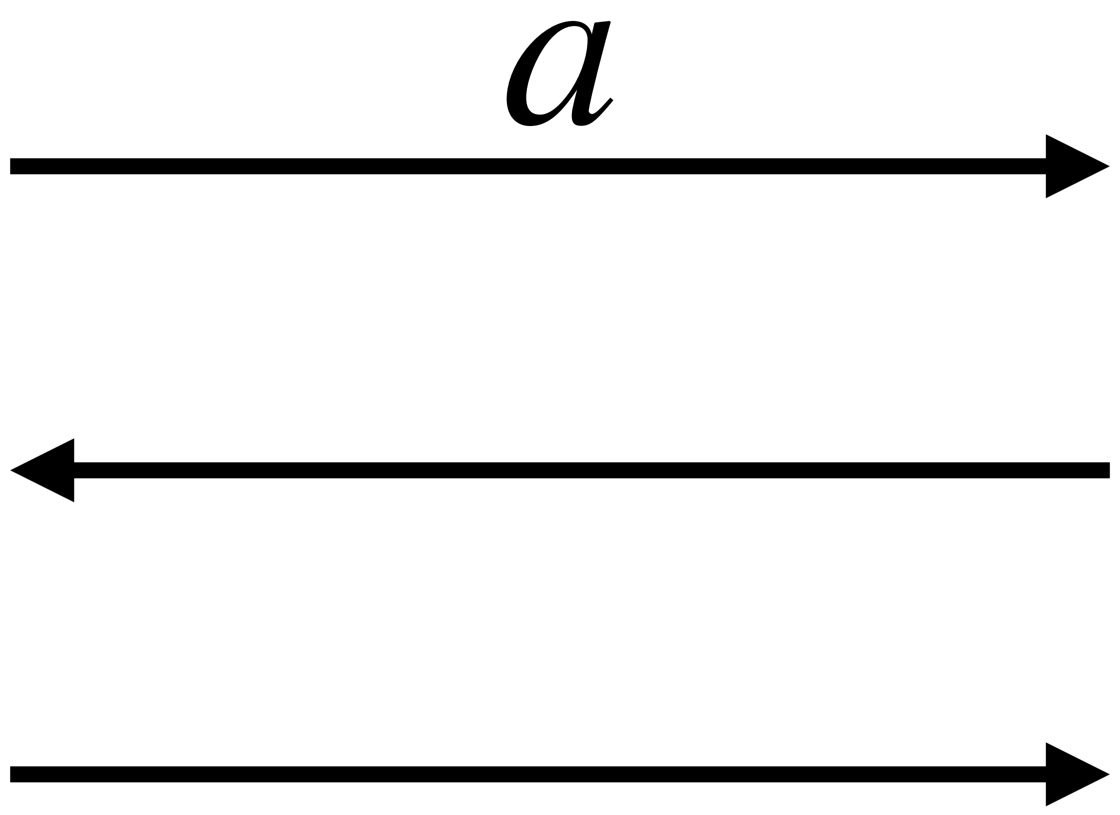
[Damgård 02]



$X =$

$P(X, w)$

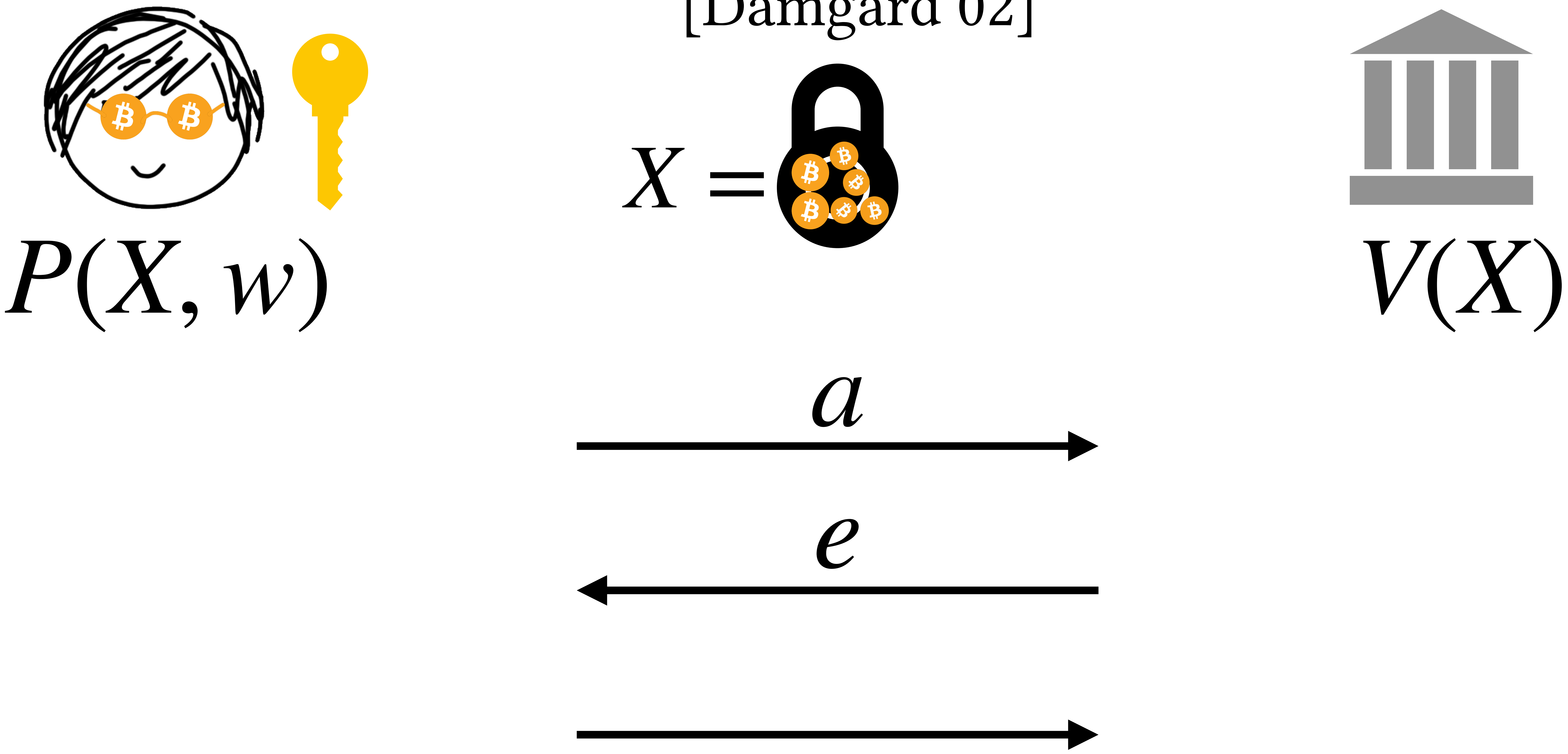
$V(X)$



a

Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$

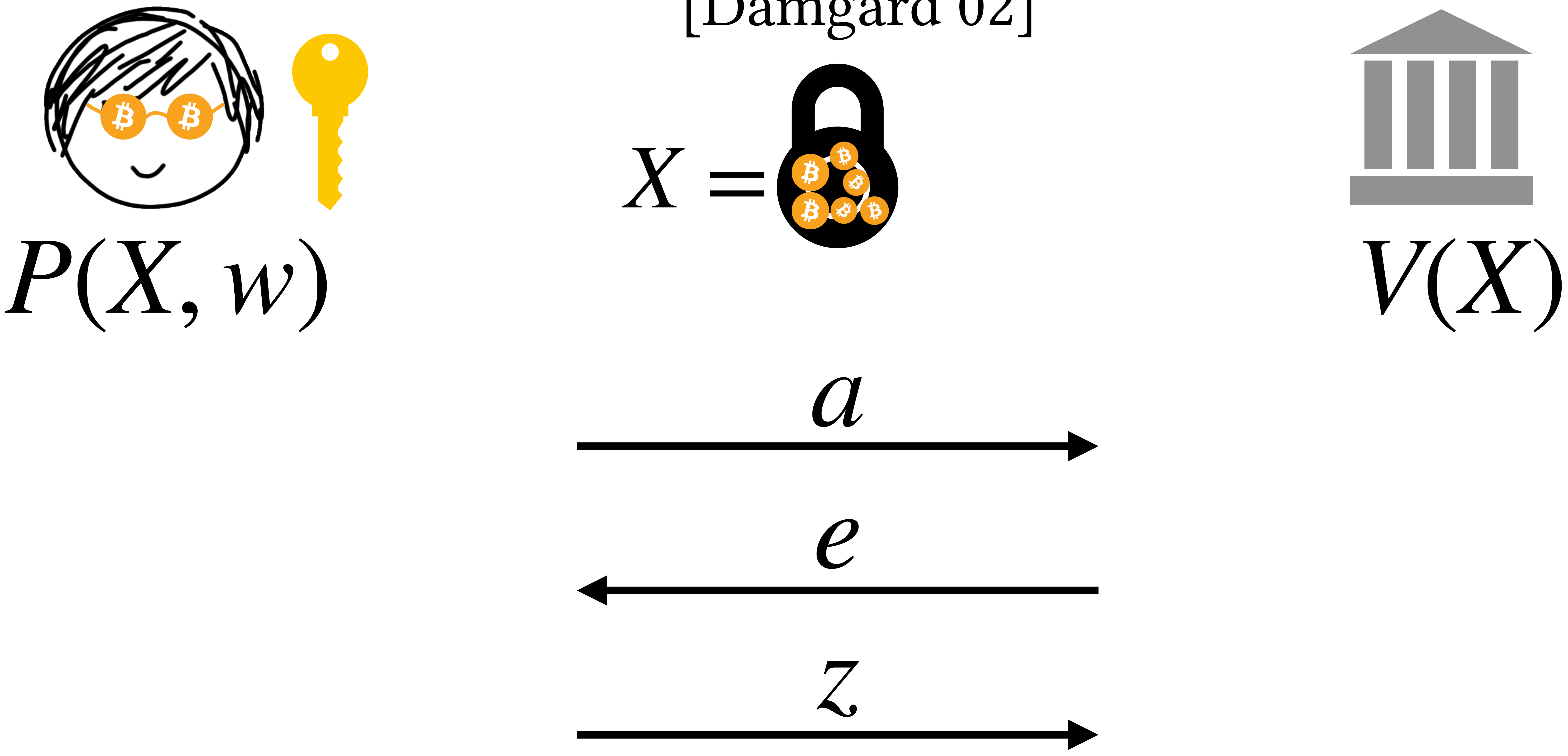
$V(X)$

a

e

Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$

$V(X)$

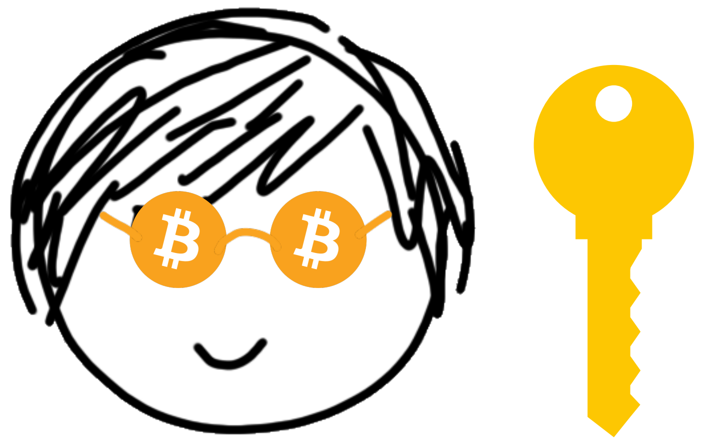
$$P \xrightarrow{a} V$$

$$P \xleftarrow{e} V$$

$$P \xrightarrow{z} V$$

Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$

$V(X)$

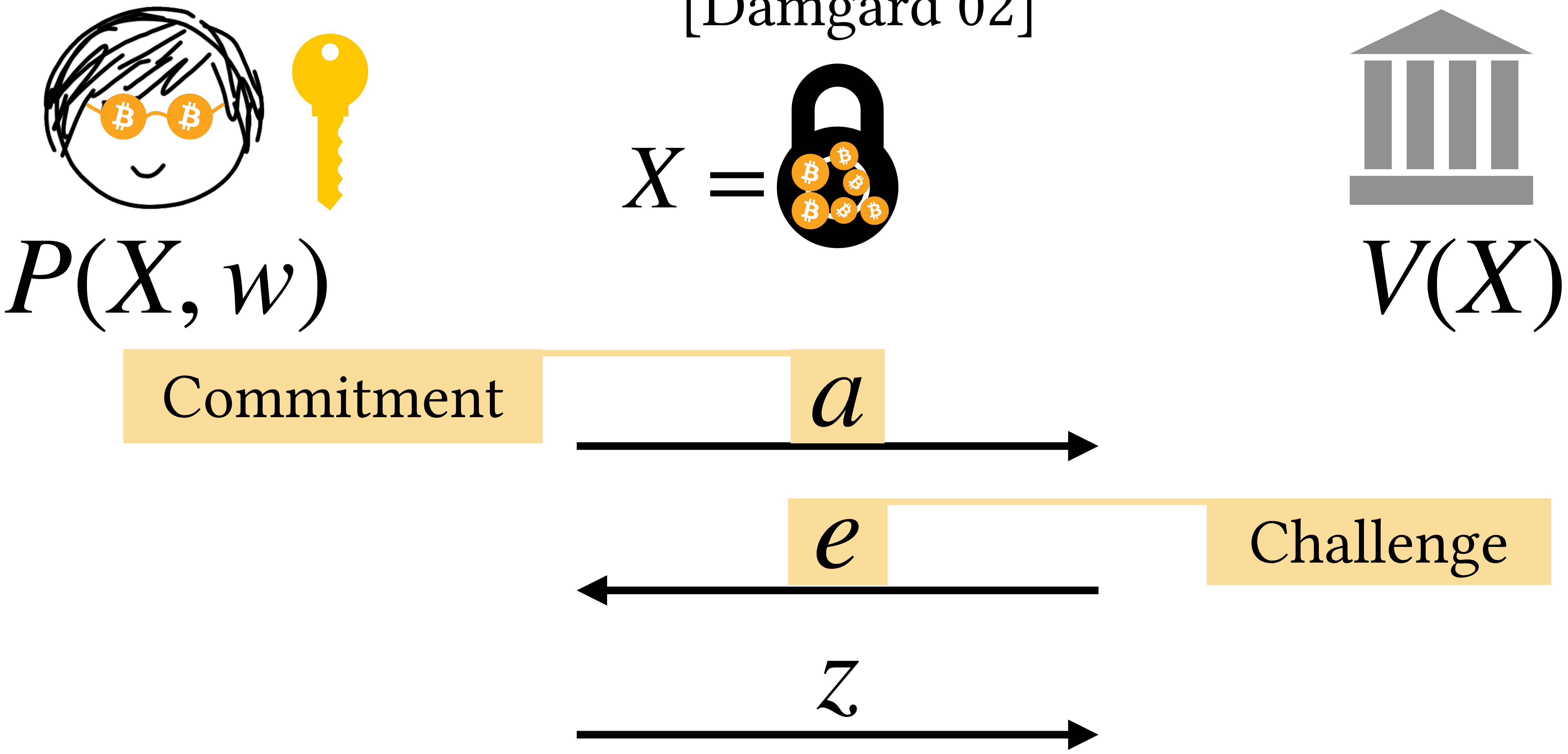
Commitment $\quad a \quad \longrightarrow$

$\longleftarrow \quad e$

$z \quad \longrightarrow$

Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$

$V(X)$

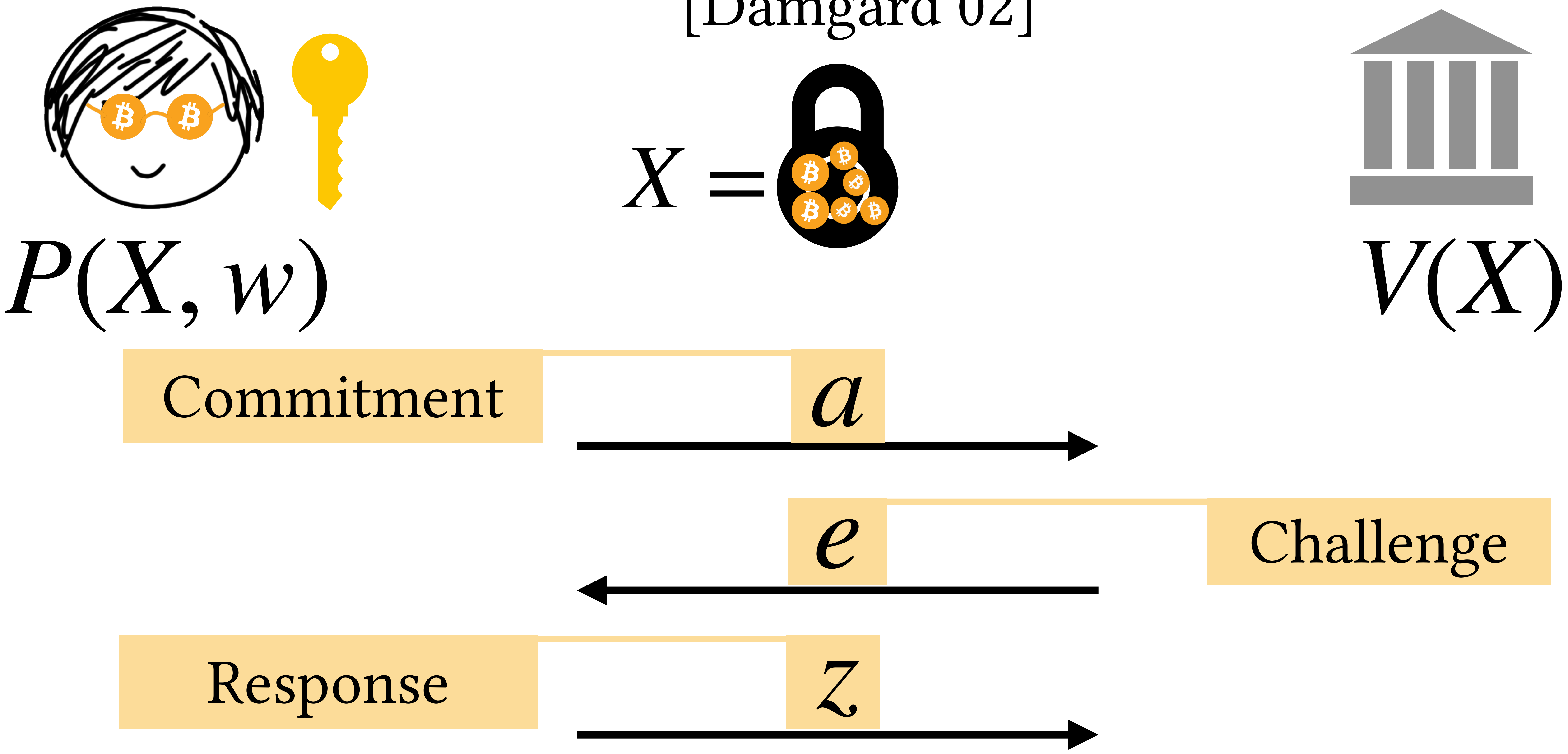
Commitment a

e Challenge

z

Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$

$V(X)$

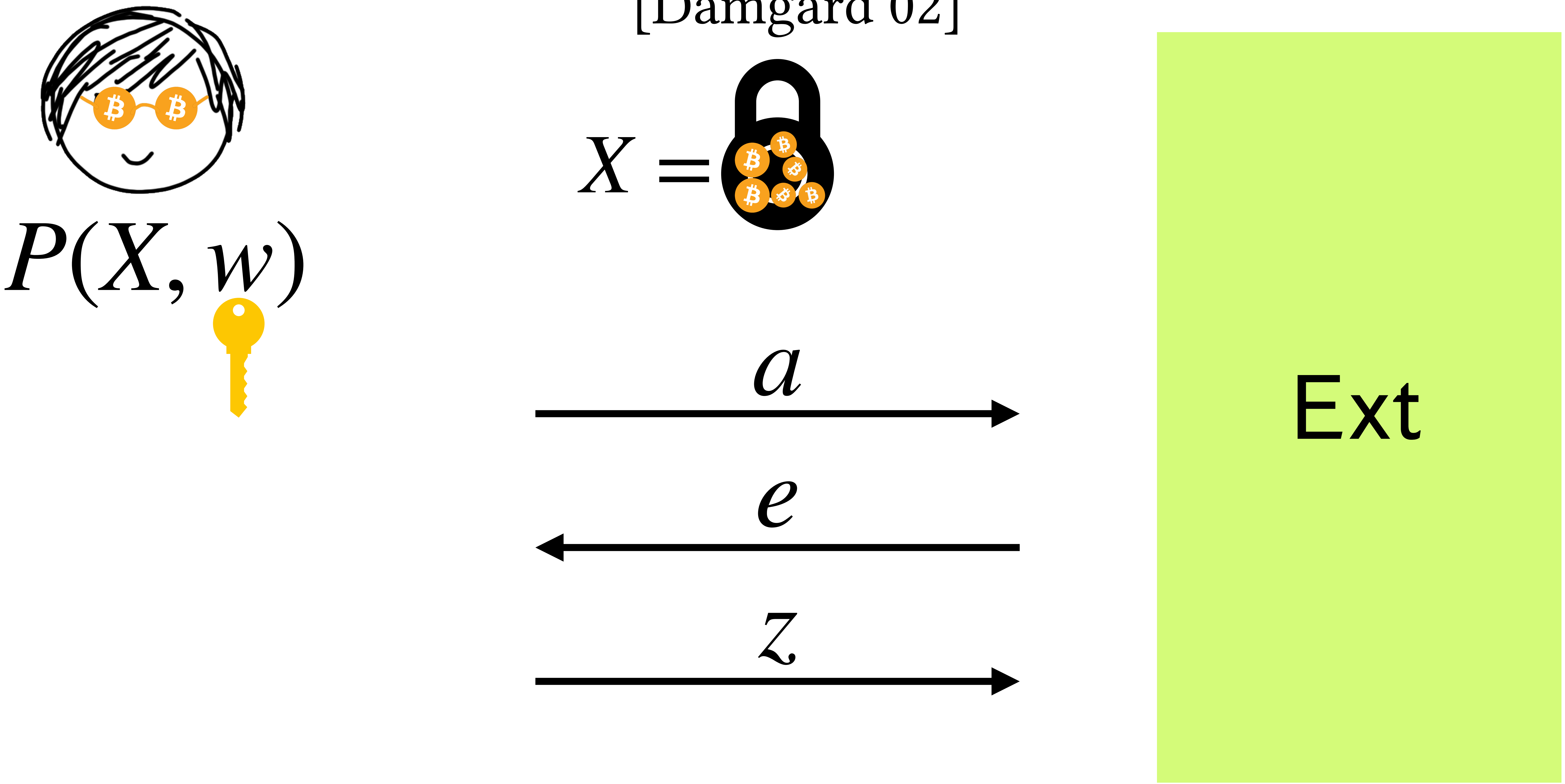
Commitment a

e Challenge

Response z

Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$

a

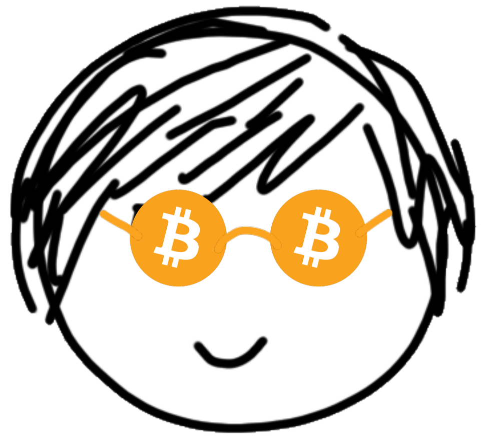
e

z

Ext

Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$

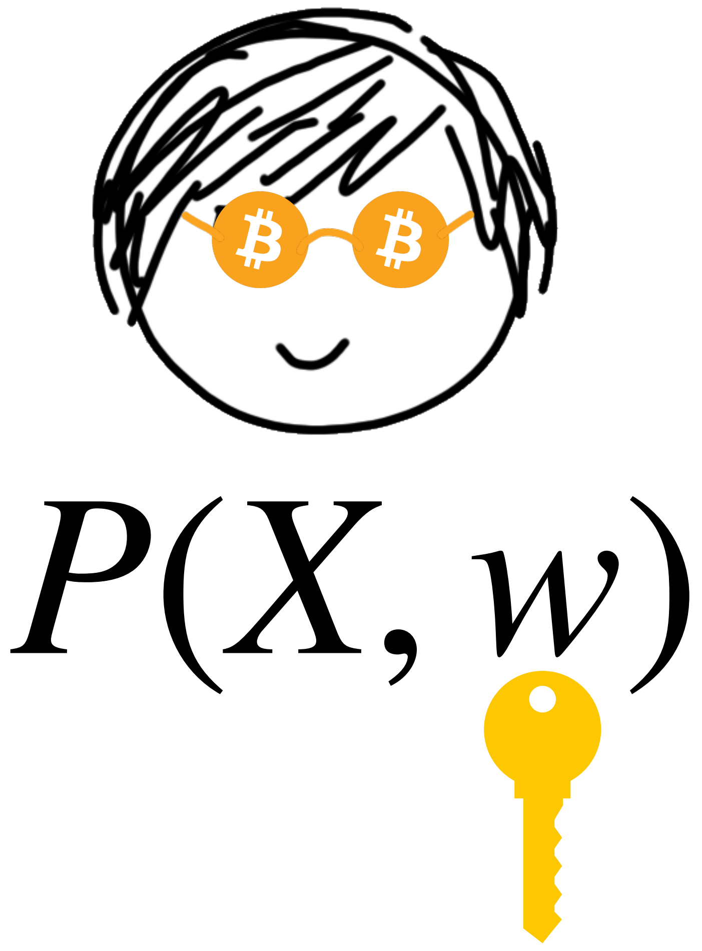
$$\xrightarrow{a}$$

Ext

$e \quad z$

Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$

$$\xrightarrow{\quad a \quad}$$

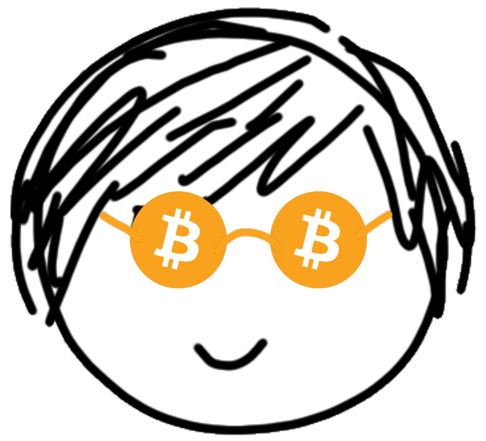
$$\xleftarrow{\quad e' \quad}$$

Ext

$e \quad z$

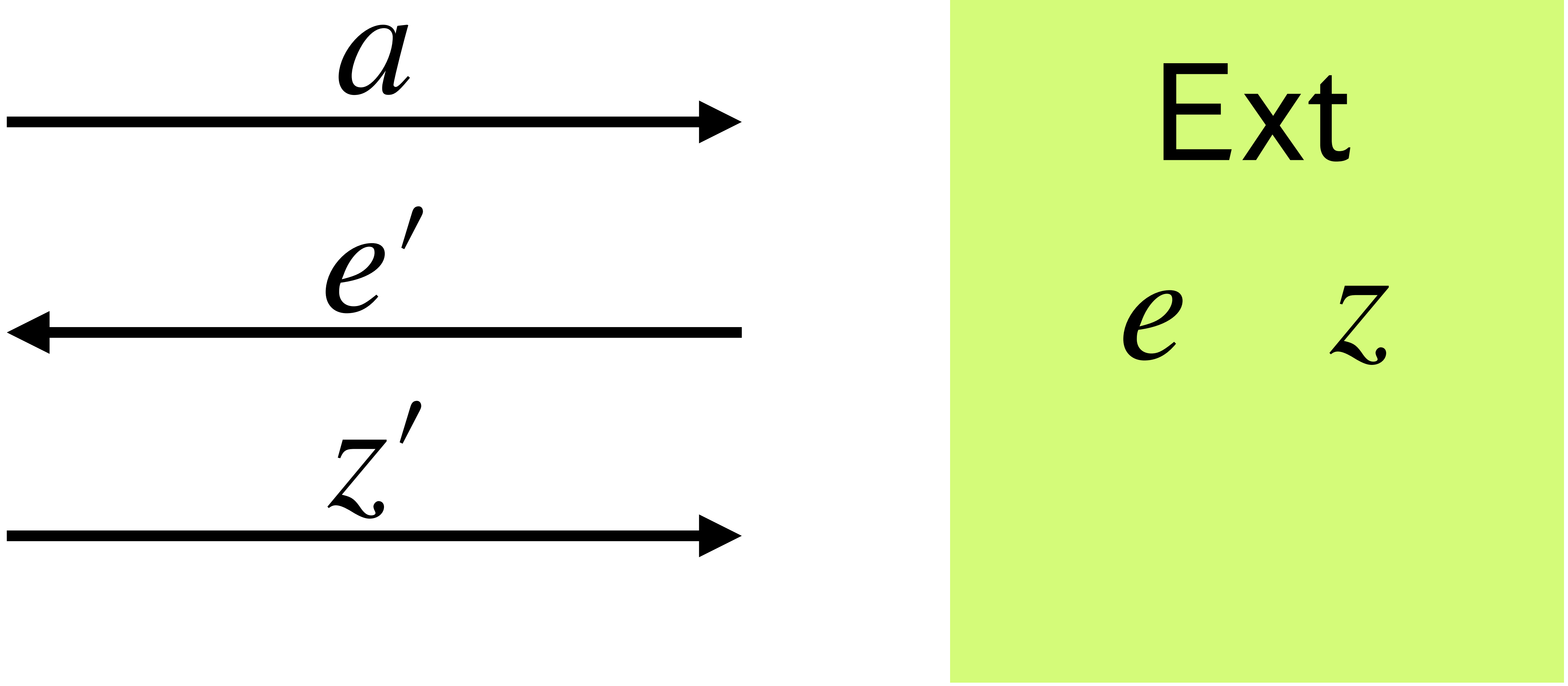
Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$



a

e'

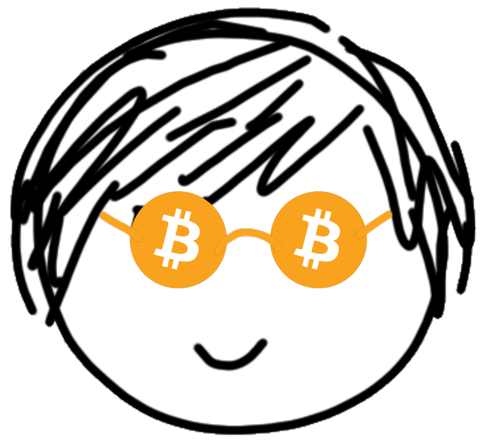
z'

Ext

e z

Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$

$\xrightarrow{a}$

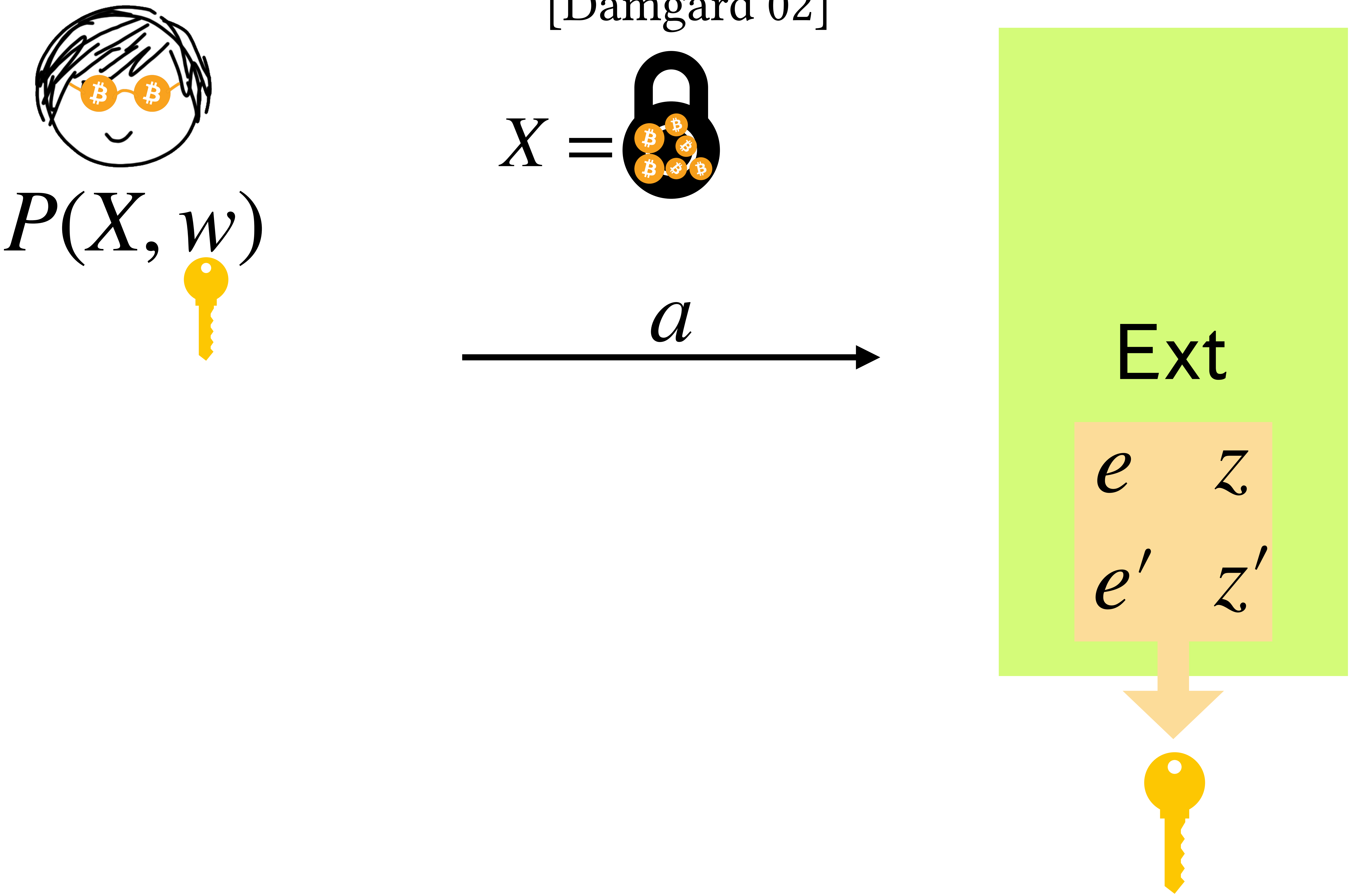
Ext

$e \quad z$

$e' \quad z'$

Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$

a

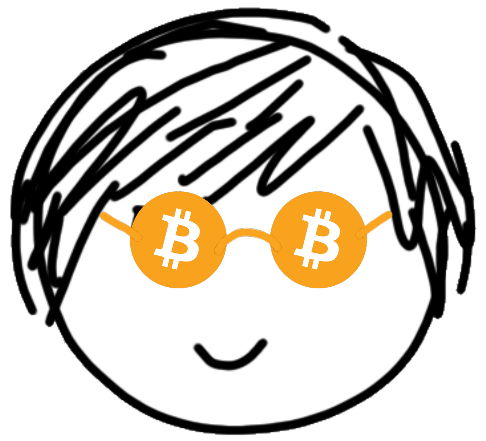
Ext

$e \quad z$

$e' \quad z'$

Σ Protocols

[Damgård 02]



$P(X, w)$

$X =$

$$\xrightarrow{\quad a \quad}$$

Ext

$e \quad z$

$e' \quad z'$

<u>Toy example</u>

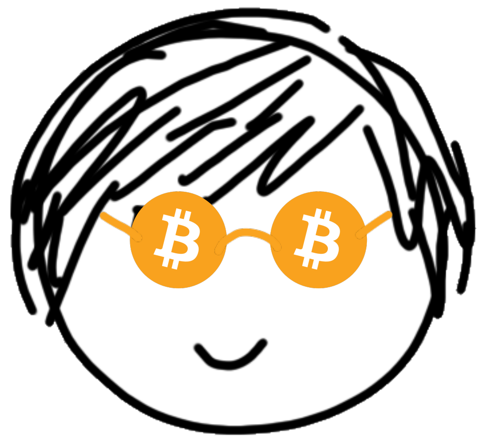
$z = we + f(a)$

$z' = we' + f(a)$

solve for w

Σ Protocols

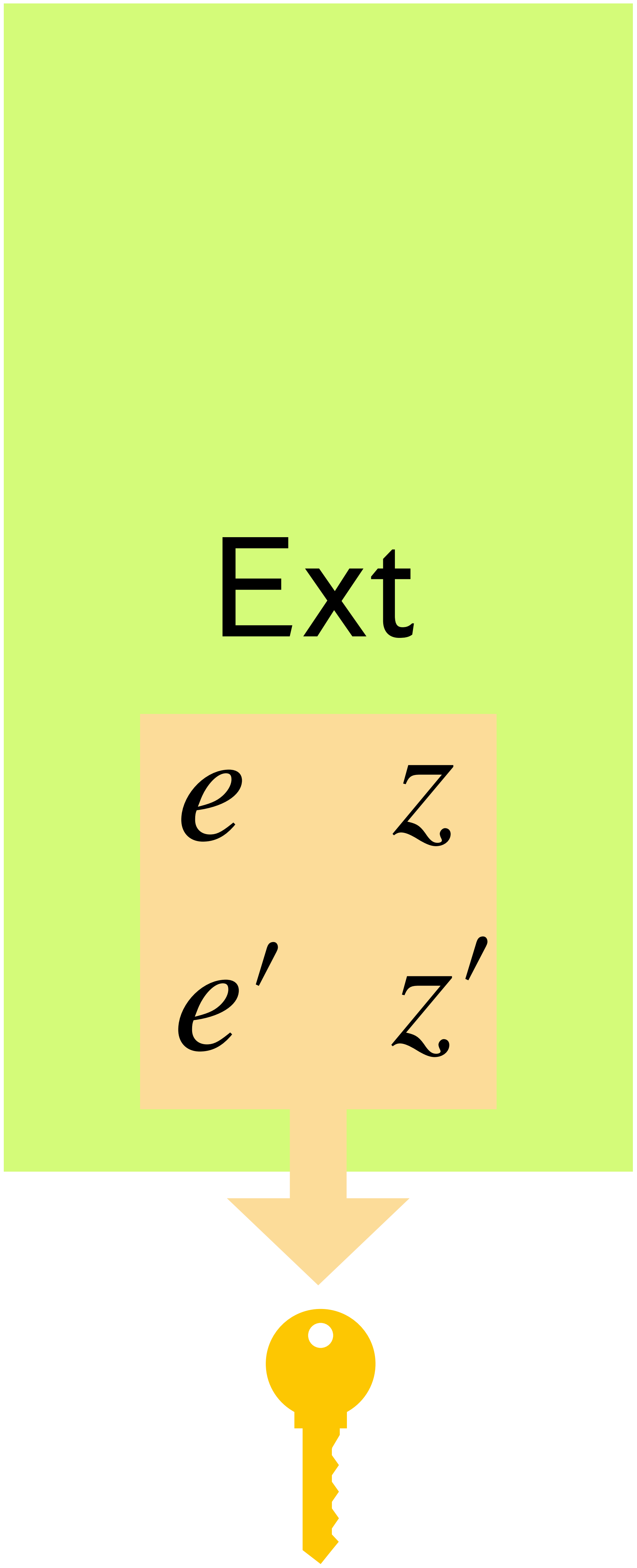
[Damgård 02]



$P(X, w)$

$X =$

$$\xrightarrow{\quad a \quad}$$



Ext

$e \quad z$

$e' \quad z'$

This is a useful protocol feature to keep in mind

<u>Toy example</u>

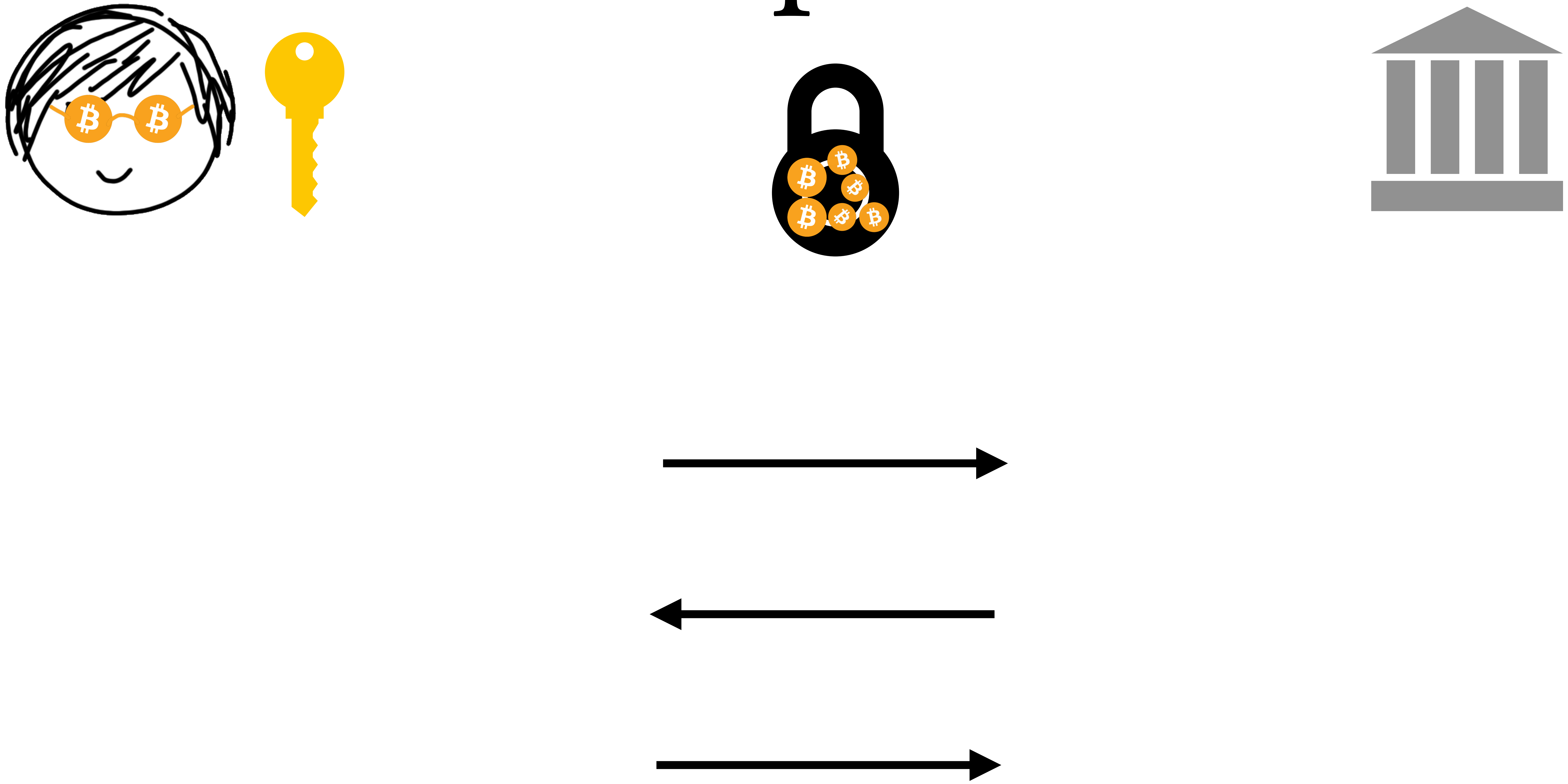
$z = we + f(a)$

$z' = we' + f(a)$

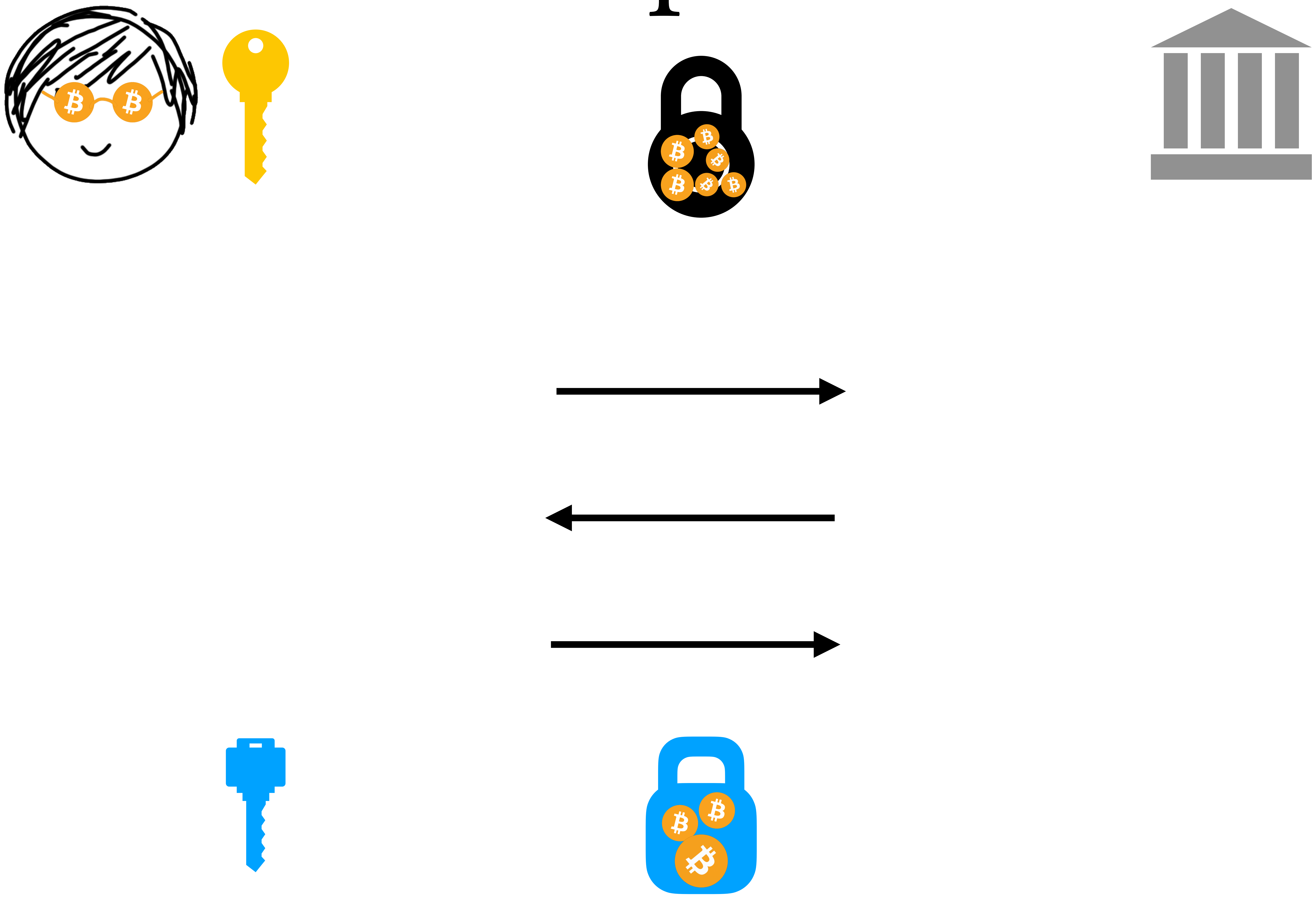
solve for w

Composable Non-interactive Zero-knowledge Proofs in the Random Oracle Model

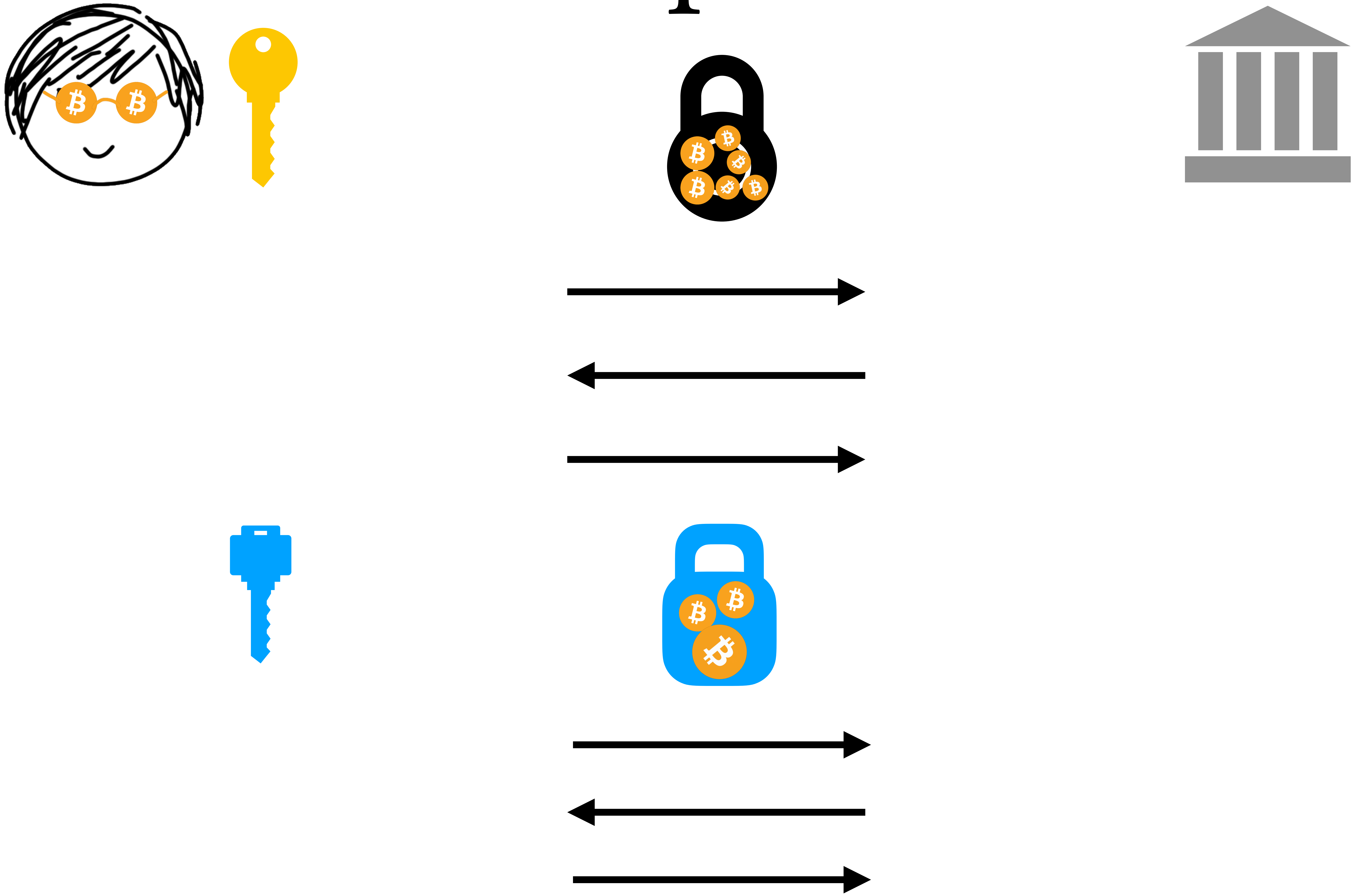
Composable?



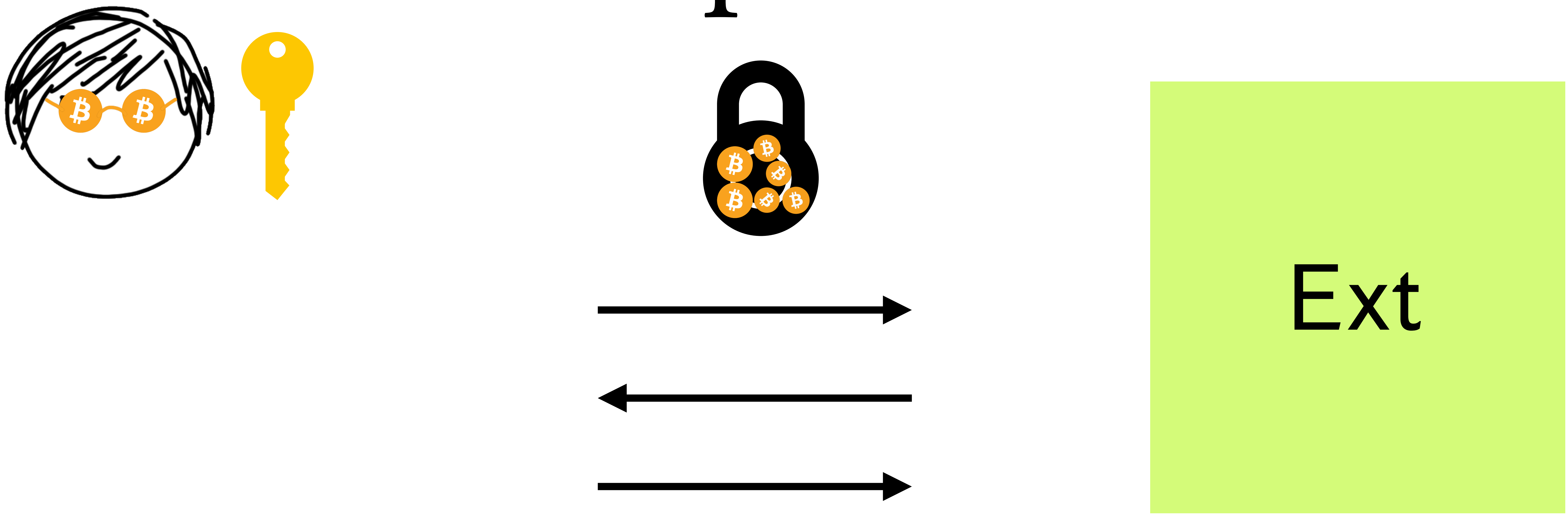
Composable?



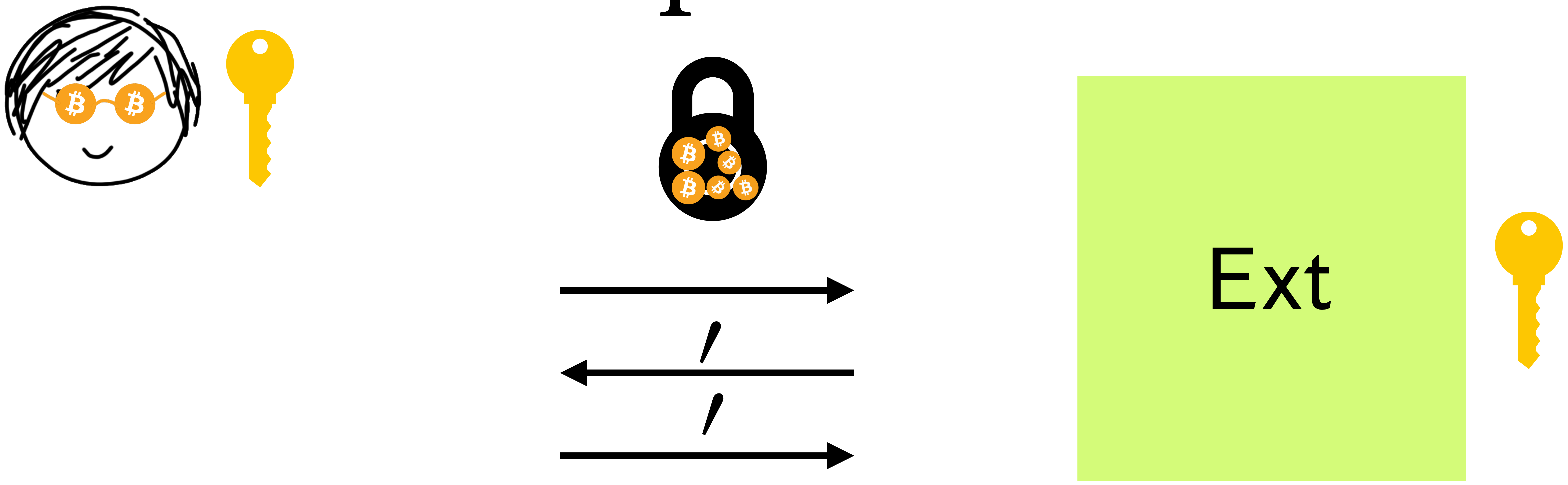
Composable?



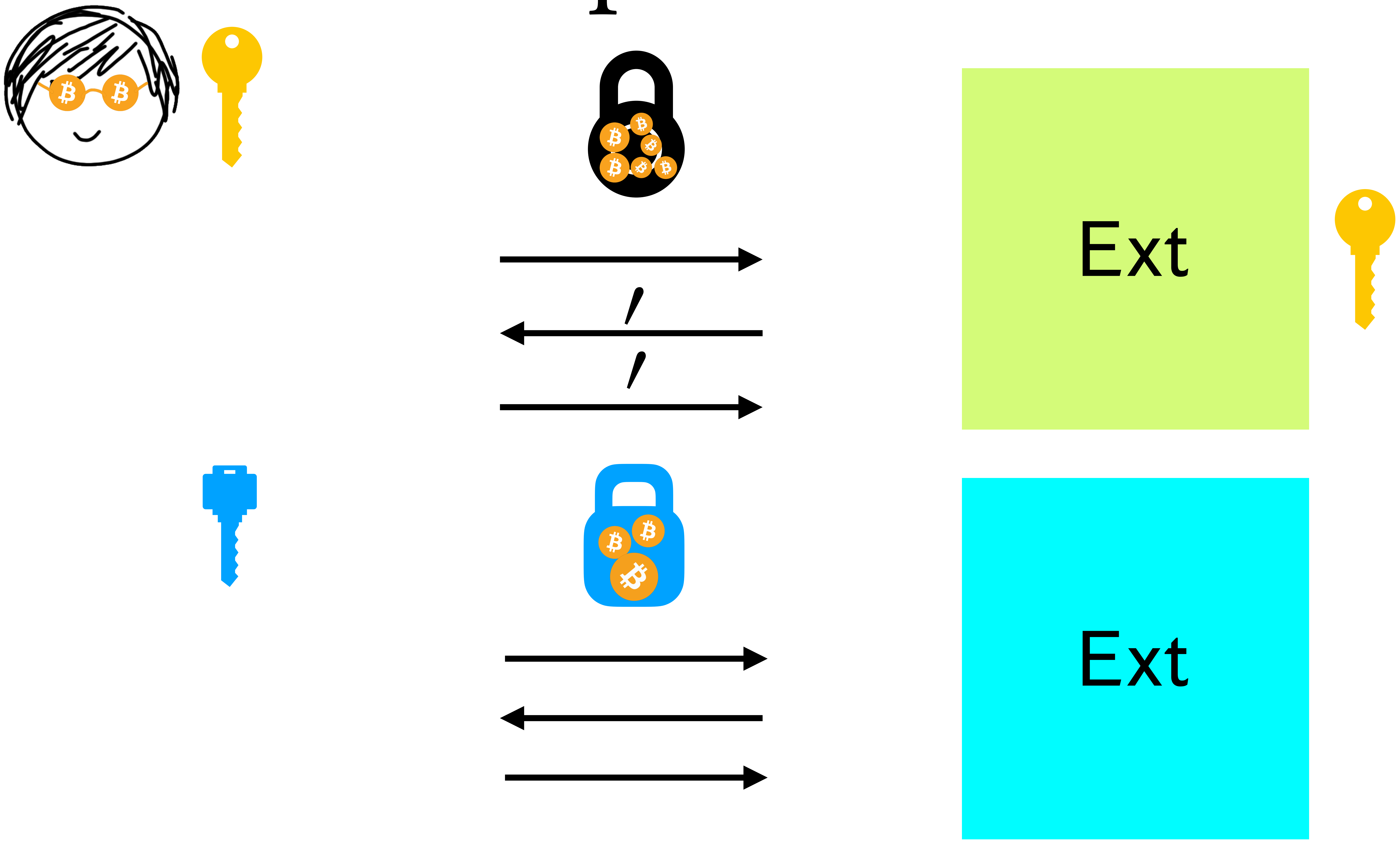
Composable?



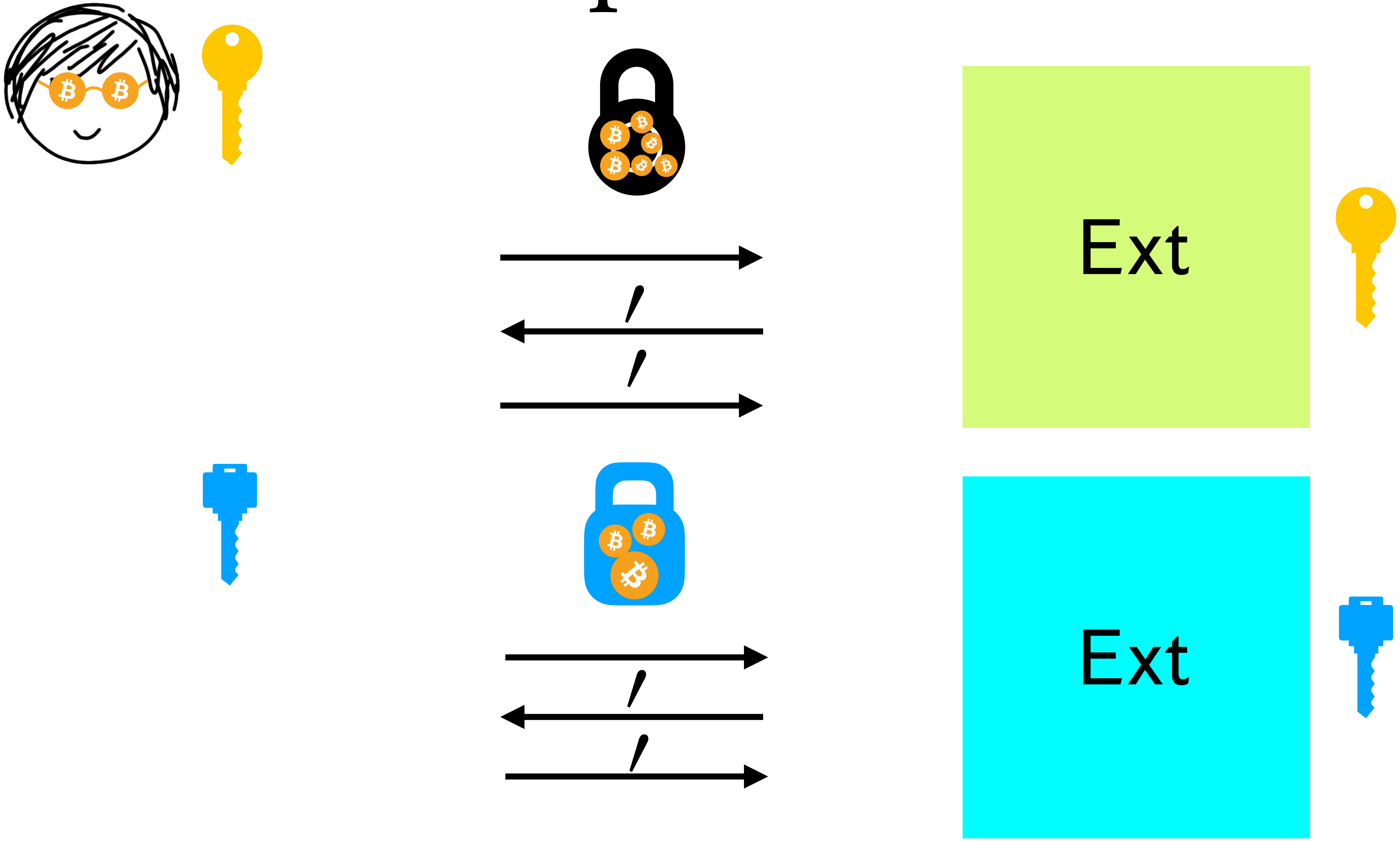
Composable?



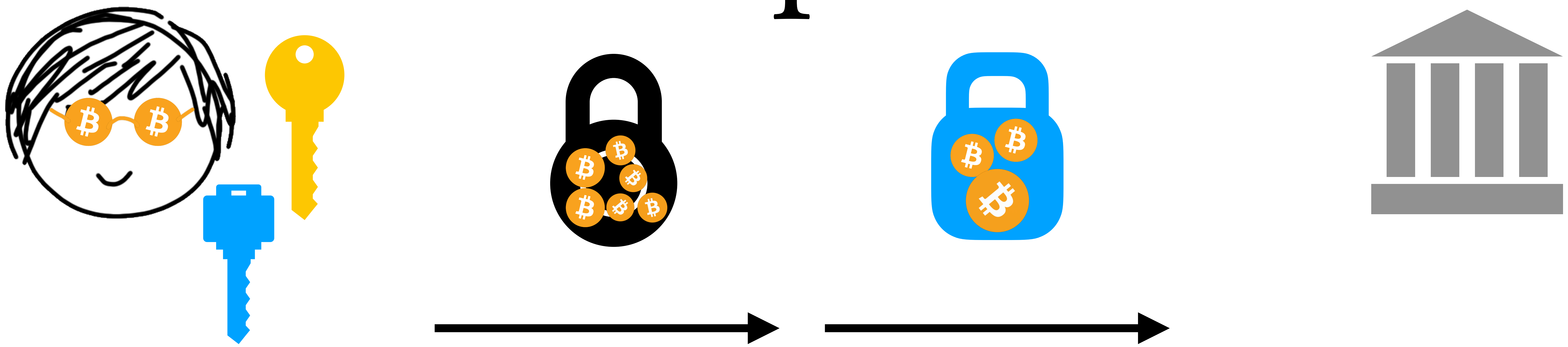
Composable?



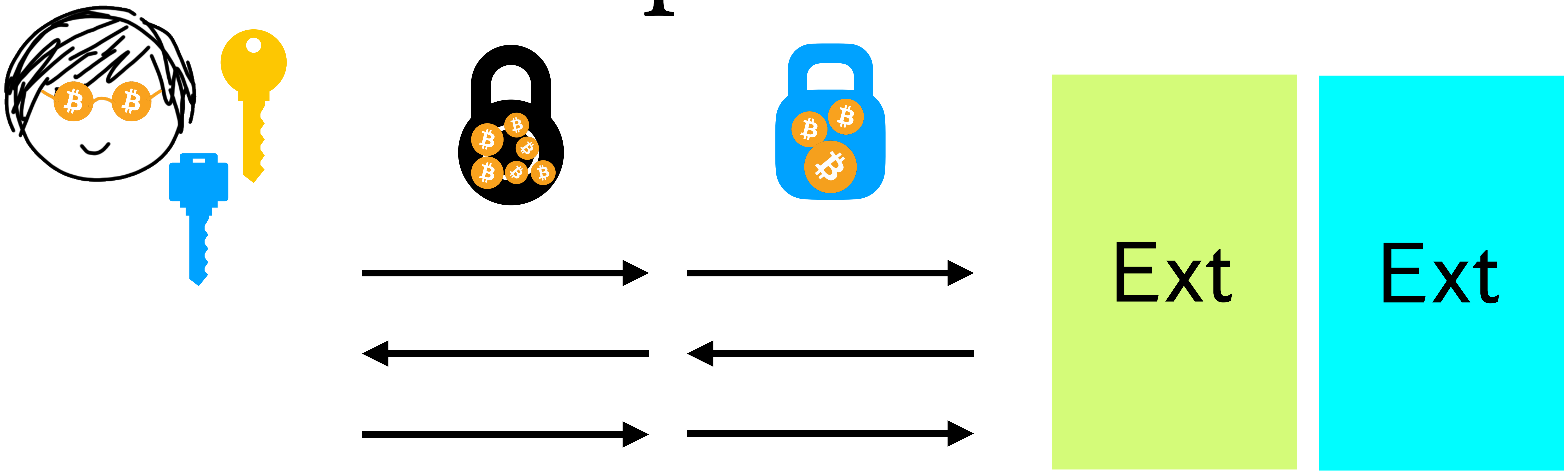
Composable?



Composable?



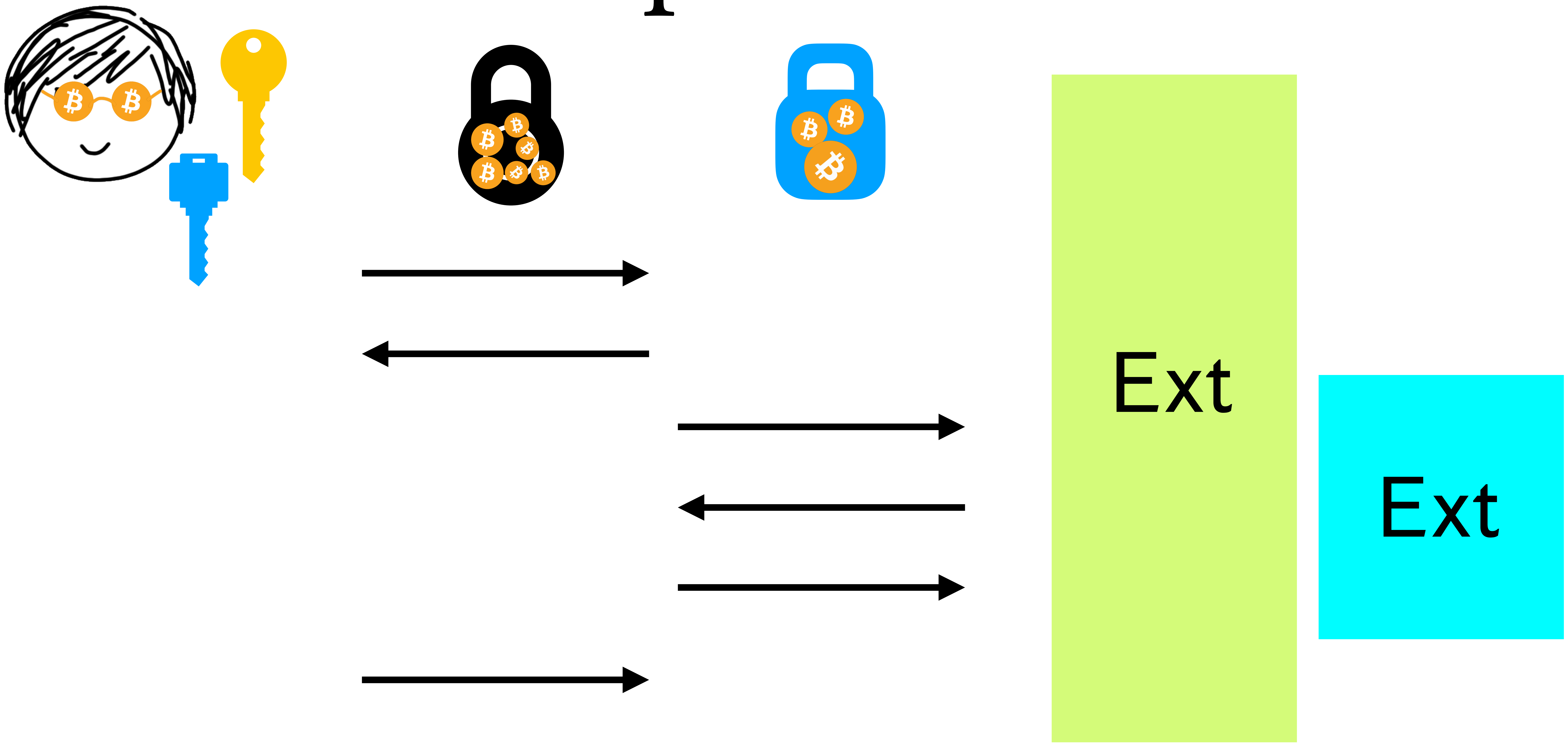
Composable?



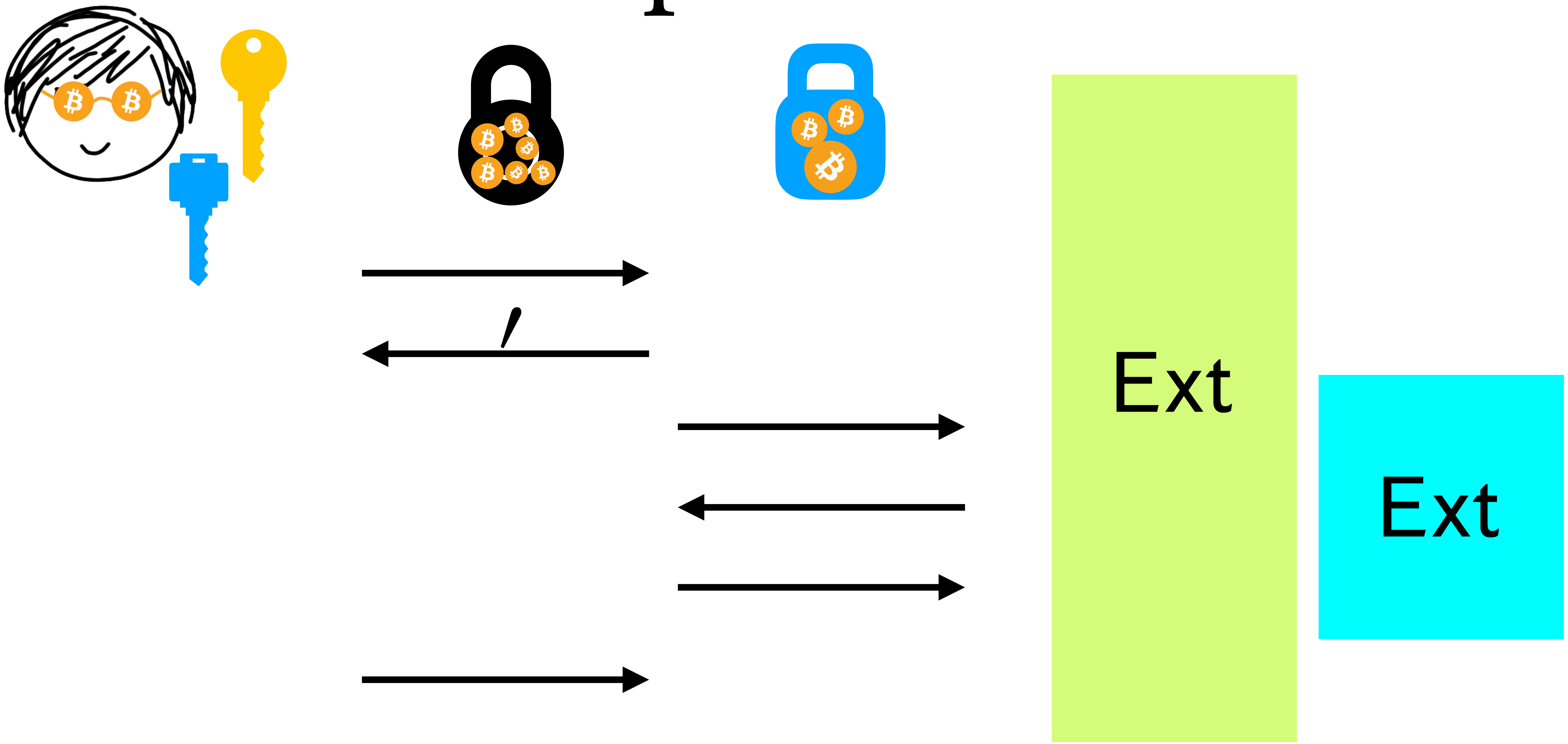
Ext

Ext

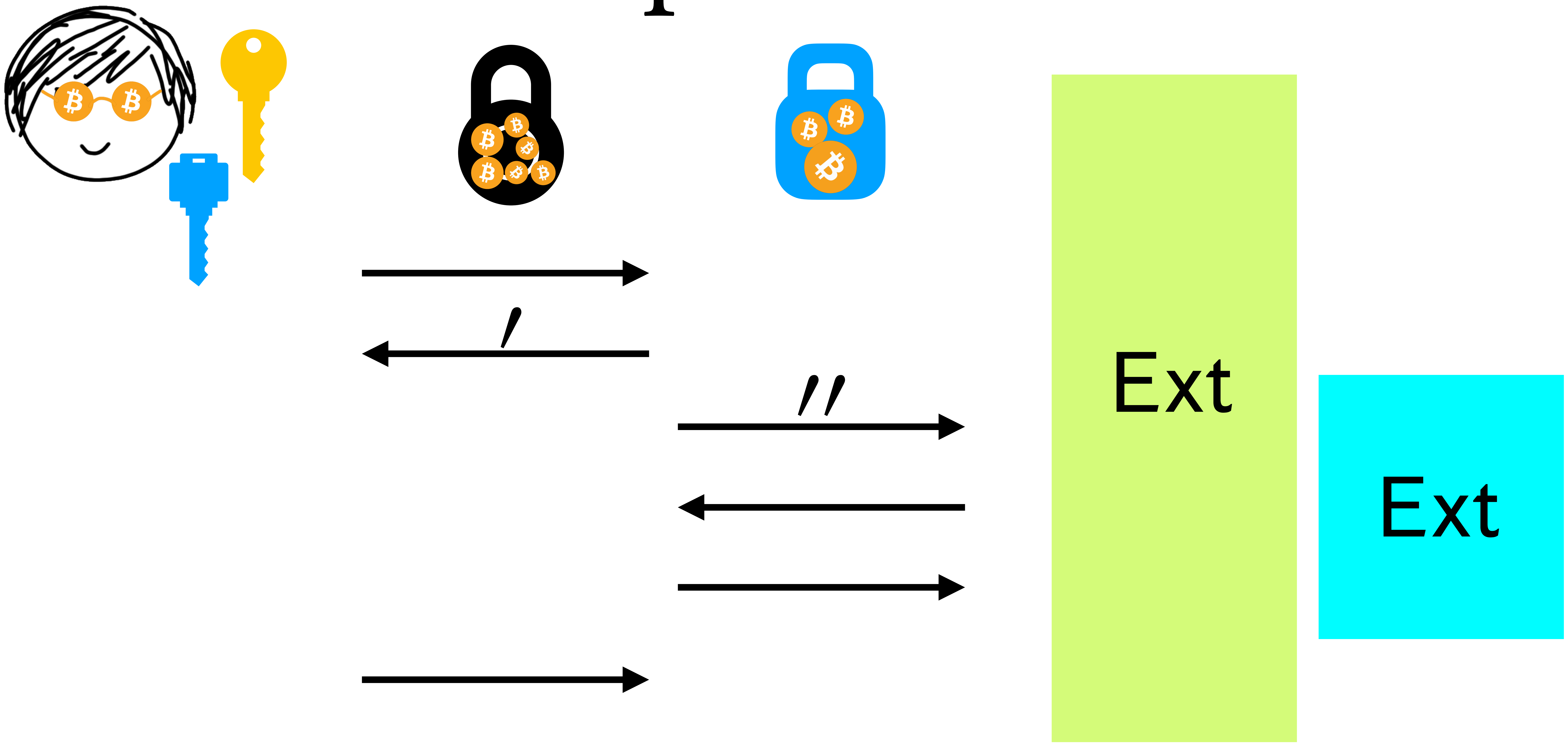
Composable?



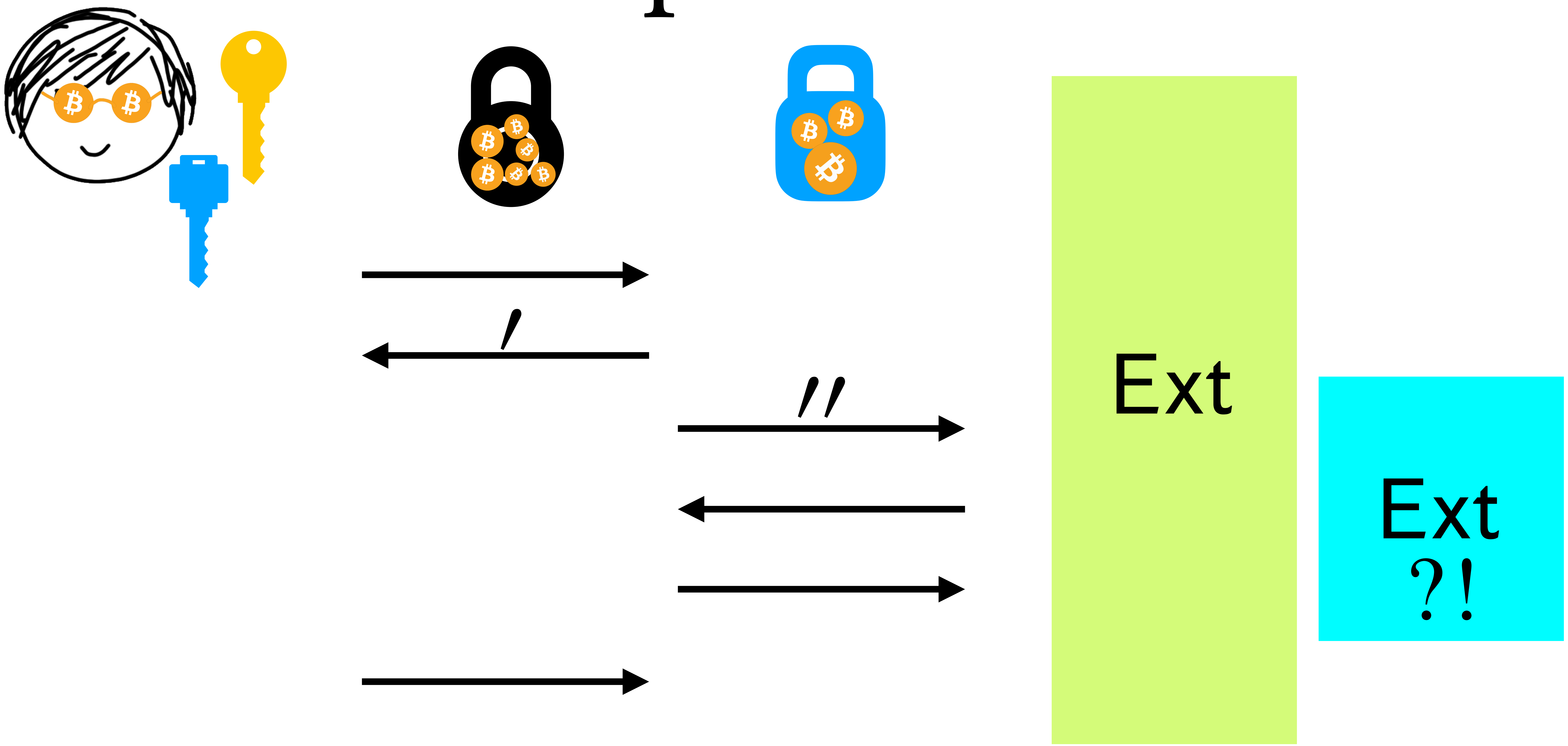
Composable?



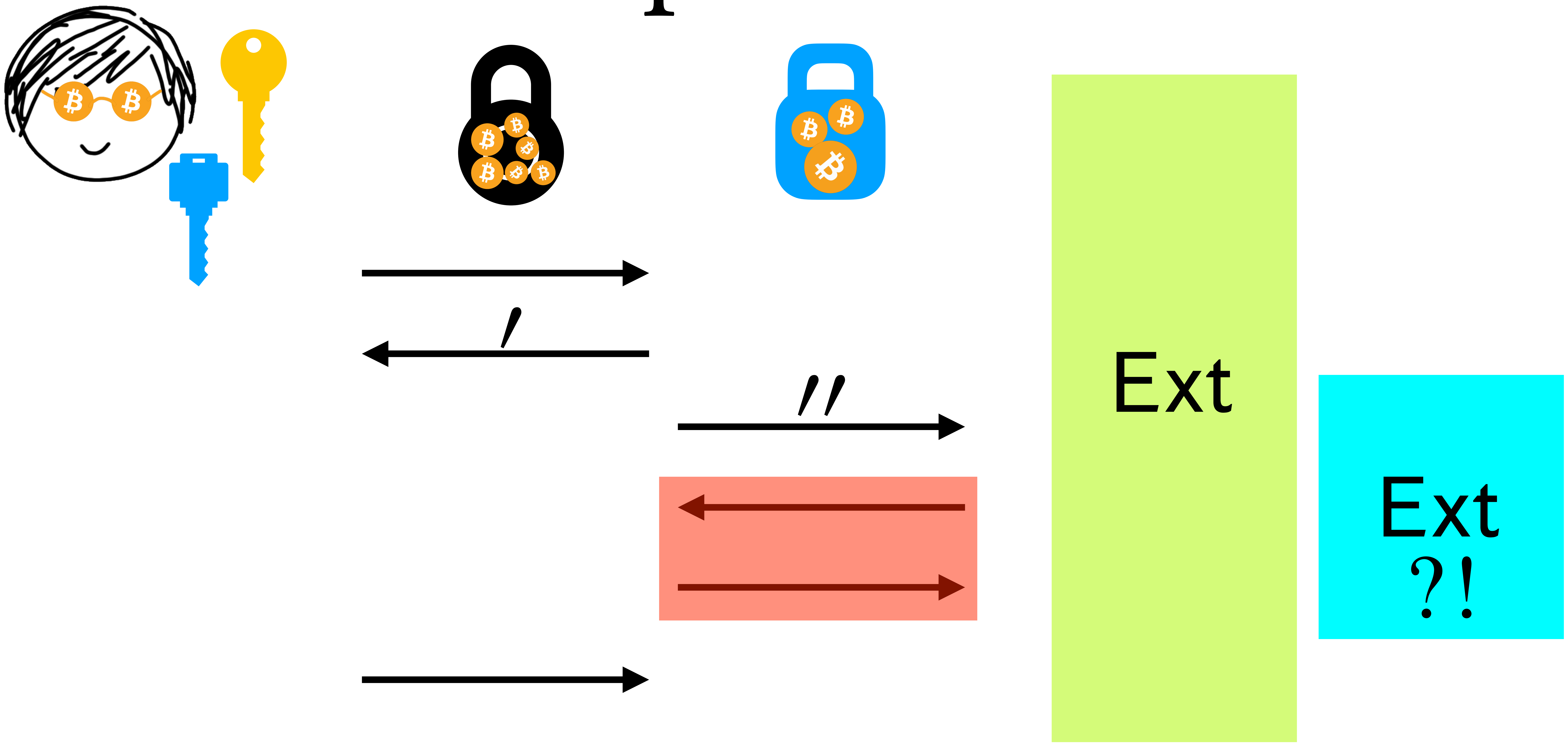
Composable?



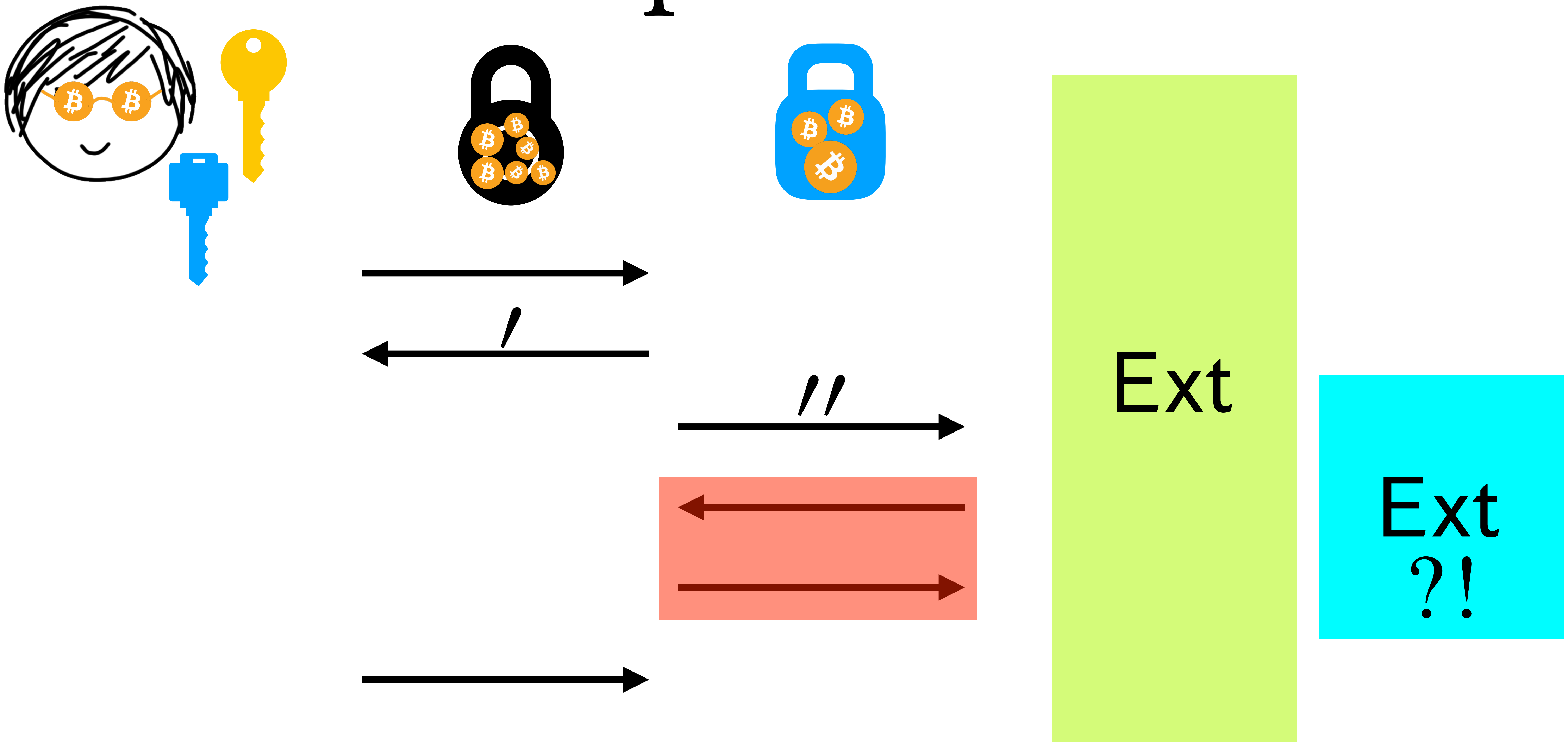
Composable?



Composable?



Composable?



Rewinding extraction strategies are bad for concurrent composition

Straight-line Extraction

- What special privileges can we grant Ext that compose nicely?

- One option is a “Common Reference String”
  - i.e. system parameter for which Ext has a backdoor
  - Well studied, theoretically sound
  - Unsatisfying in practice; trusted generator needed

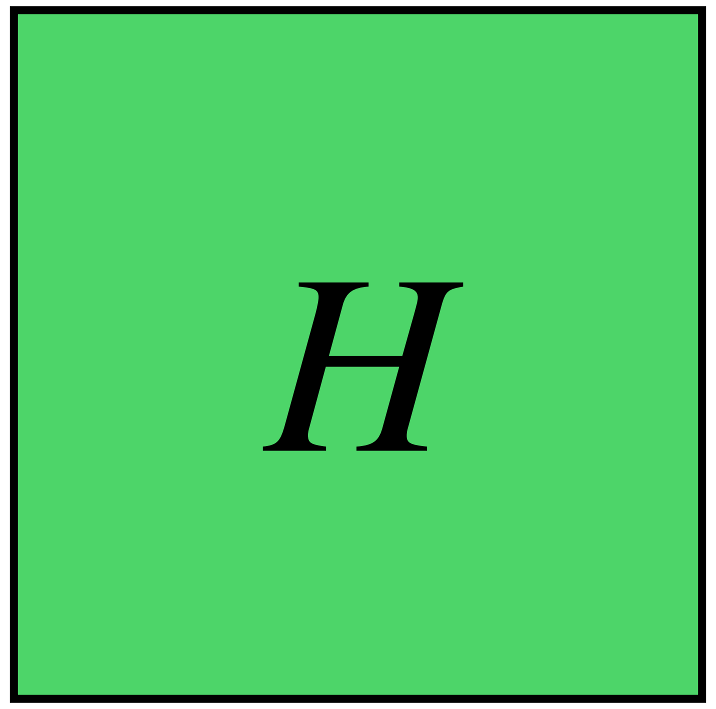
Composable

Non-interactive

Zero-knowledge Proofs

in the Random Oracle Model

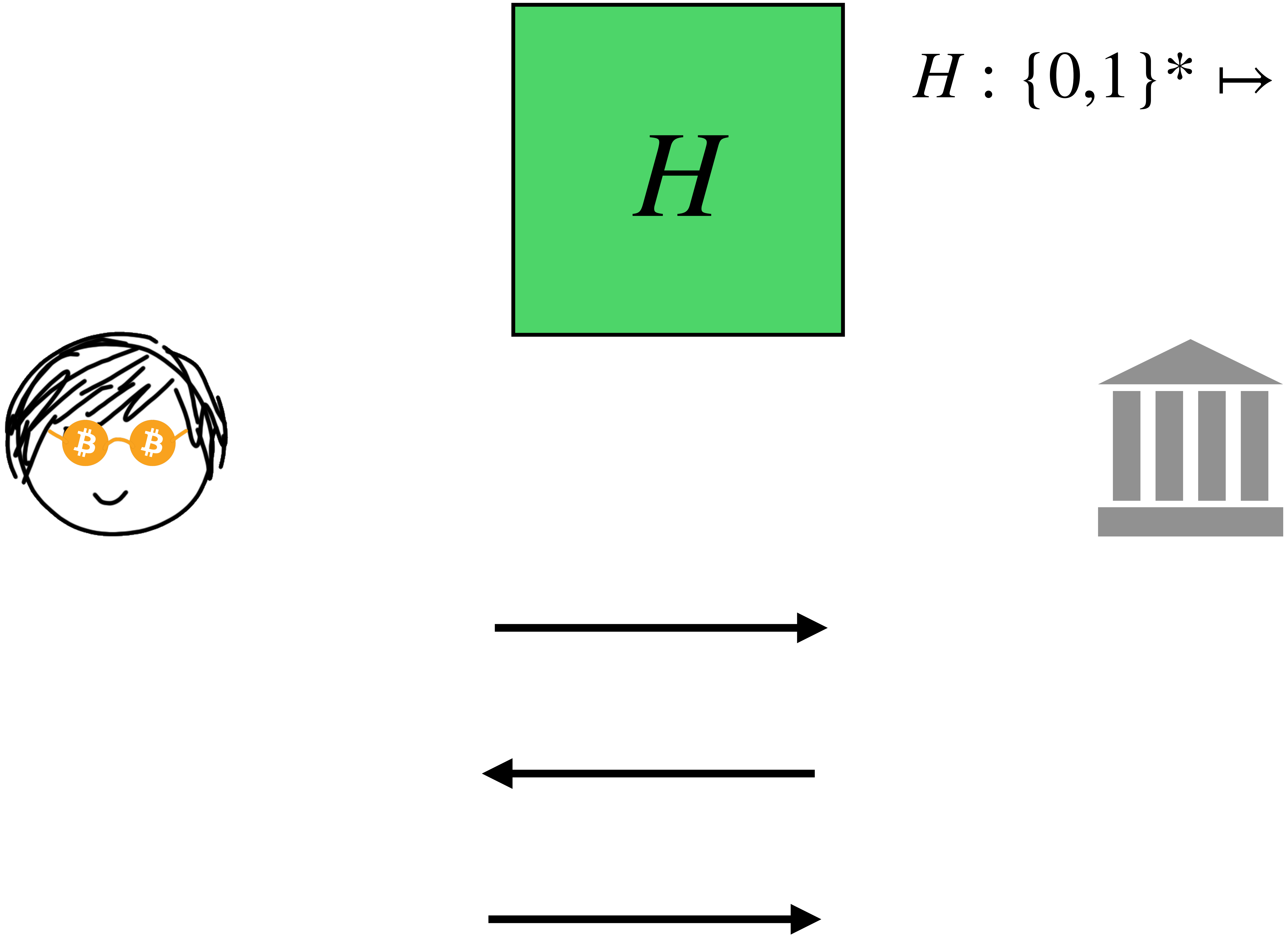
Random Oracle Model



H

$$H : \{0,1\}^* \mapsto \{0,1\}^\ell$$

Random Oracle Model



$H : \{0,1\}^* \mapsto \{0,1\}^\ell$

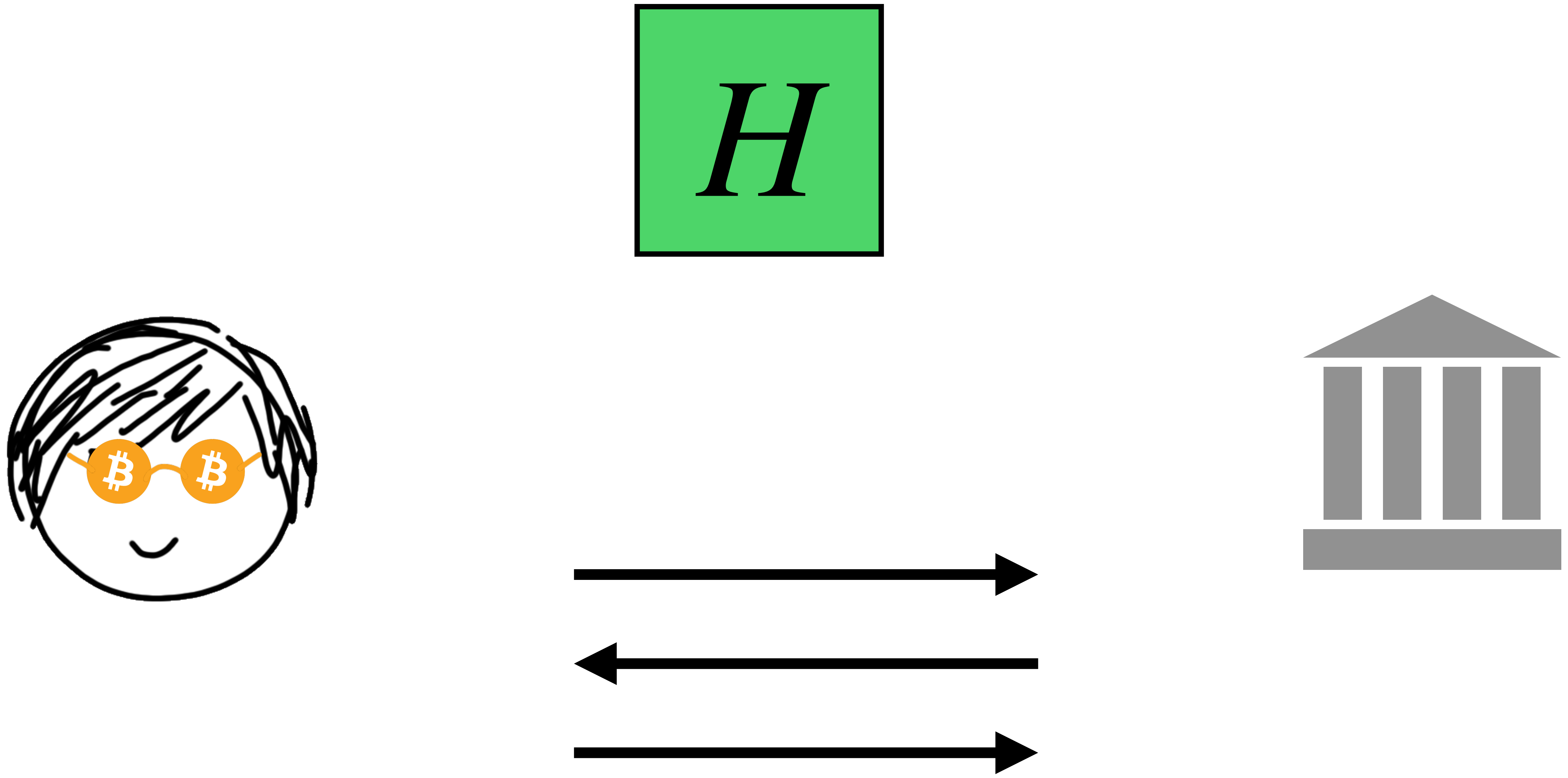
Random Oracle Model

- Began as a heuristic to analyze protocols that use cryptographic hash functions

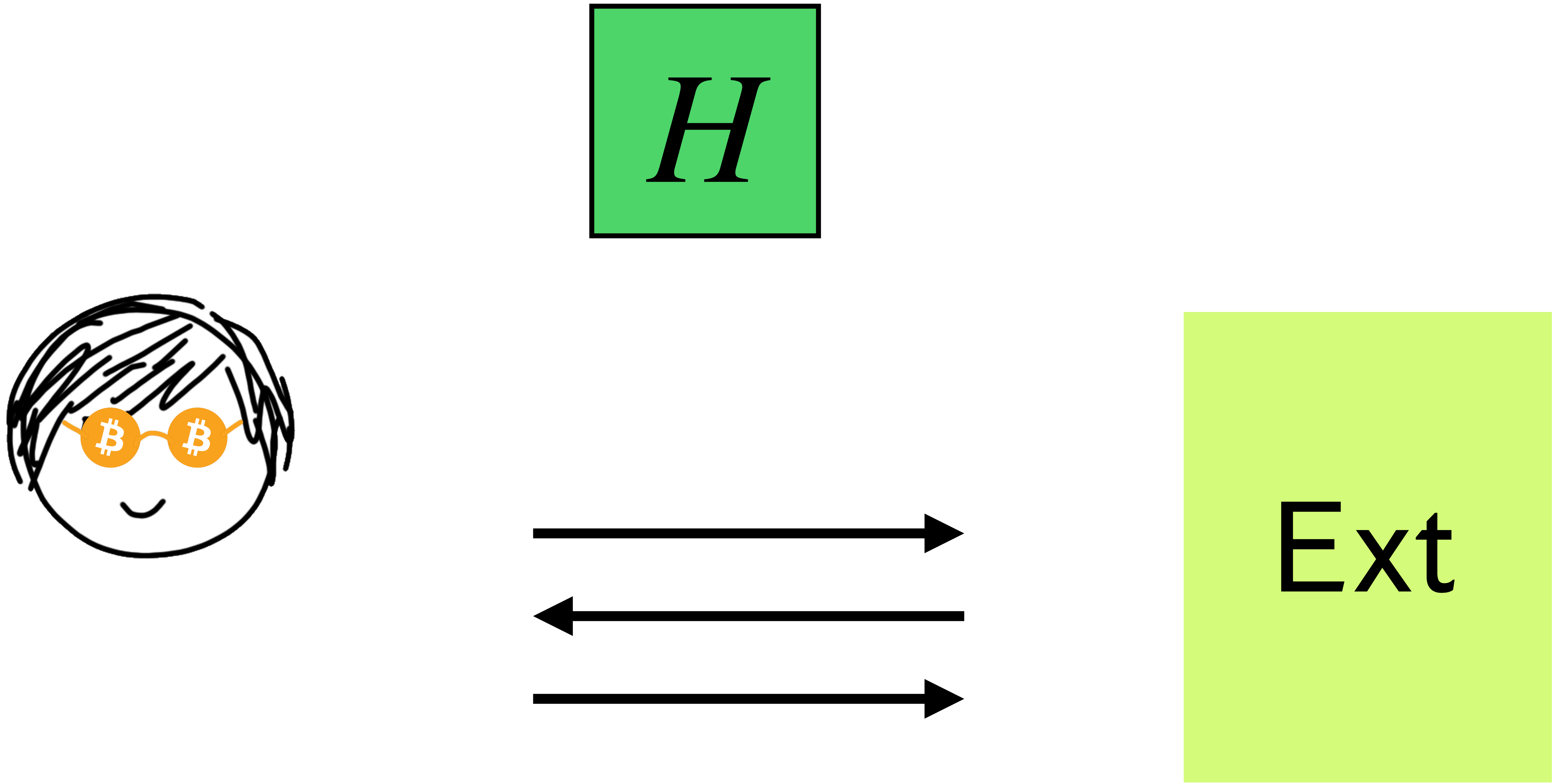
- Developed as a methodology to design efficient protocols with meaningful provable guarantees

- <u>Intuition</u>:
  - Cryptographic hashes are complex and highly unstructured
  - Unless you evaluate $H(x)$ from scratch, it looks random

Random Oracles as Ext Privilege



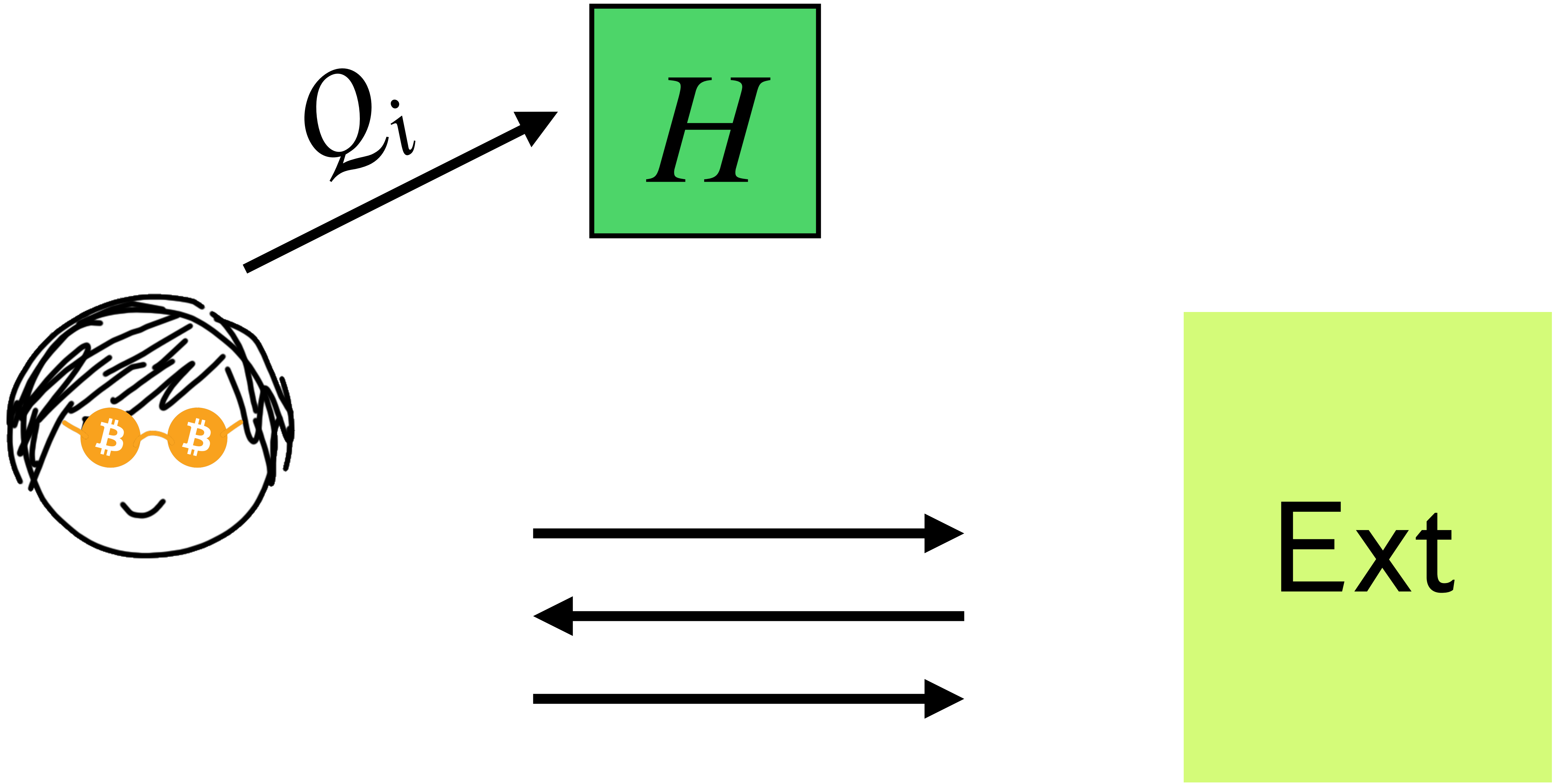
Random Oracles as Ext Privilege



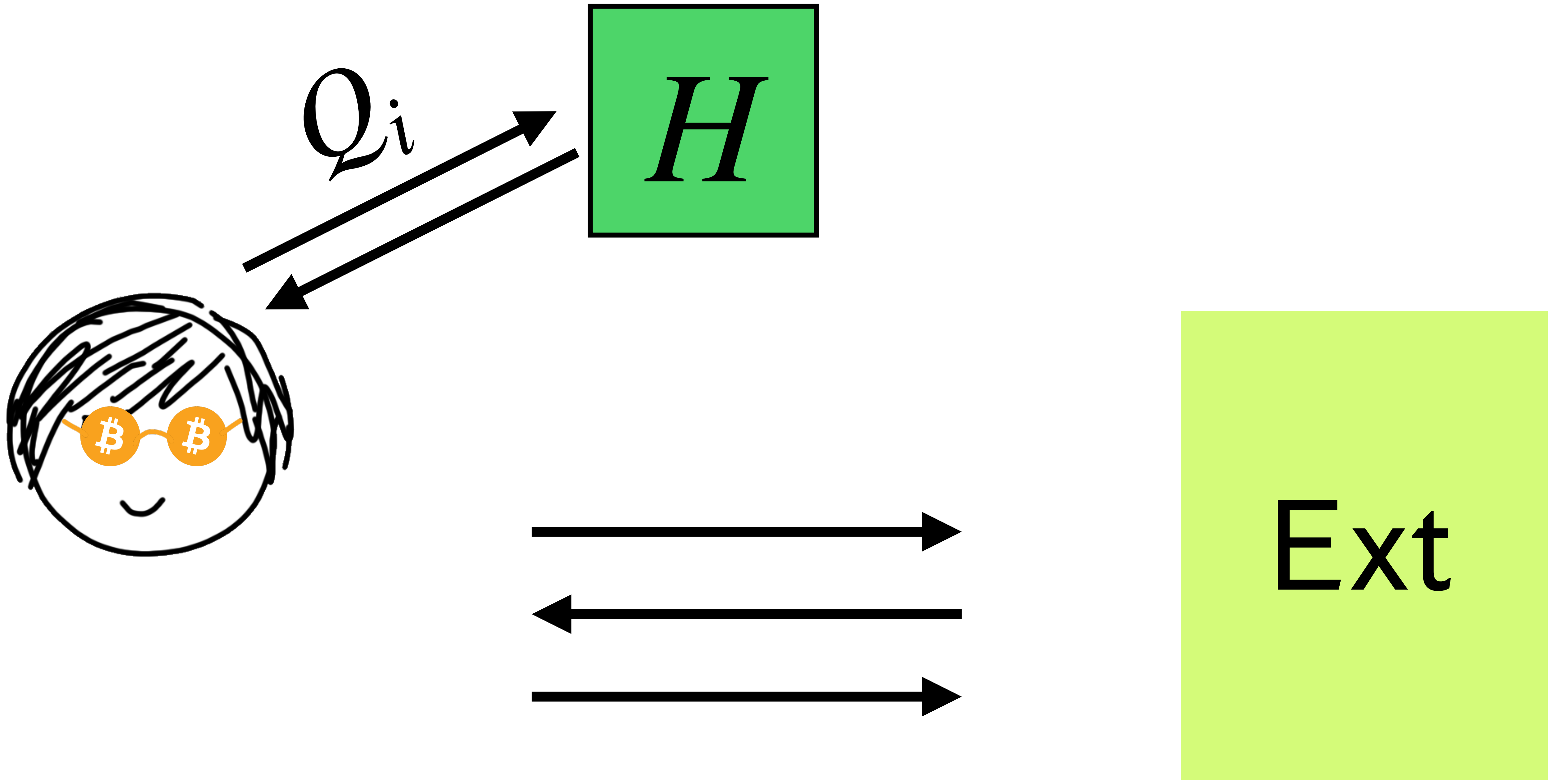
H

Ext

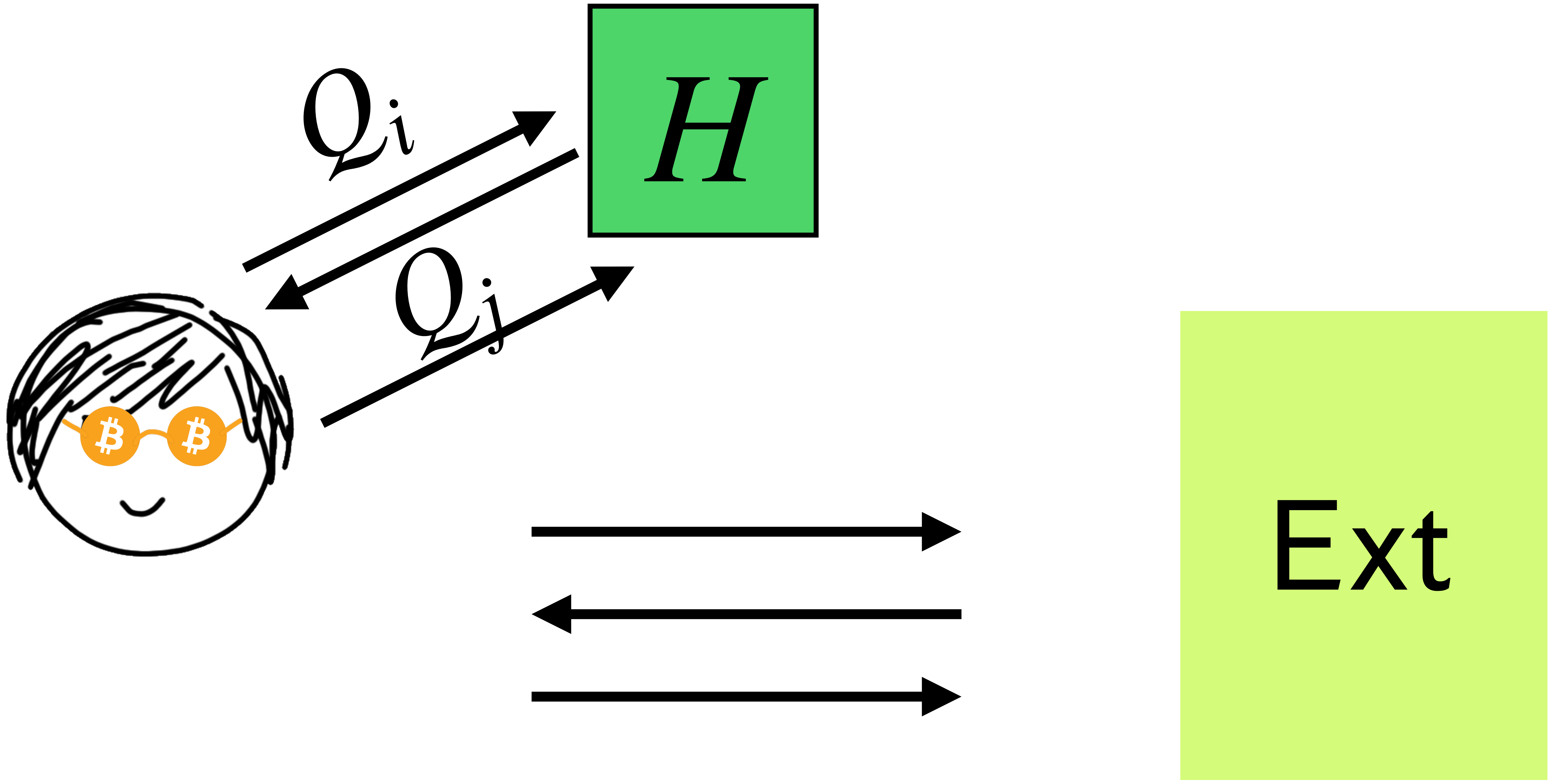
Random Oracles as Ext Privilege



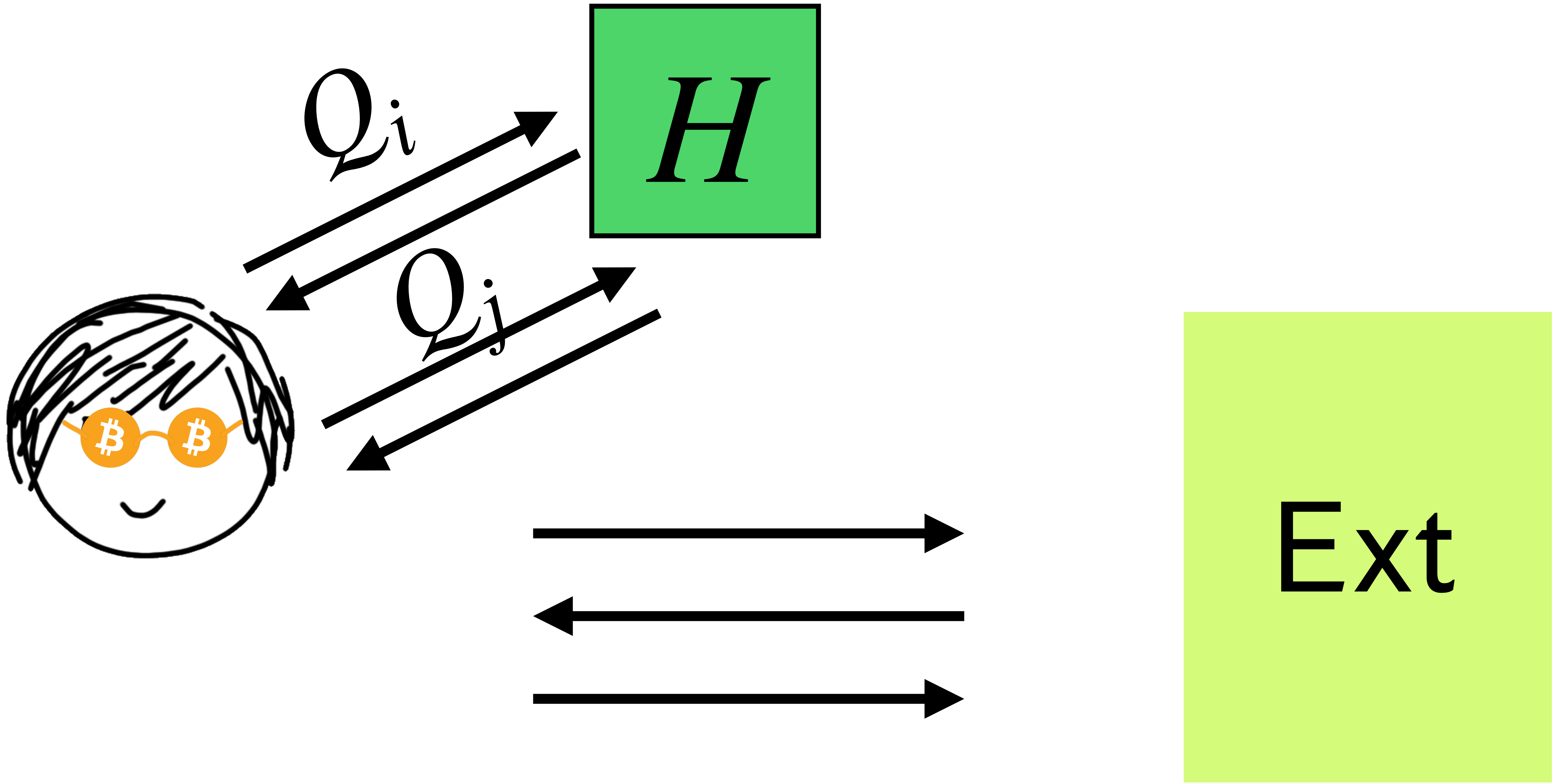
Random Oracles as Ext Privilege



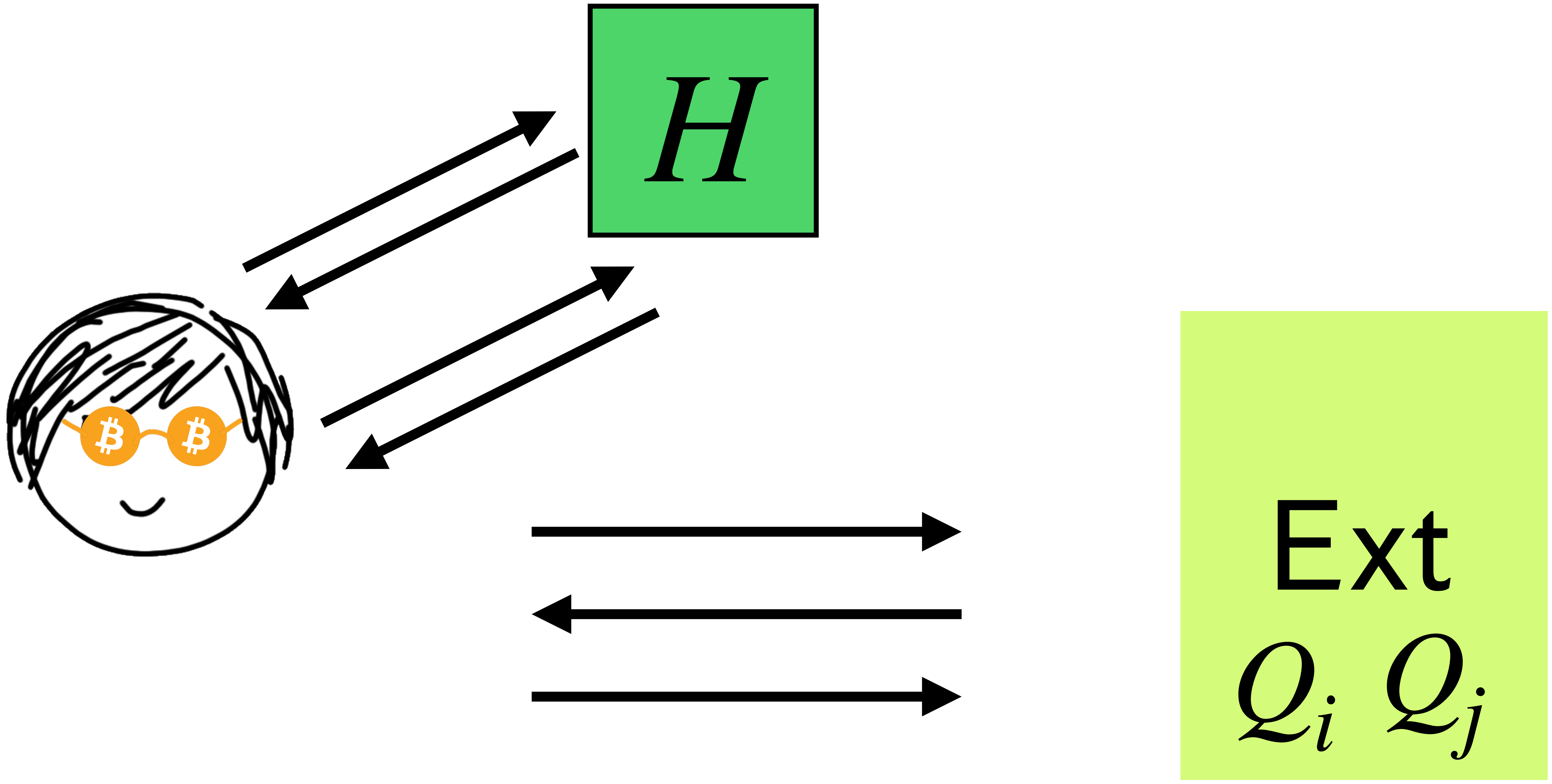
Random Oracles as Ext Privilege



Random Oracles as Ext Privilege

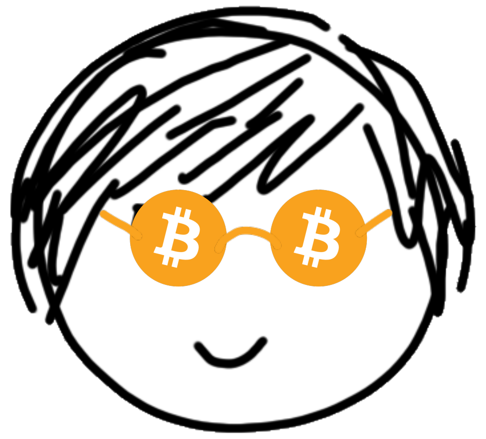


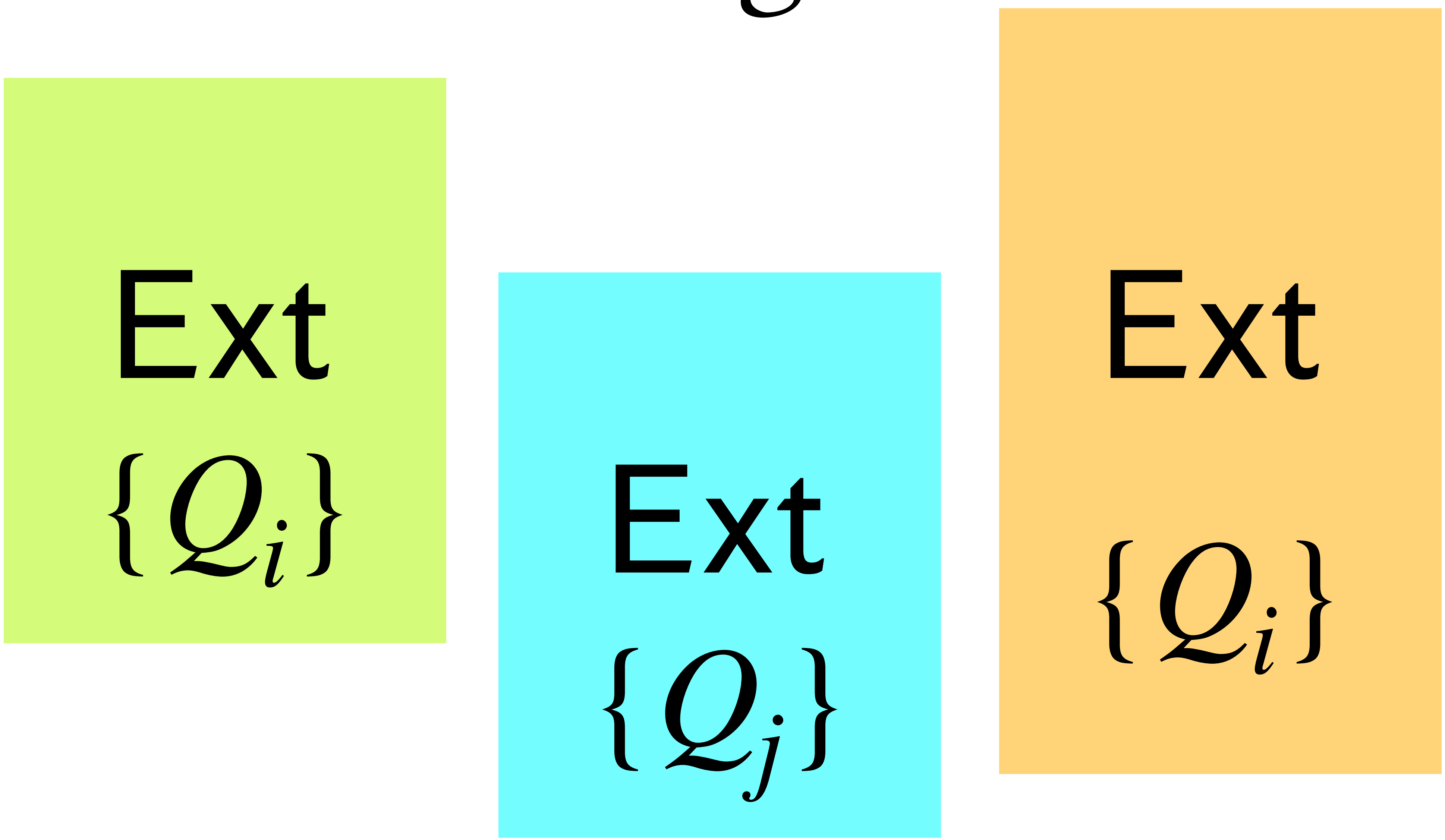
Random Oracles as Ext Privilege



Random Oracles as Ext Privilege

- Bob "knows" all of the $\{Q_i\}$ values queried to H
- Ext could obtain useful information from $\{Q_i\}$
- $\{Q_i\}$ can be obtained without rewinding





Ext $\{Q_i\}$

Ext $\{Q_j\}$

Ext $\{Q_i\}$

Composable

Non-interactive

Zero-knowledge Proofs

in the Random Oracle Model

Non-interactive

- As the name suggests, a non-interactive proof is a single message protocol

- Useful communication pattern for many applications

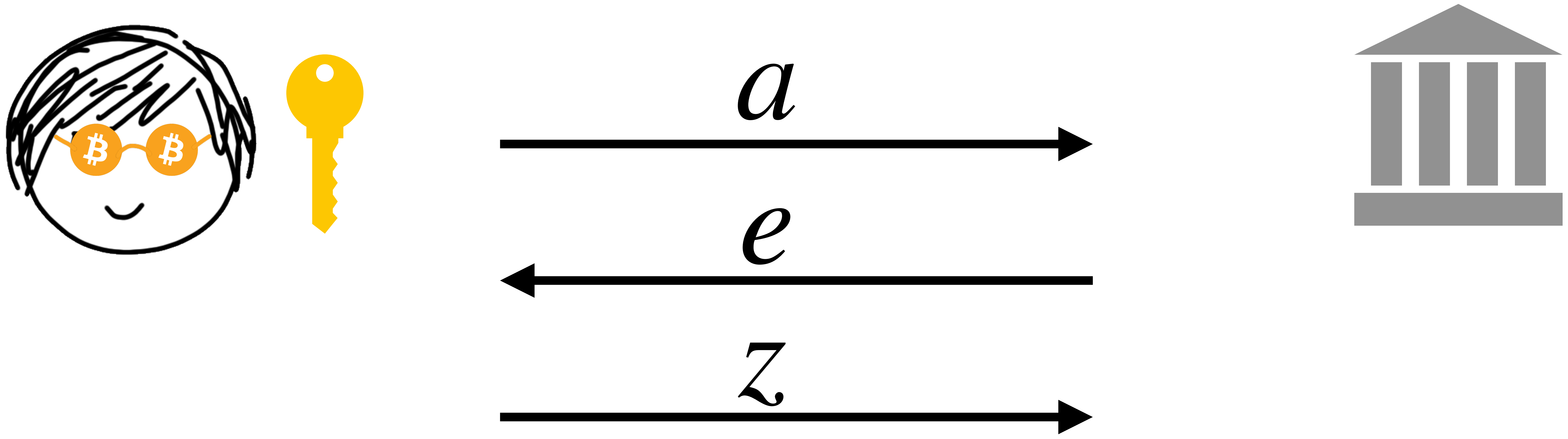
- Common methodology: compile Σ protocol

- [Pass 03] gave a simple straight-line extractable compiler in the random oracle model

Fischlin’s Compiler

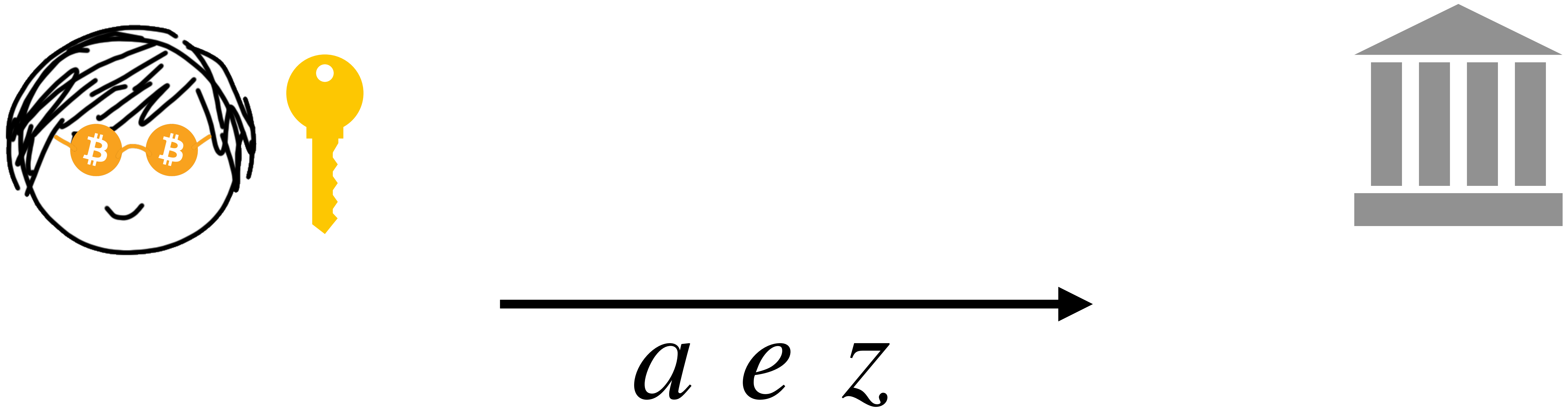
- [Fischlin 05] gave a much more efficient compiler in the same model as [Pass 03]

- More interesting to analyze, and has remained the state of the art for $\Sigma \mapsto \mathsf{NIZK}$ compilers



Fischlin’s Compiler

- [Fischlin 05] gave a much more efficient compiler in the same model as [Pass 03]
- More interesting to analyze, and has remained the state of the art for Σ ↦ NIZK compilers

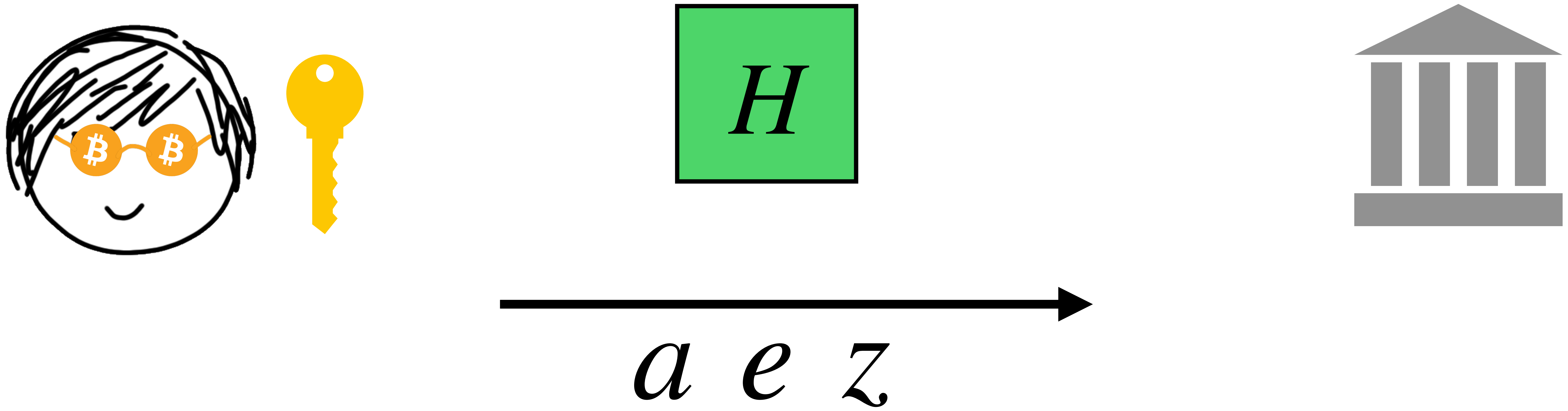


$a \quad e \quad z$

Fischlin’s Compiler

- [Fischlin 05] gave a much more efficient compiler in the same model as [Pass 03]

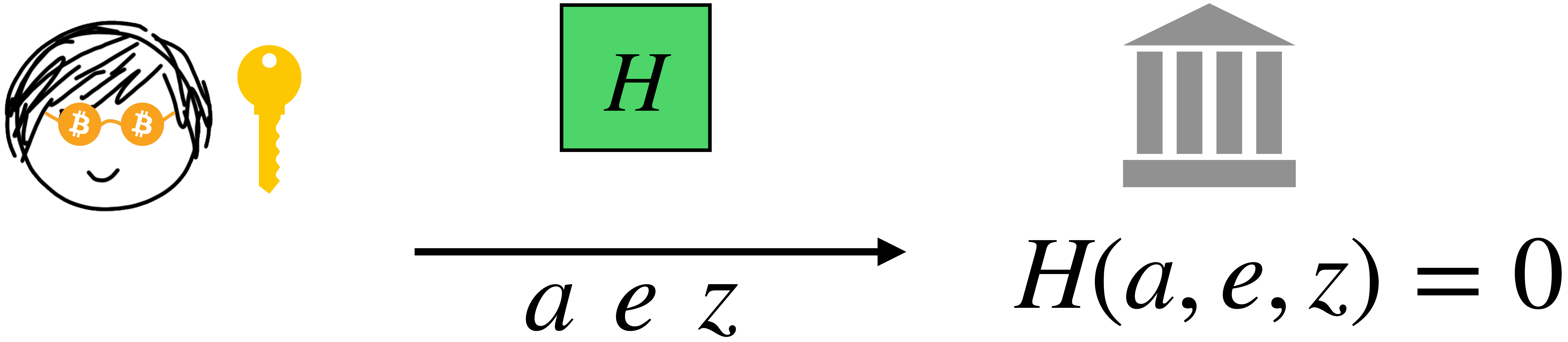
- More interesting to analyze, and has remained the state of the art for $\Sigma \mapsto$ NIZK compilers



Fischlin's Compiler

- [Fischlin 05] gave a much more efficient compiler in the same model as [Pass 03]

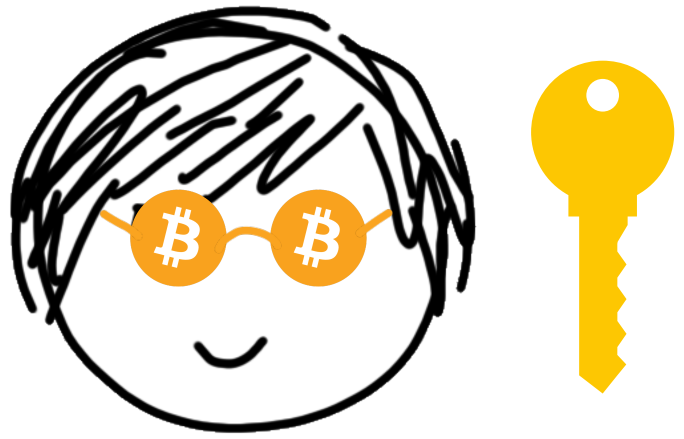
- More interesting to analyze, and has remained the state of the art for $\Sigma \mapsto$ NIZK compilers

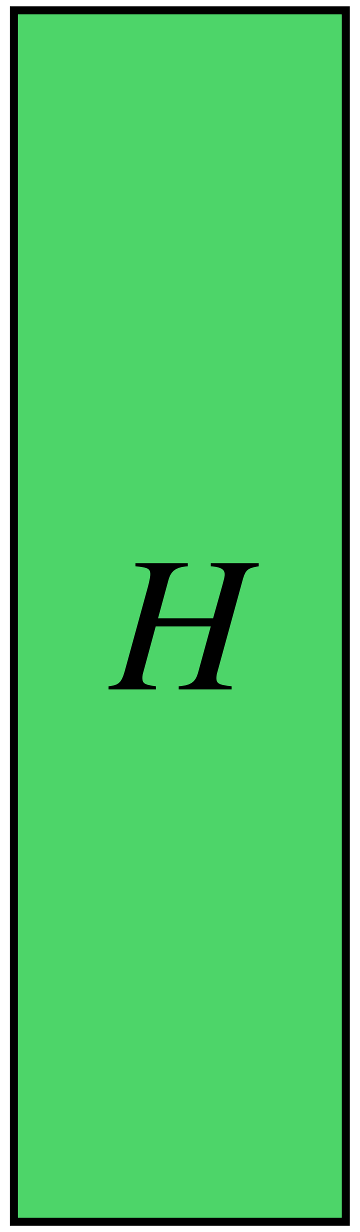


$$H(a, e, z) = 0$$

Fischlin's Transformation

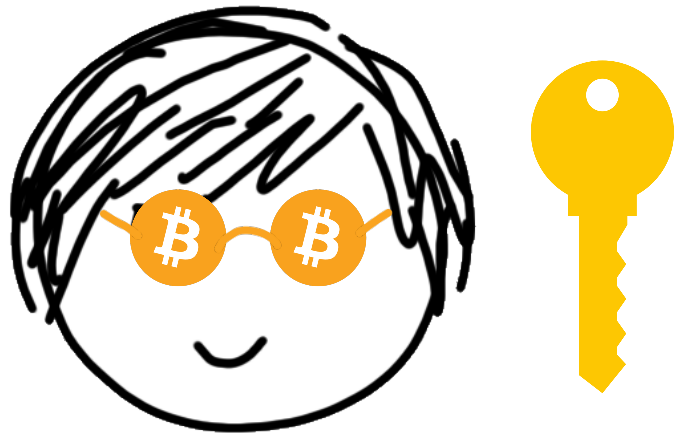
- Let $\boxed{H}: \{0,1\}^* \mapsto \{0,1\}^\ell$ be a random oracle



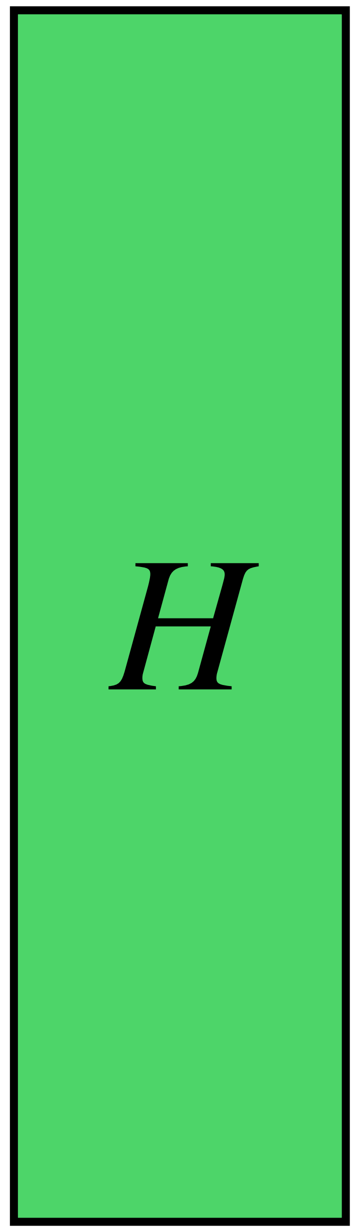


Fischlin's Transformation

- Let $\boxed{H}$: $\{0,1\}^* \mapsto \{0,1\}^\ell$ be a random oracle

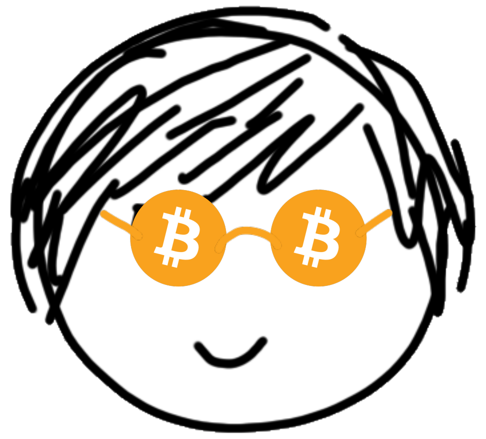


Sample Σ-protocol first message 'a'



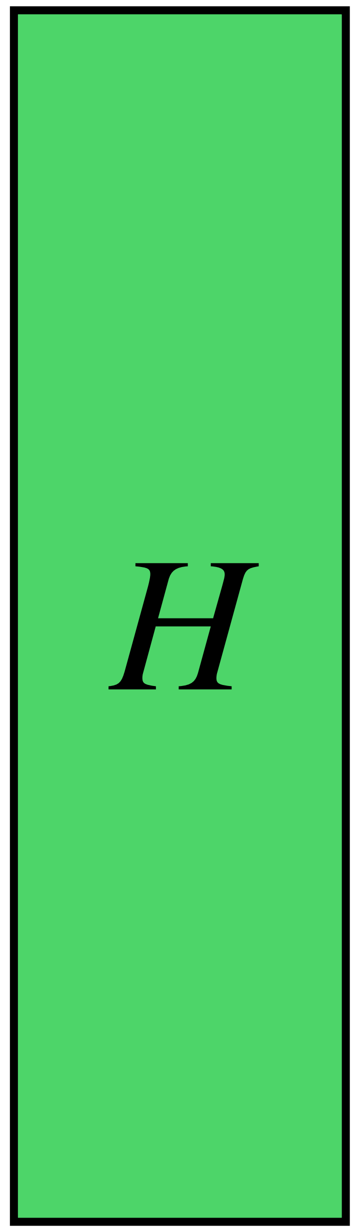
Fischlin's Transformation

- Let $\boxed{H}$: $\{0,1\}^* \mapsto \{0,1\}^\ell$ be a random oracle



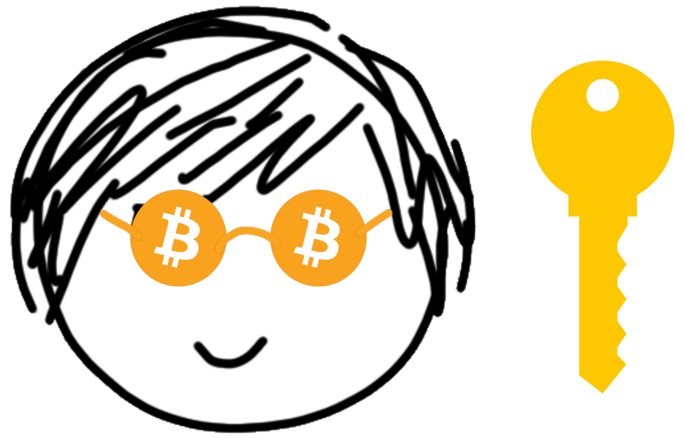
Sample Σ-protocol first message 'a'

$(a,0,z_0)$

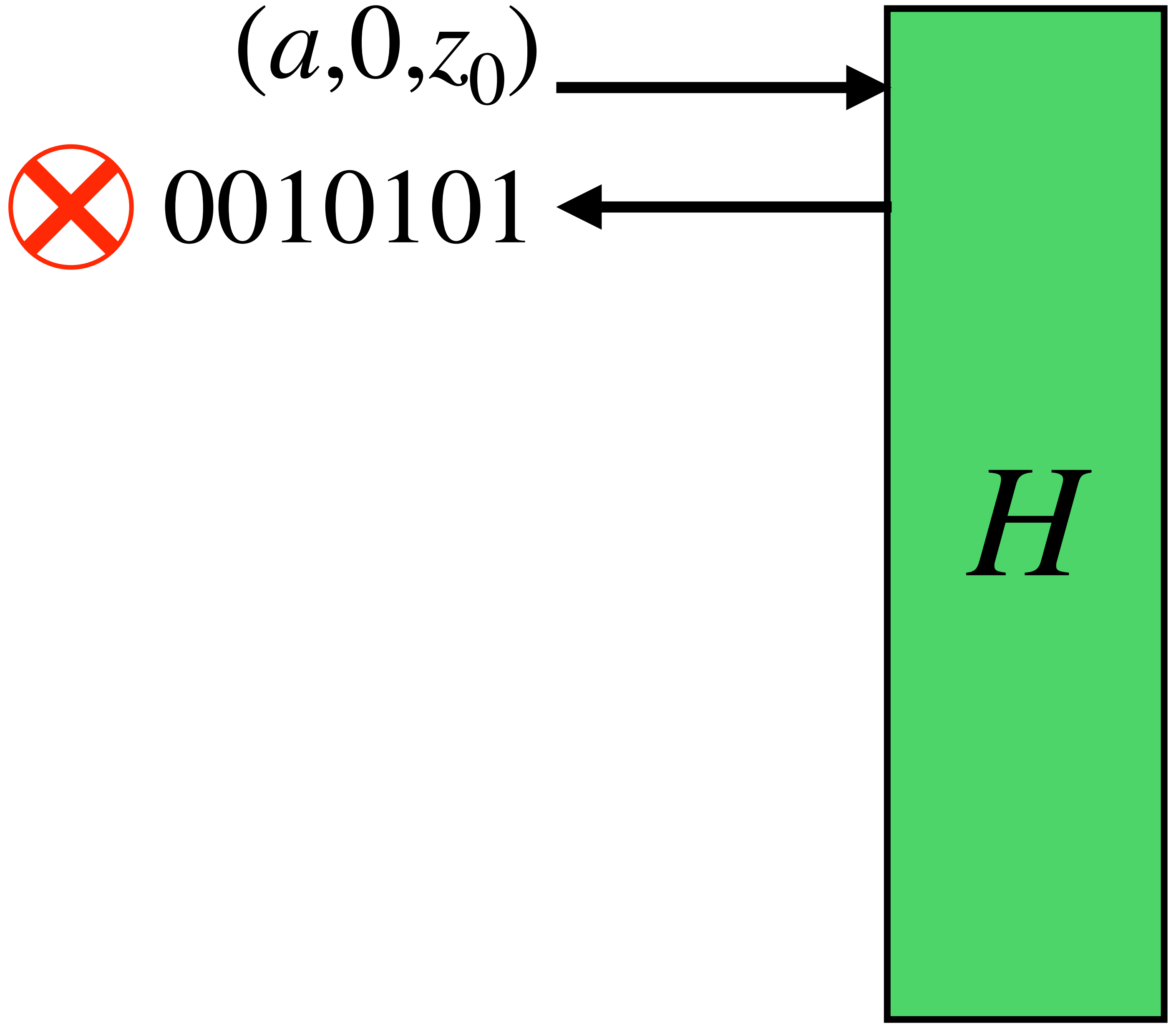


Fischlin's Transformation

- Let $\boxed{H}$: $\{0,1\}^* \mapsto \{0,1\}^\ell$ be a random oracle

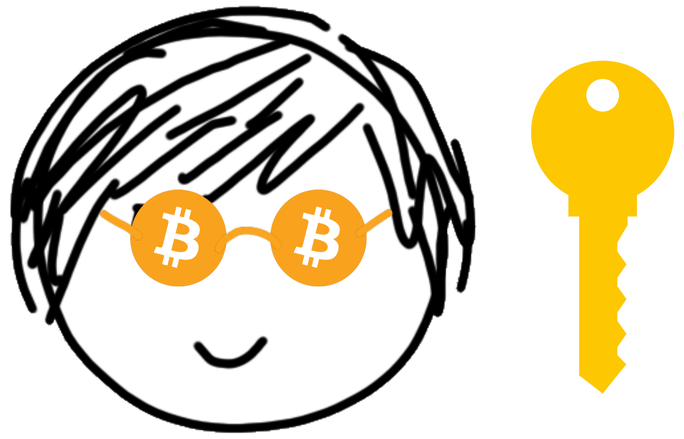


Sample Σ-protocol first message 'a'

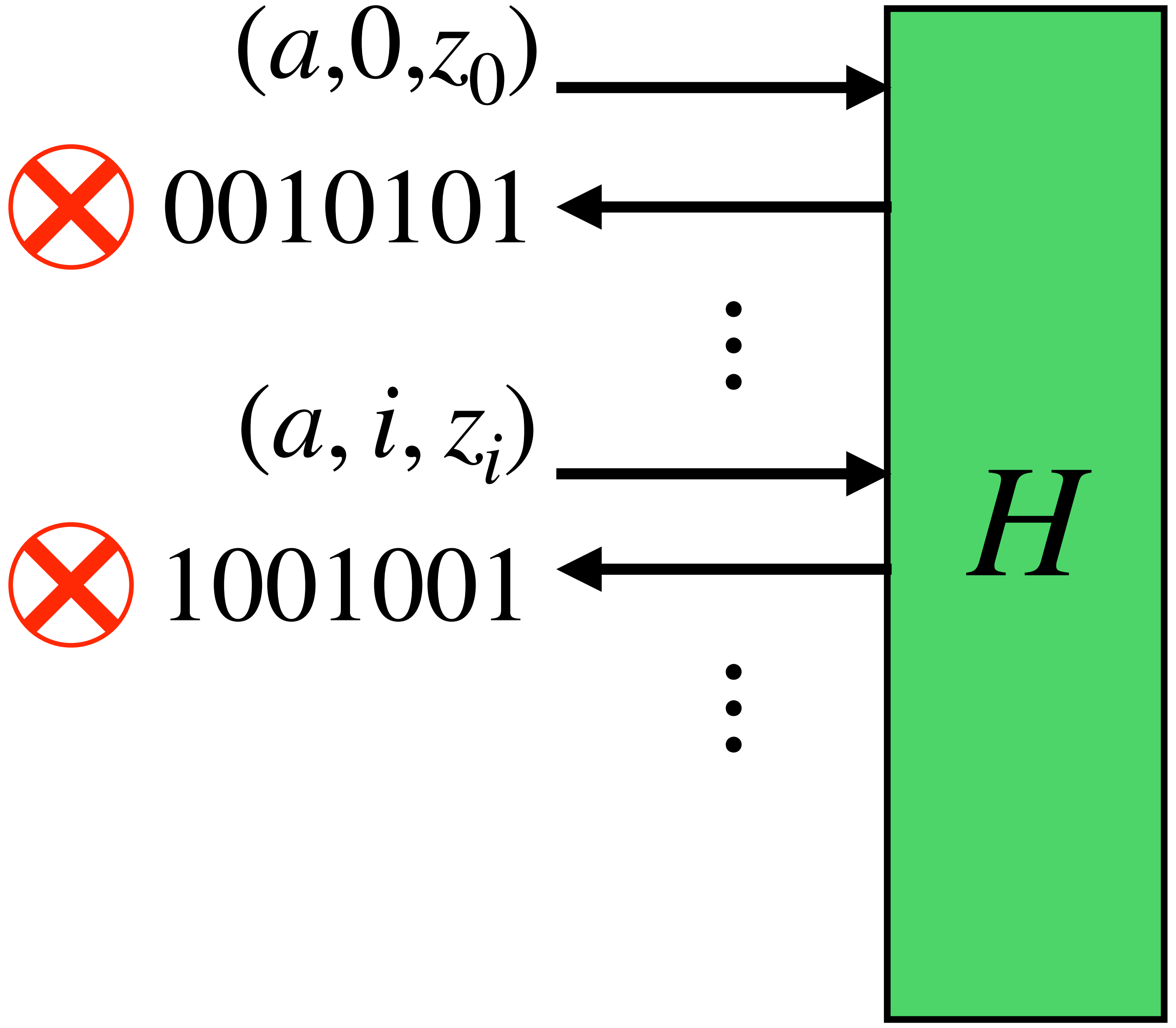


Fischlin's Transformation

- Let $\boxed{H}$: $\{0,1\}^* \mapsto \{0,1\}^\ell$ be a random oracle



Sample Σ-protocol first message 'a'



$(a,0,z_0)$

☒ 0010101

⋮

(a, i, z_i)

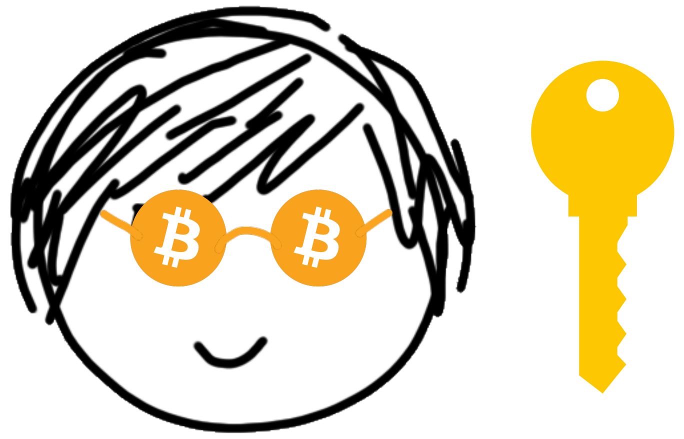
☒ 1001001

⋮

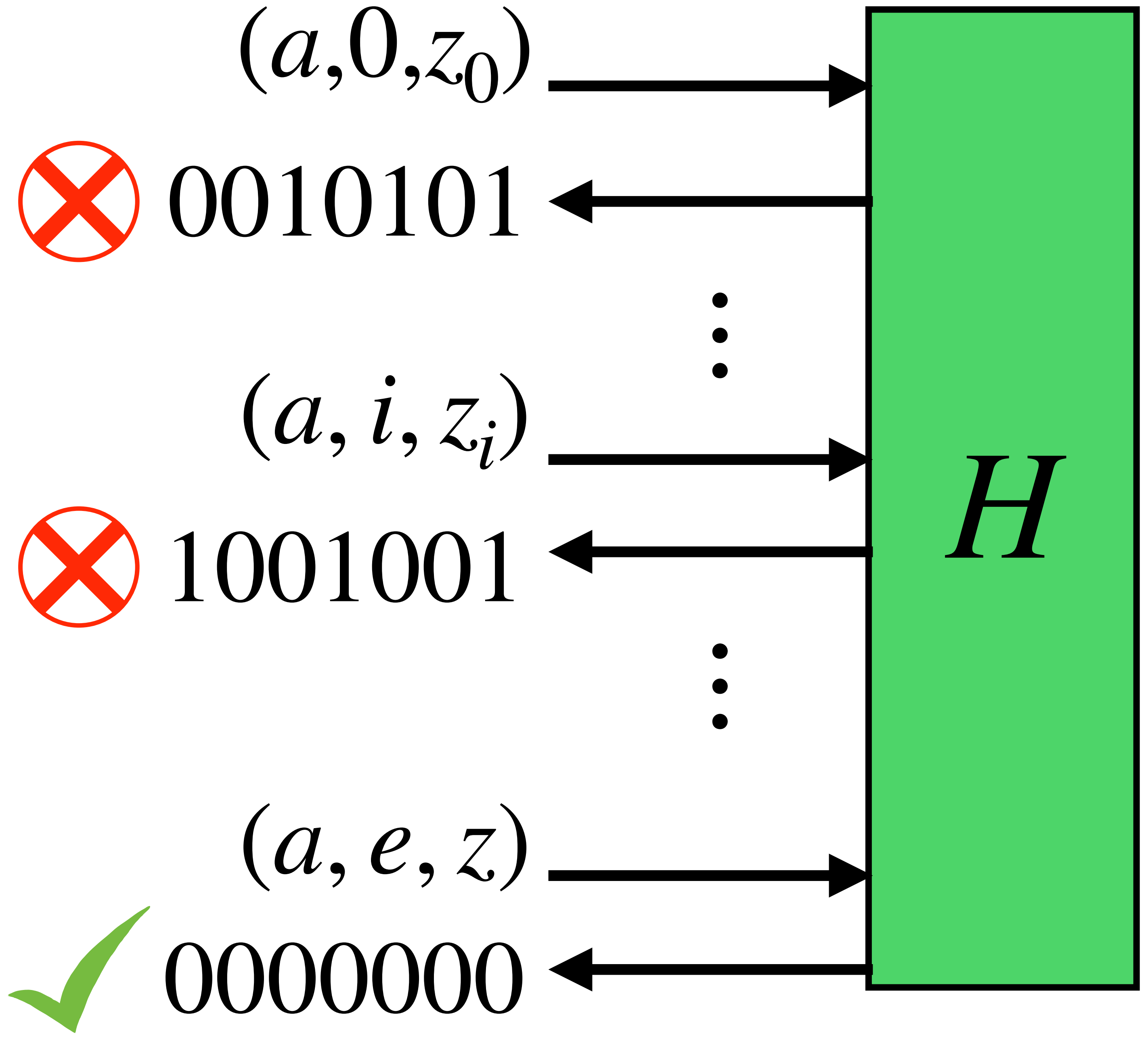
H

Fischlin's Transformation

- Let $\boxed{H}: \{0,1\}^* \mapsto \{0,1\}^\ell$ be a random oracle



Sample Σ-protocol first message 'a'



$(a,0,z_0)$

☒ 0010101

⋮

(a, i, z_i)

☒ 1001001

⋮

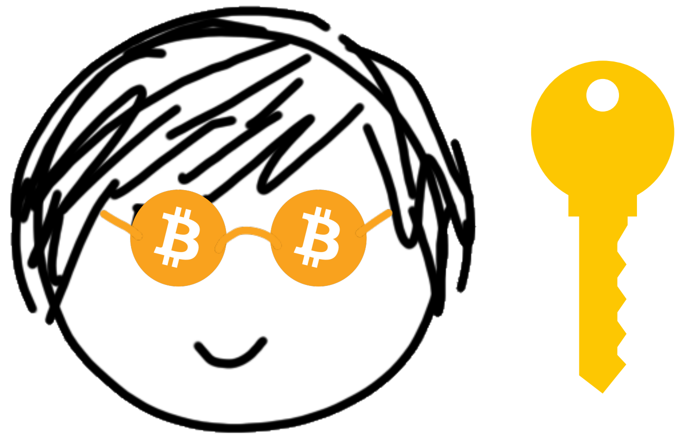
(a, e, z)

✓ 0000000

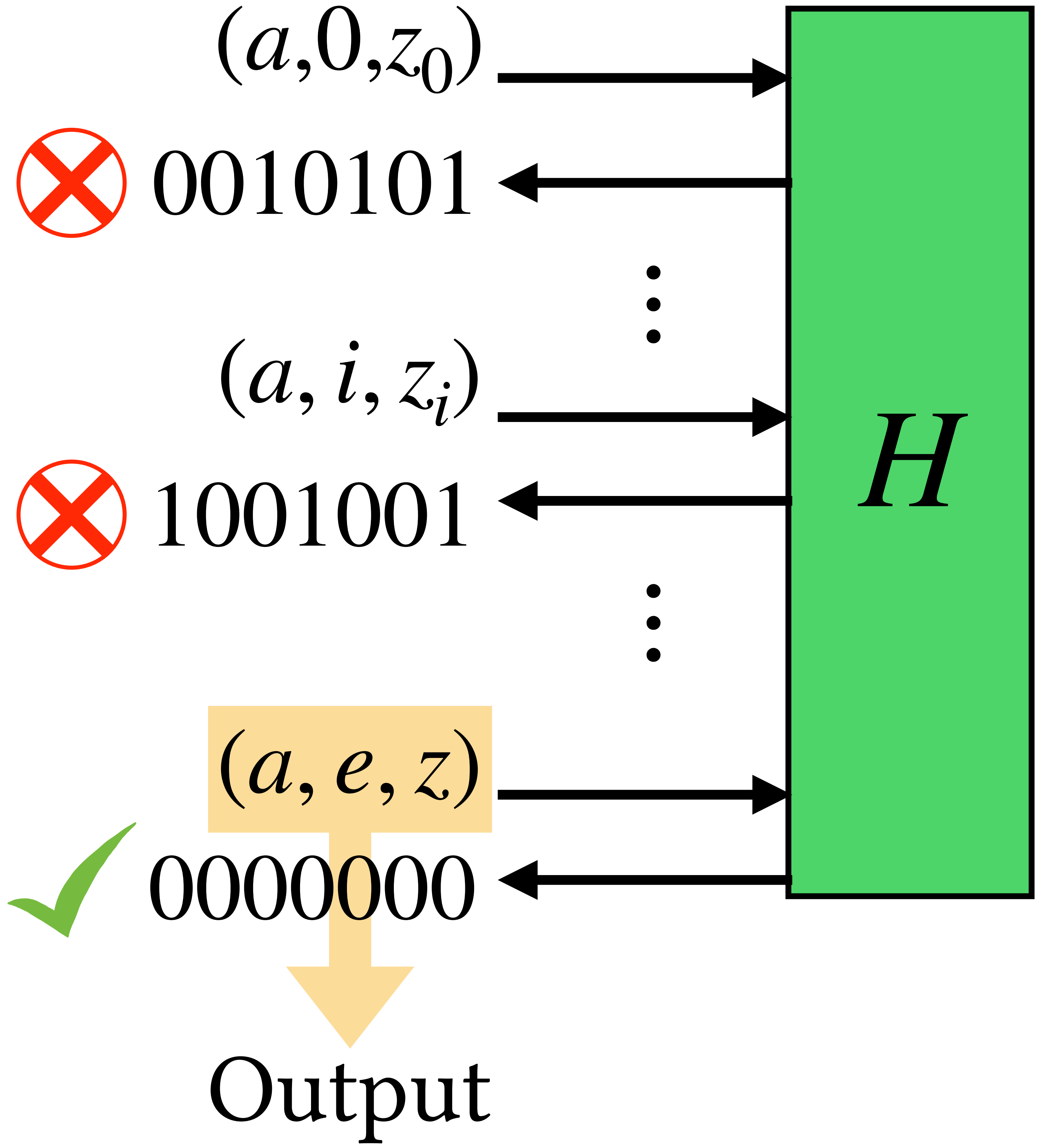
H

Fischlin's Transformation

- Let $\boxed{H}: \{0,1\}^* \mapsto \{0,1\}^{\ell}$ be a random oracle



Sample Σ-protocol first message 'a'



$(a,0,z_0)$

✗ 0010101

⋮

(a,i,z_i)

✗ 1001001

⋮

(a,e,z)

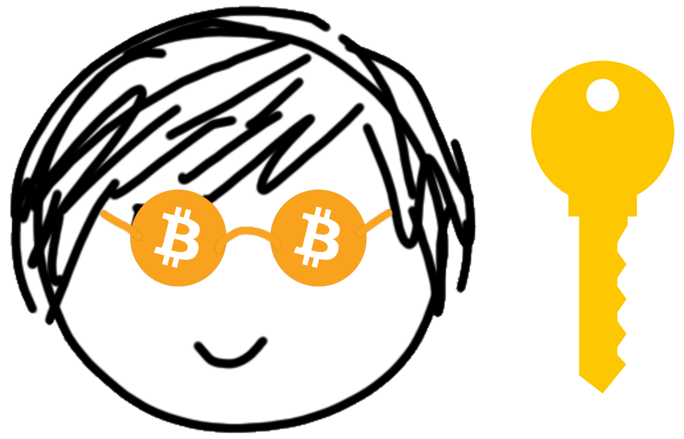
✓ 0000000

Output

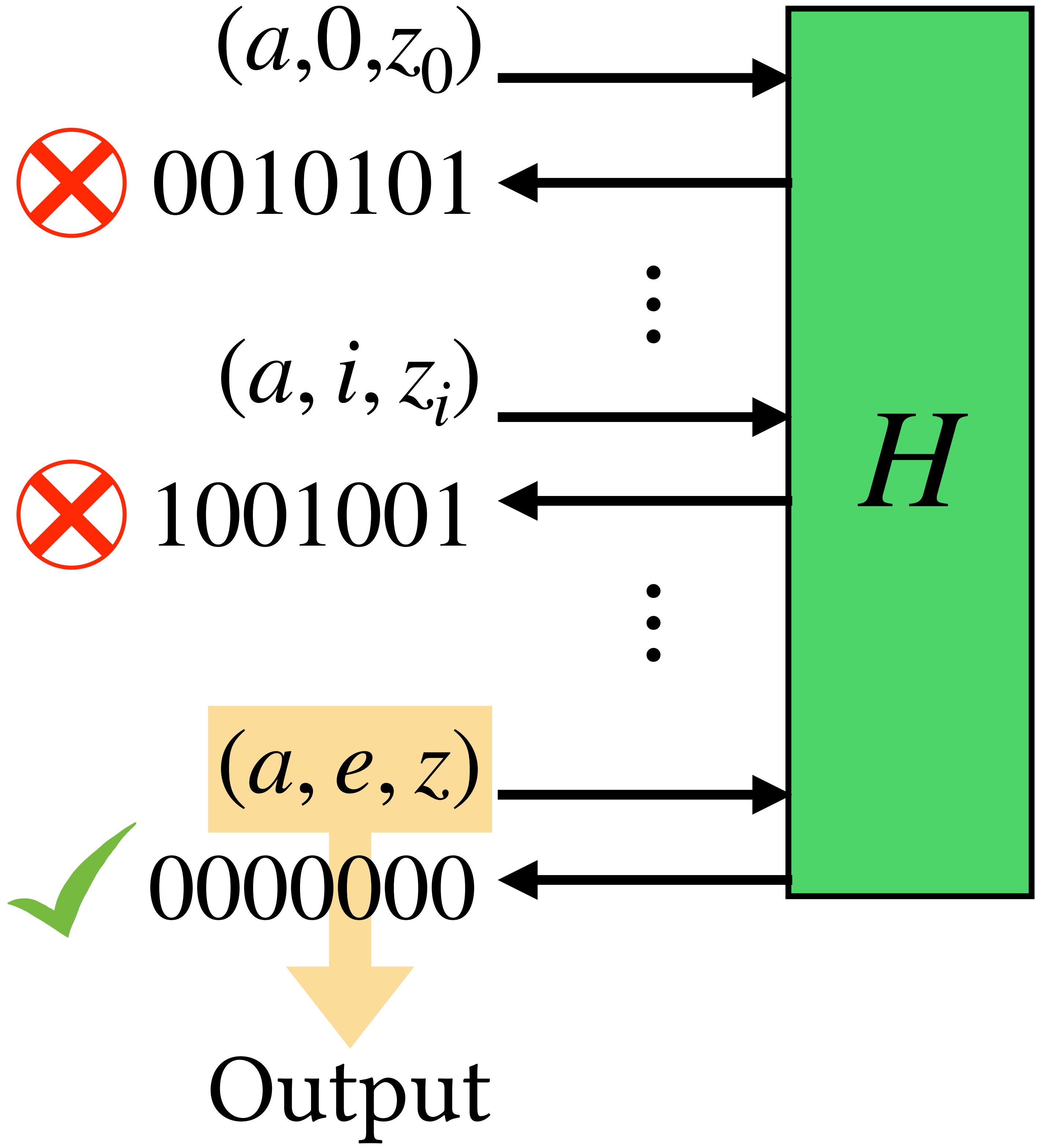
H

Fischlin's Transformation

- Let $\boxed{H}$: $\{0,1\}^* \mapsto \{0,1\}^\ell$ be a random oracle



Sample Σ-protocol first message 'a'



$(a,0,z_0)$

✗ 0010101

⋮

(a,i,z_i)

✗ 1001001

⋮

(a,e,z)

✓ 0000000

Output

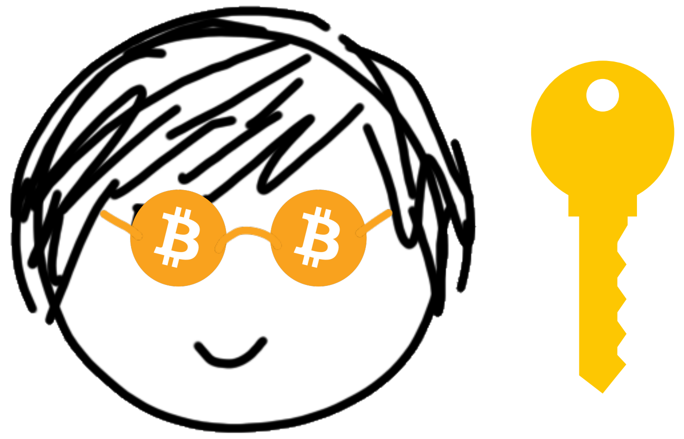
H

Soundness: Except with Pr=$2^{-\ell}$, P is forced to query more than one accepting transcript to $\boxed{H}$

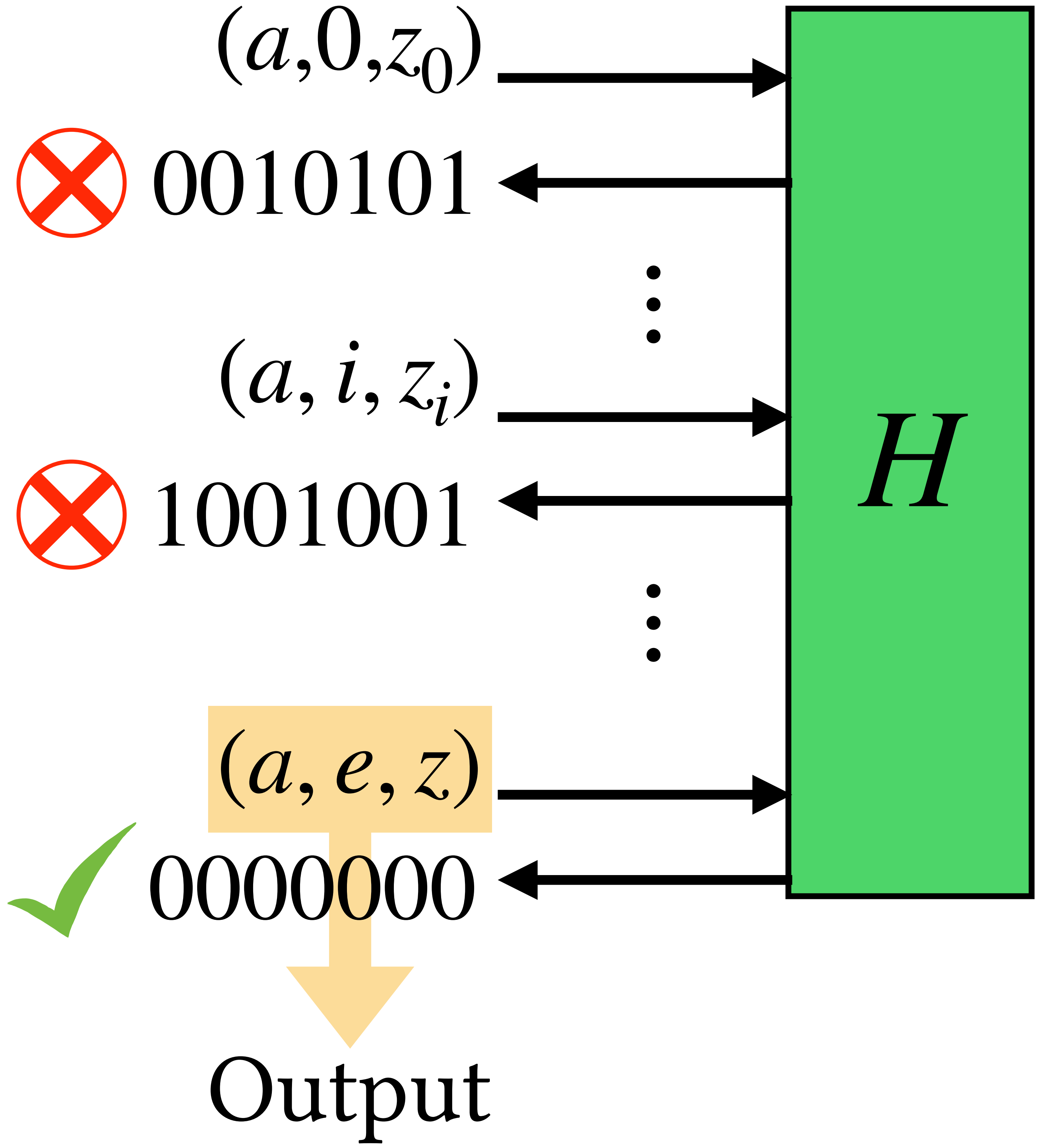
Completeness: P terminates in poly time when ℓ is small, i.e. $O(\log \kappa)$

Fischlin’s Transformation

- Let $\boxed{H}$: $\{0,1\}^* \mapsto \{0,1\}^{\ell}$ be a random oracle



Sample Σ-protocol first message ‘a’



$(a,0,z_0)$ ✗ 0010101

⋮

(a,i,z_i) ✗ 1001001

⋮

(a,e,z) ✓ 0000000

H

Output

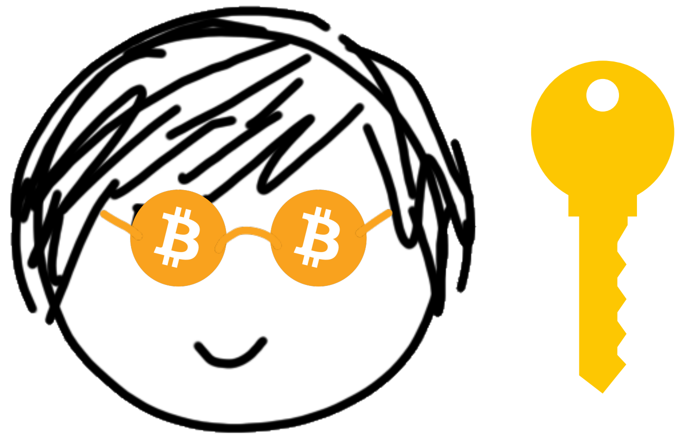
This gives Ext the values (e,z) and (e',z') as needed, by looking at queries made to H

Soundness: Except with Pr=$2^{-\ell}$, P is forced to query more than one accepting transcript to $\boxed{H}$

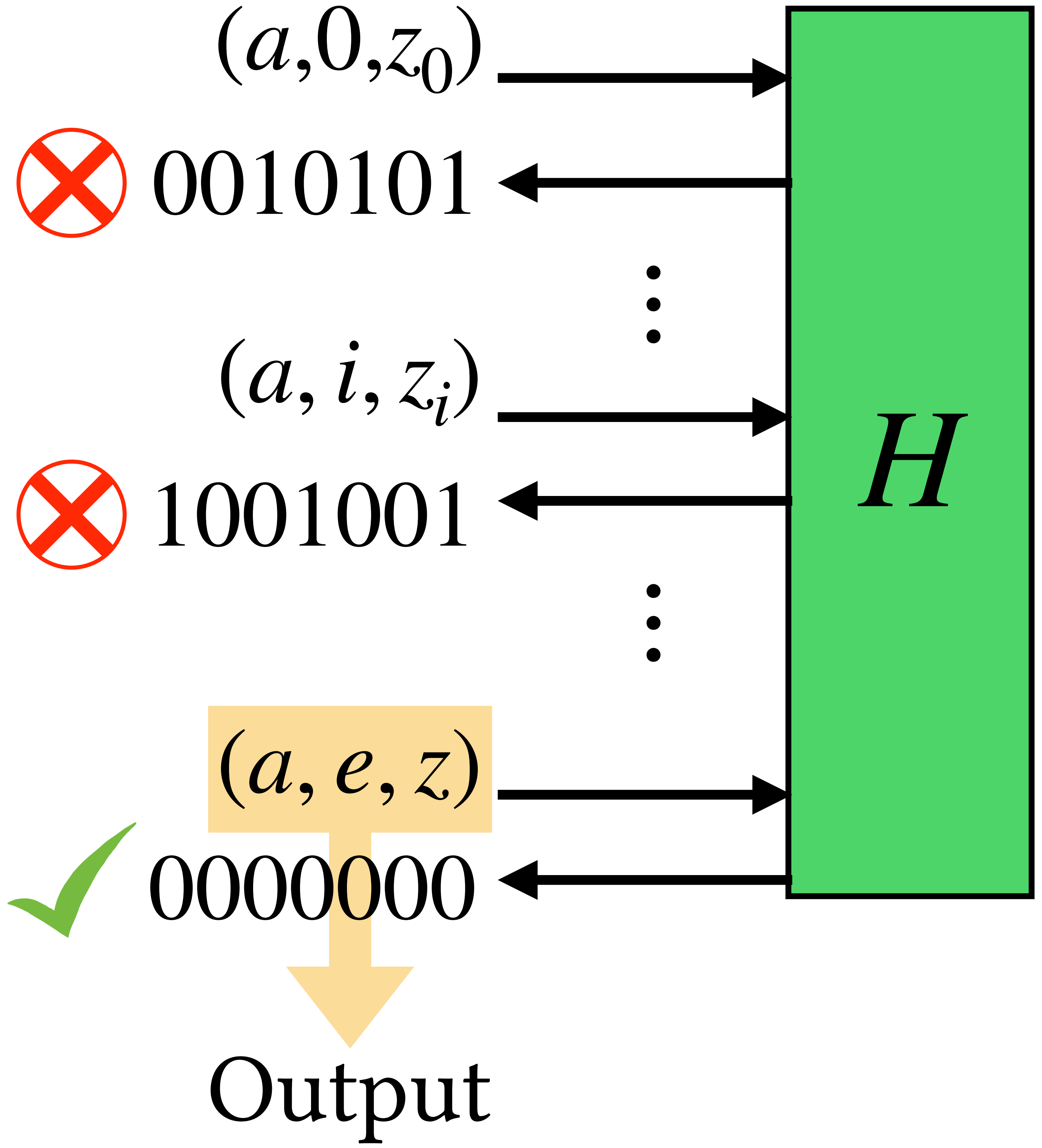
Completeness: P terminates in poly time when ℓ is small, i.e. $O(\log\kappa)$

Fischlin’s Transformation

- Let $\boxed{H}$: $\{0,1\}^* \mapsto \{0,1\}^\ell$ be a random oracle



Sample Σ-protocol first message ‘a’



This gives Ext the values (e, z) and (e', z') as needed, by looking at queries made to H

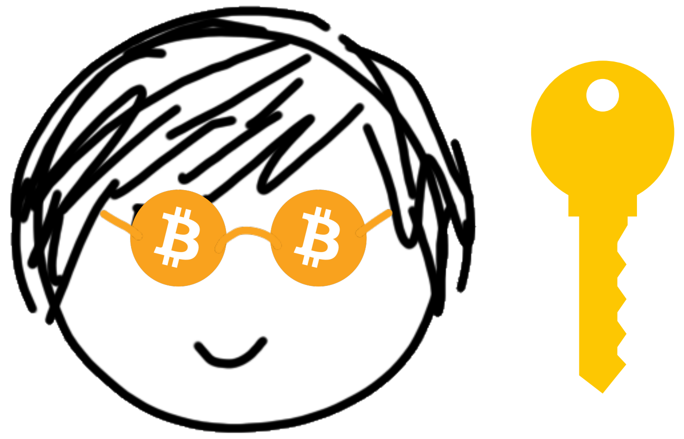
Soundness: Except with Pr=$2^{-\ell}$, P is forced to query more than one accepting transcript to $\boxed{H}$

Completeness: P terminates in poly time when ℓ is small, i.e. $O(\log \kappa)$

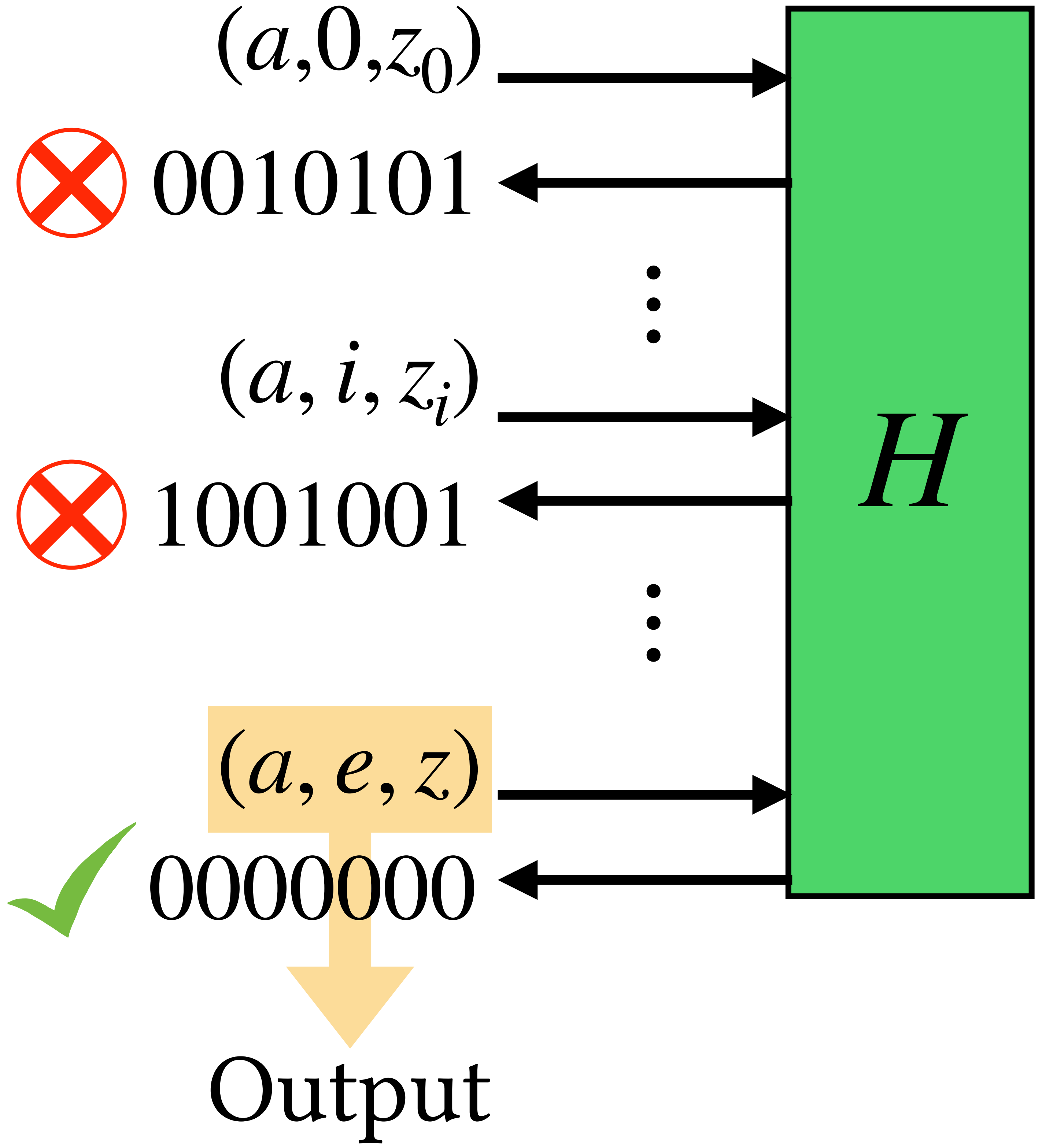
Problem!

Fischlin’s Transformation

- Let $\boxed{H}: \{0,1\}^* \mapsto \{0,1\}^\ell$ be a random oracle



Sample Σ-protocol first message ‘a’



$(a, 0, z_0)$

✗ 0010101

⋮

(a, i, z_i)

✗ 1001001

⋮

(a, e, z)

✓ 0000000

H

Output

This gives Ext the values (e, z) and (e', z') as needed, by looking at queries made to H

Soundness: Except with $\Pr = 2^{-\ell}$, P is forced to query more than one accepting transcript to $\boxed{H}$

Completeness: P terminates in poly time when ℓ is small, i.e. $O(\log \kappa)$

Problem!

Full Soundness: Repeat r times

Fischlin vs Pass: Qualitative

- Pass' compiler works for *any* Sigma protocol
- Fischlin's compiler works for a restricted class of Sigma protocols with 'quasi-unique responses'
- Supported by many standard Sigma protocols (eg. DLog), but many *may* not—especially if a statement can have multiple witnesses (eg. Pedersen Commitment opening, 1-of-2 witnesses, etc.)

Quasi-unique Responses [Fischlin 05]

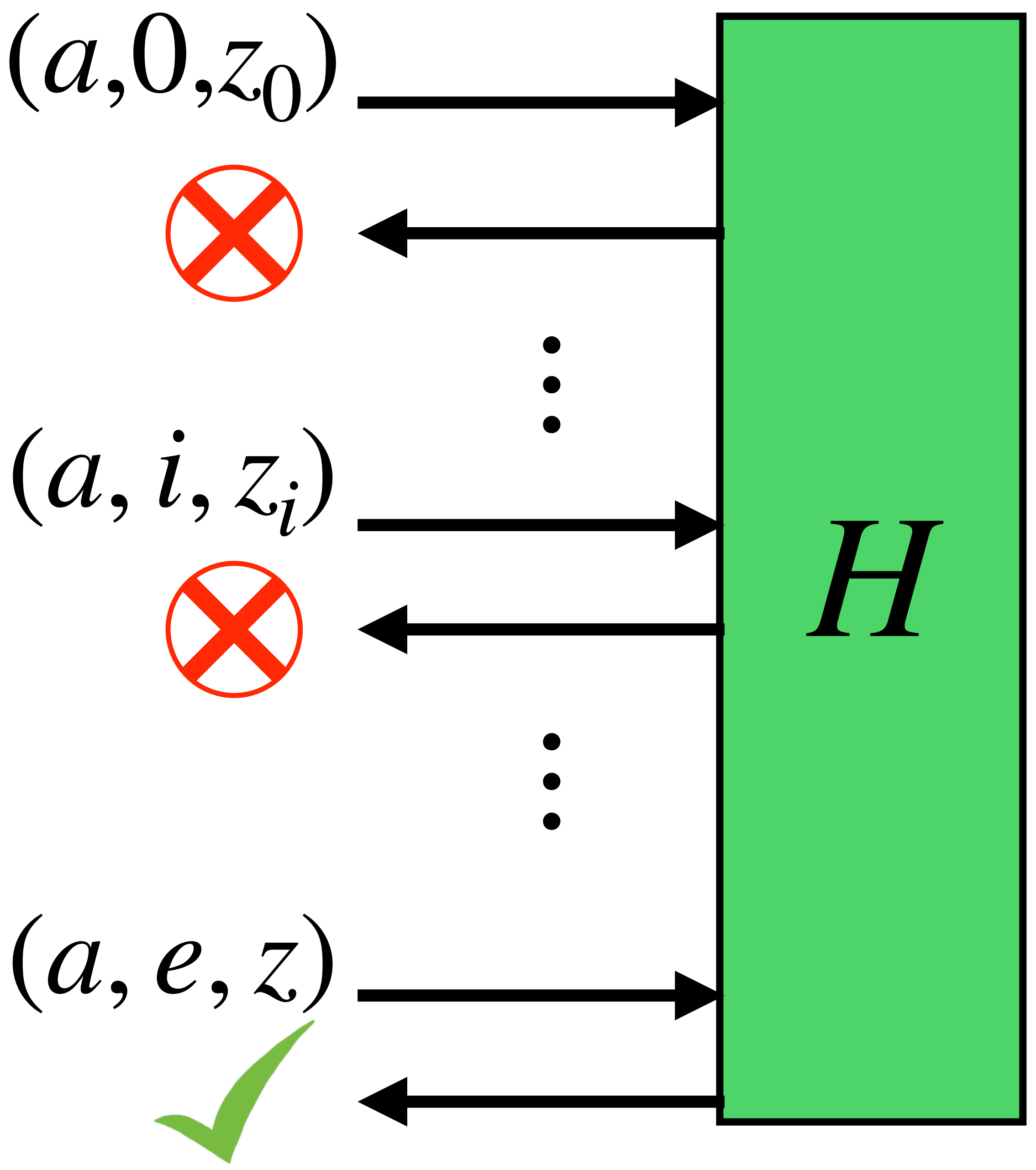
Hard: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ such that $V(a, e, z) = V(a, e, z') = 1$

Fixing (a, e) fixes z

Quasi-unique Responses [Fischlin 05]

Hard: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ such that $V(a, e, z) = V(a, e, z') = 1$

Fixing (a, e) fixes z

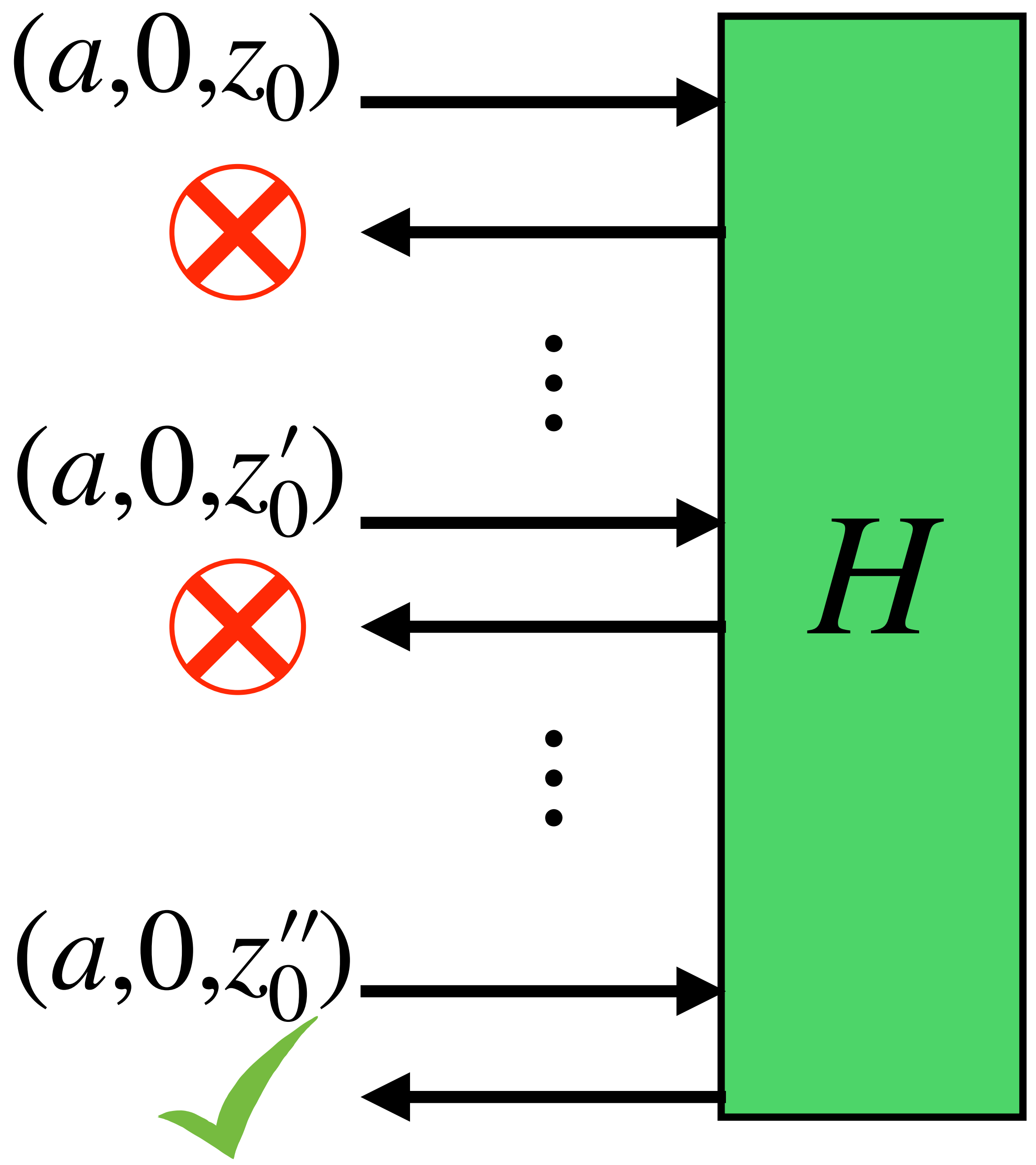


Quasi-unique Responses [Fischlin 05]

Hard: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ such that $V(a, e, z) = V(a, e, z') = 1$

Fixing (a, e) fixes z

Easy to see how this ties into soundness of Fischlin's compiler

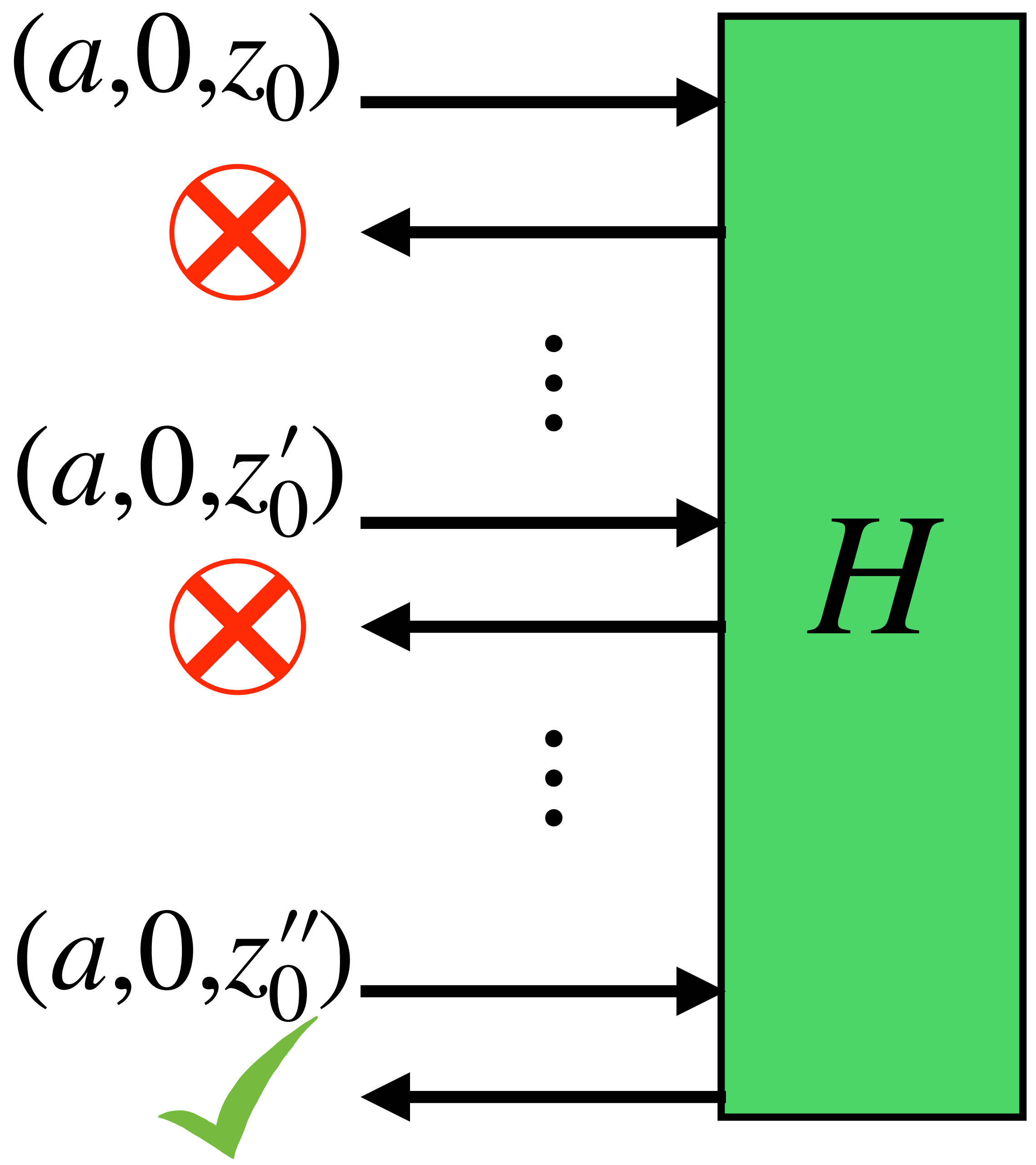


Quasi-unique Responses [Fischlin 05]

Hard: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ such that $V(a, e, z) = V(a, e, z') = 1$

Fixing (a, e) fixes z

Easy to see how this ties into soundness of Fischlin's compiler



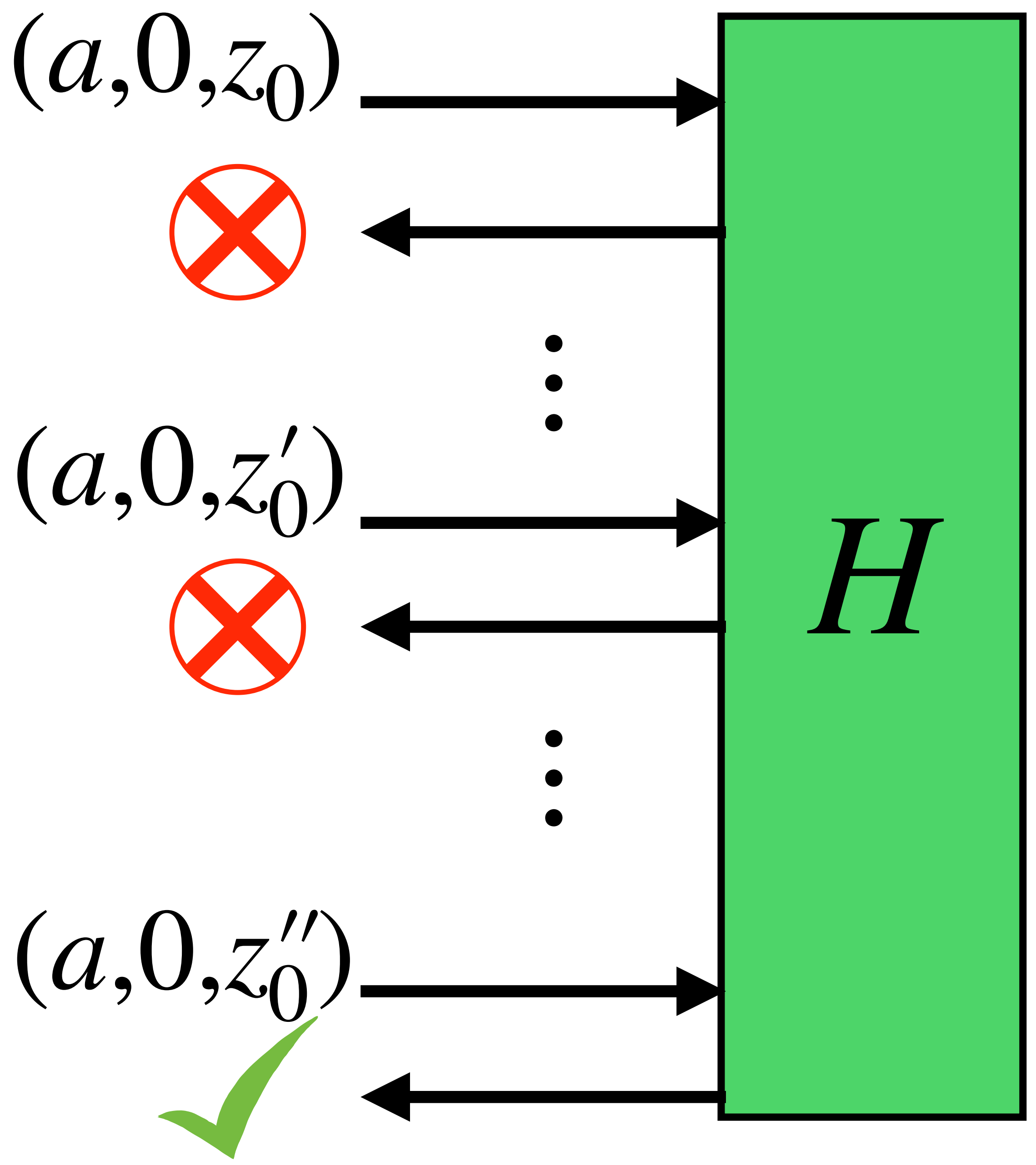
Prover can produce a proof without ever having to try more than one challenge

Quasi-unique Responses [Fischlin 05]

Hard: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ such that $V(a, e, z) = V(a, e, z') = 1$

Fixing (a, e) fixes z

Easy to see how this ties into soundness of Fischlin's compiler



Prover can produce a proof without ever having to try more than one challenge

Recall:
Extractor needs transcripts with different challenges

Is it *really* necessary, though?

- Folklore: breaking Sigma protocol abstraction, and simply ‘adjusting syntax’ of the extractor is usually sufficient to preserve Proof of Knowledge

- This is demonstrated by the Sigma protocol to prove knowledge of one-out-of-two witnesses [Cramer Damgård Schoenmakers 94]

- In [K shelat 22] we formalize this folklore

What about Zero-knowledge?

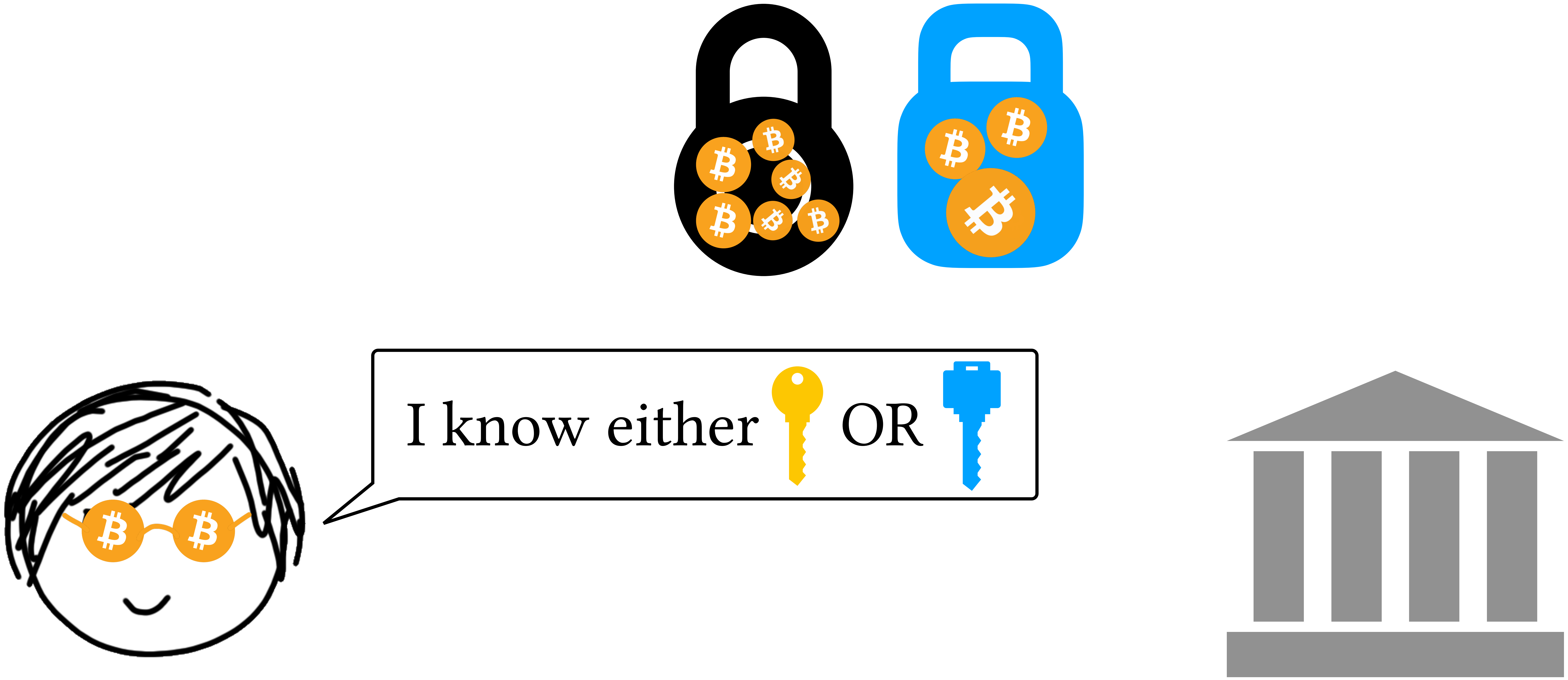
- Interestingly, Fischlin's proof of Zero-knowledge also depends on quasi-unique responses

- Unlike extraction, it is not intuitive as to why (or whether it's even necessary)

- [K shelat 22]: In the absence of unique responses, an explicit attack on *Witness Indistinguishability* (WI)

Witness Indistinguishability

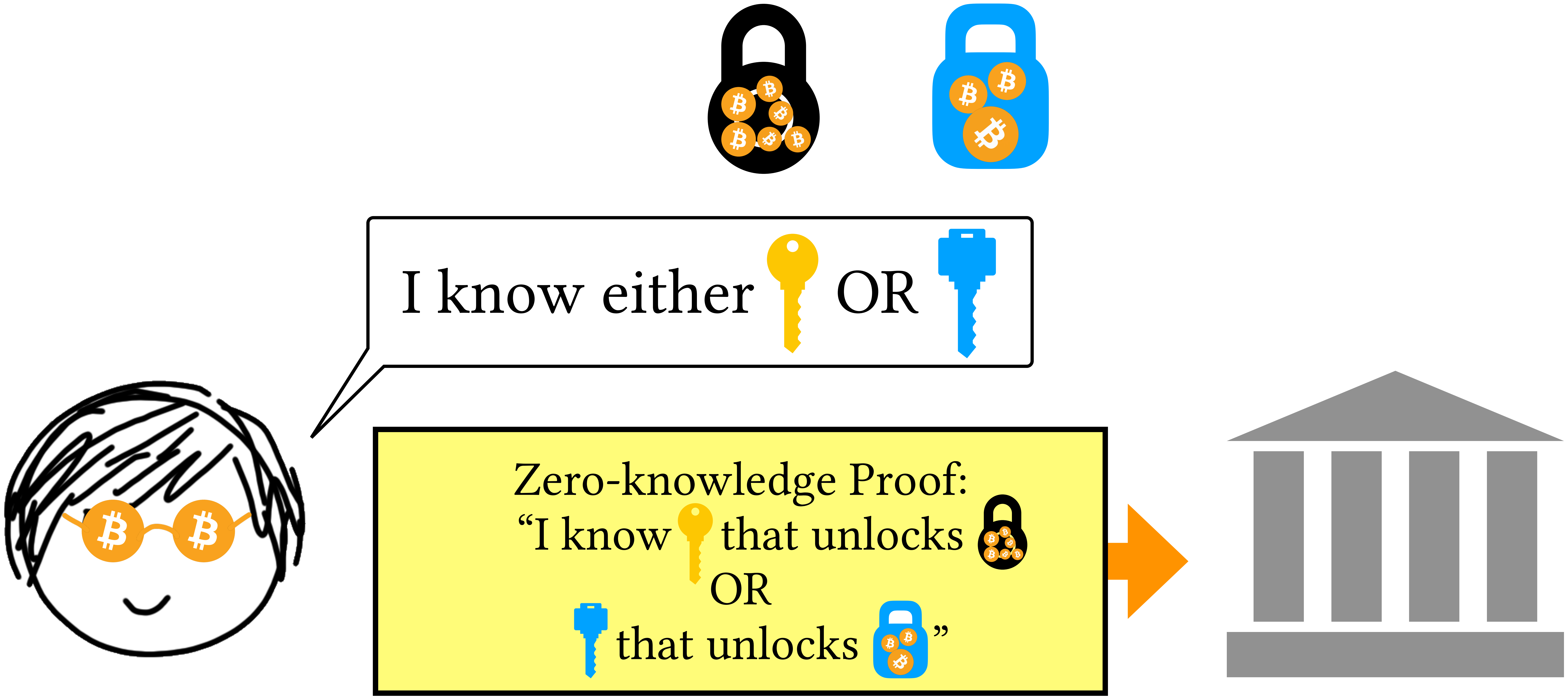
- The following kind of statement finds many applications:



I know either 🔑 OR 🔑

Witness Indistinguishability

- The following kind of statement finds many applications:

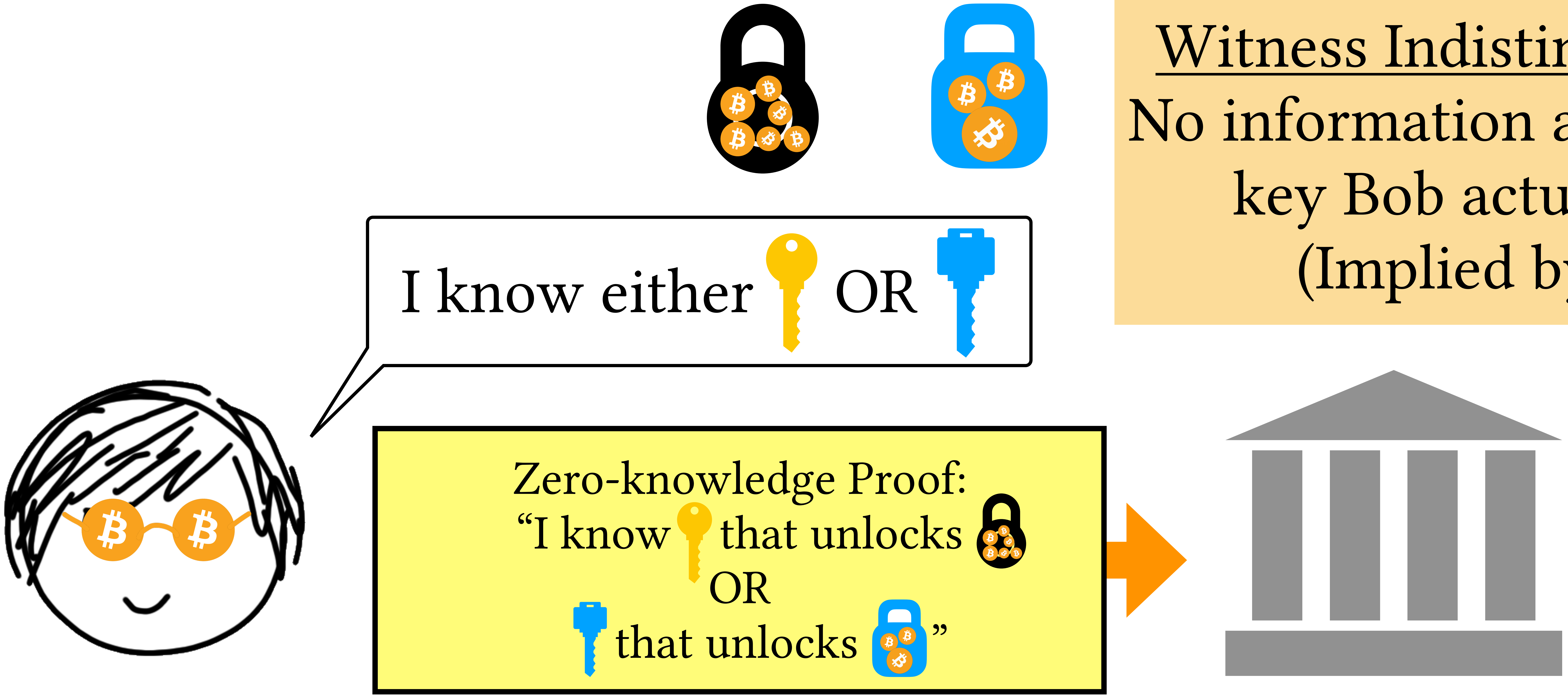


I know either 🔑 OR 🔑

Zero-knowledge Proof:
“I know 🔑 that unlocks 🔒
OR
🔑 that unlocks 🔒”

Witness Indistinguishability

- The following kind of statement finds many applications:



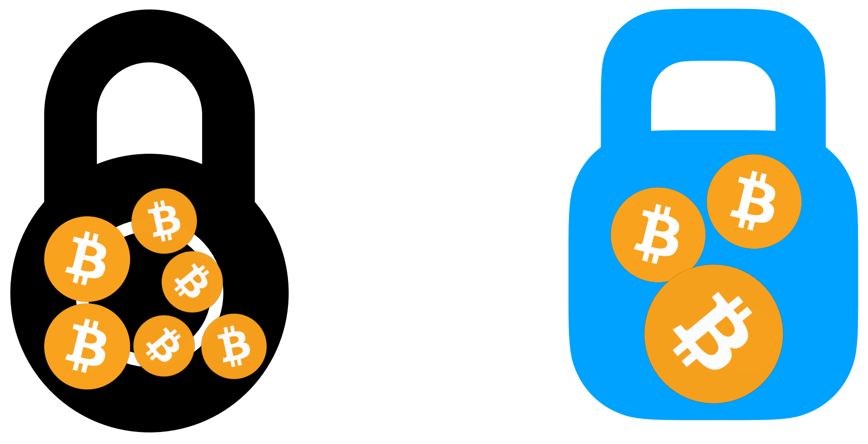
I know either OR

Zero-knowledge Proof:
"I know that unlocks
OR
that unlocks "

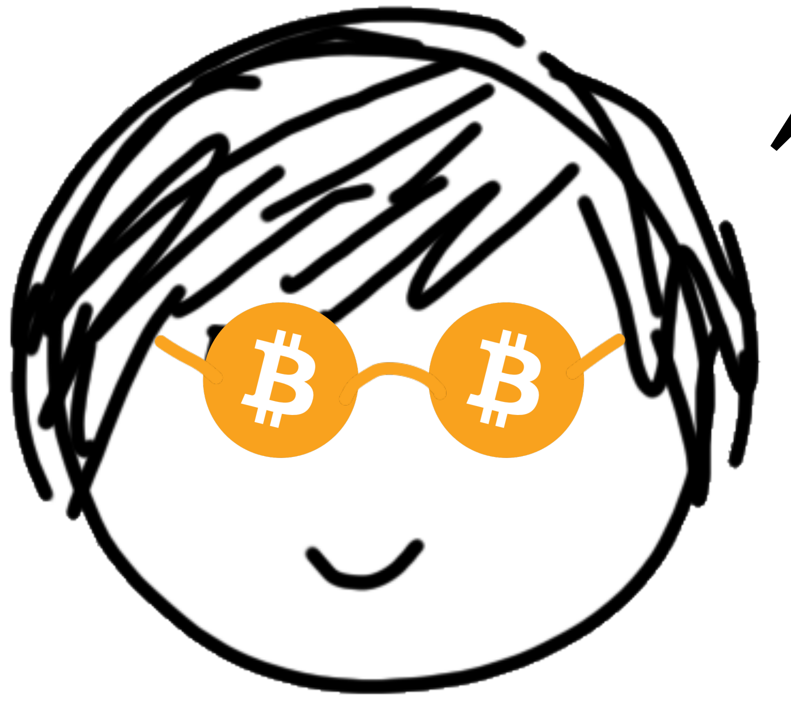
Witness Indistinguishable:
No information about which key Bob actually has
(Implied by ZK)

Witness Indistinguishability

- The following kind of statement finds many applications:



I know either 🔑 OR 🔑



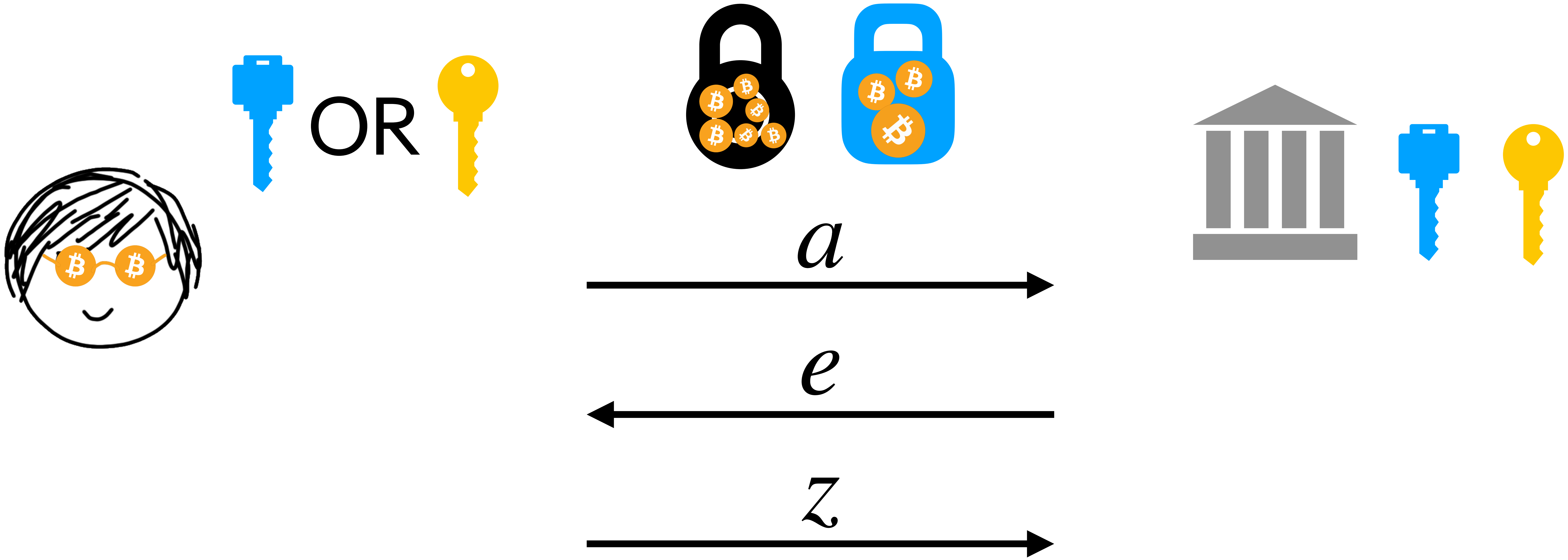
Zero-knowledge Proof:
"I know 🔑 that unlocks 🔒
OR
🔑 that unlocks 🔒"

<u>Witness Indistinguishable:</u>
No information about which key Bob actually has
(Implied by ZK)

<u>Important note:</u>
This holds even if both keys are actually known to bank
(like known plaintext security)

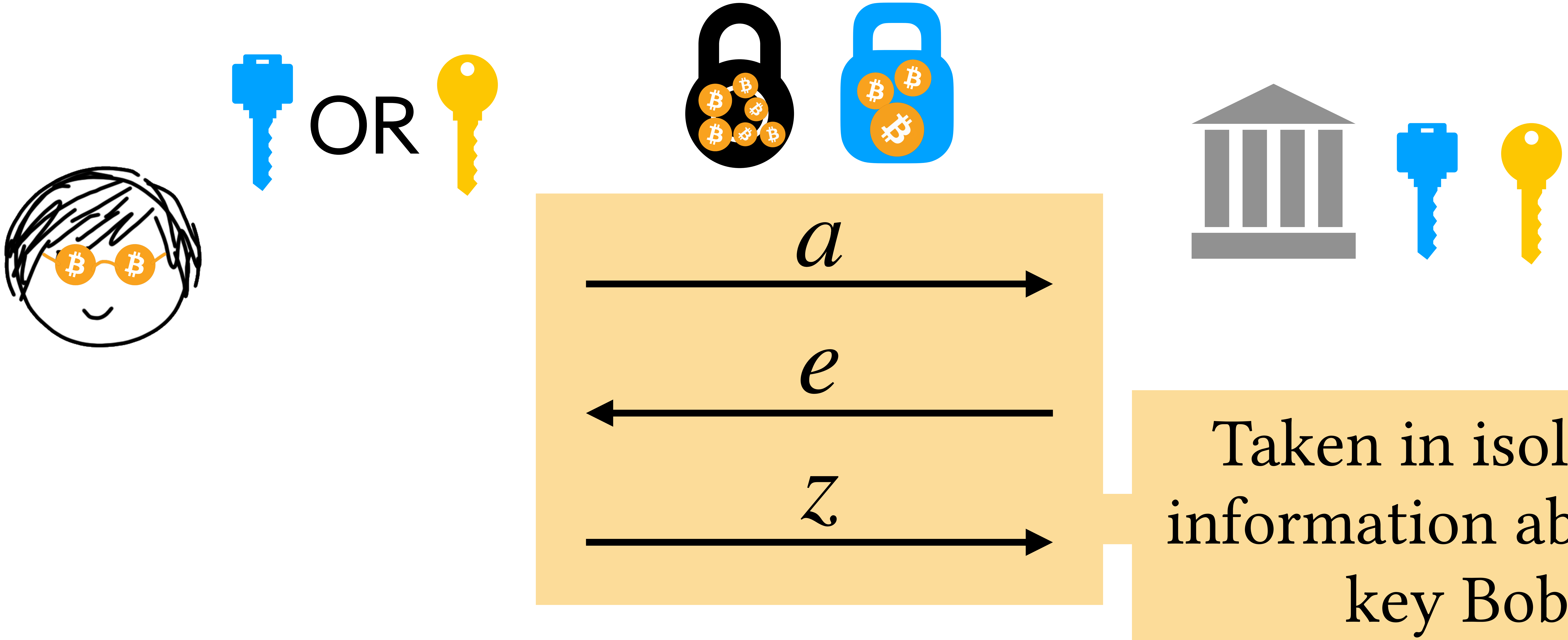
Useful Fact

- Some Σ protocols have the following property: (including some multi-witness ones)



Useful Fact

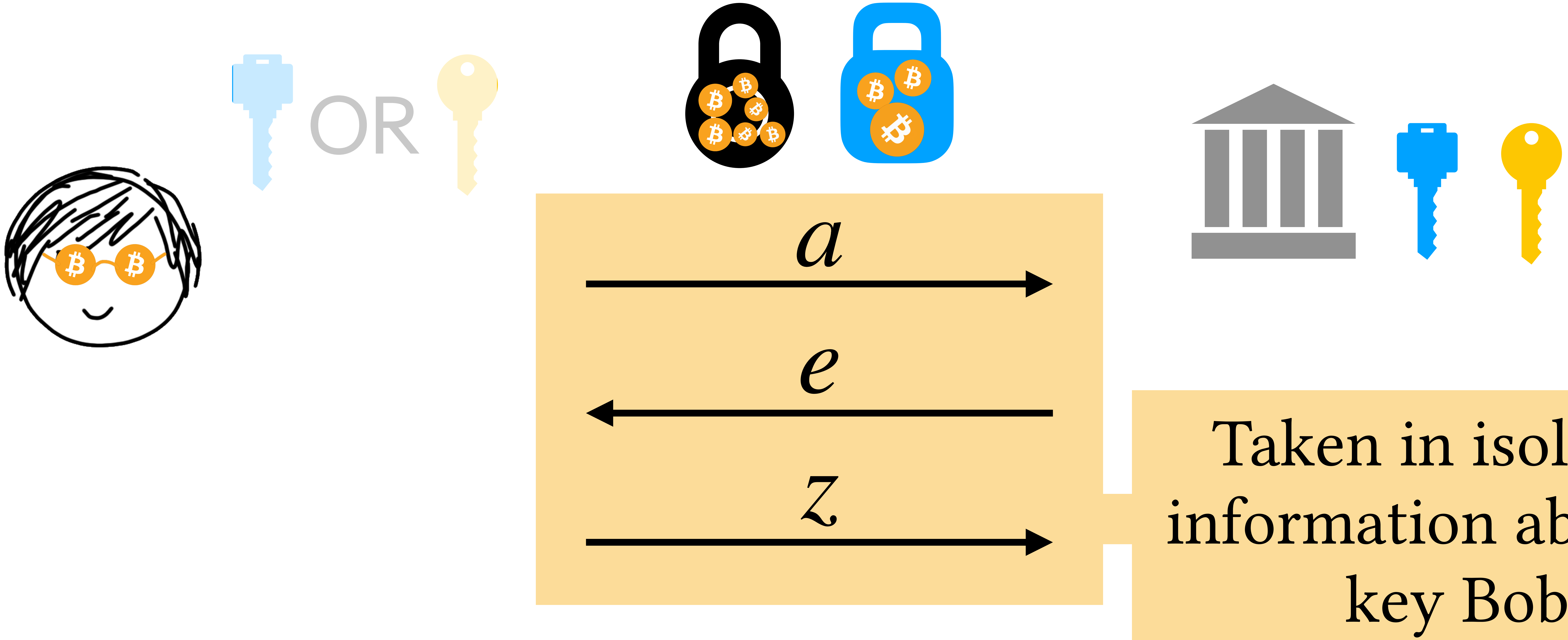
- Some Σ protocols have the following property: (including some multi-witness ones)



Taken in isolation, no information about which key Bob has

Useful Fact

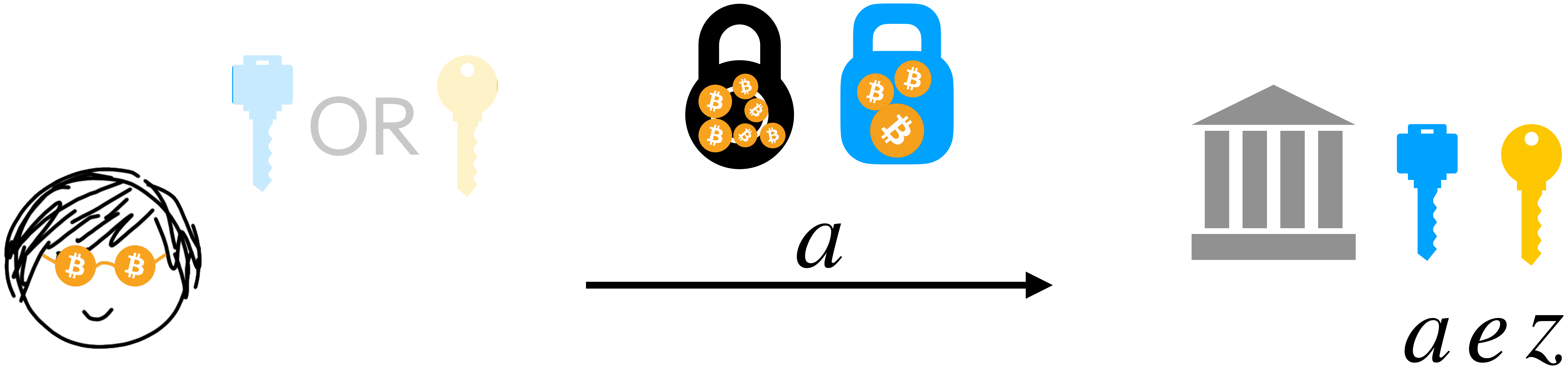
- Some Σ protocols have the following property: (including some multi-witness ones)



Taken in isolation, no information about which key Bob has

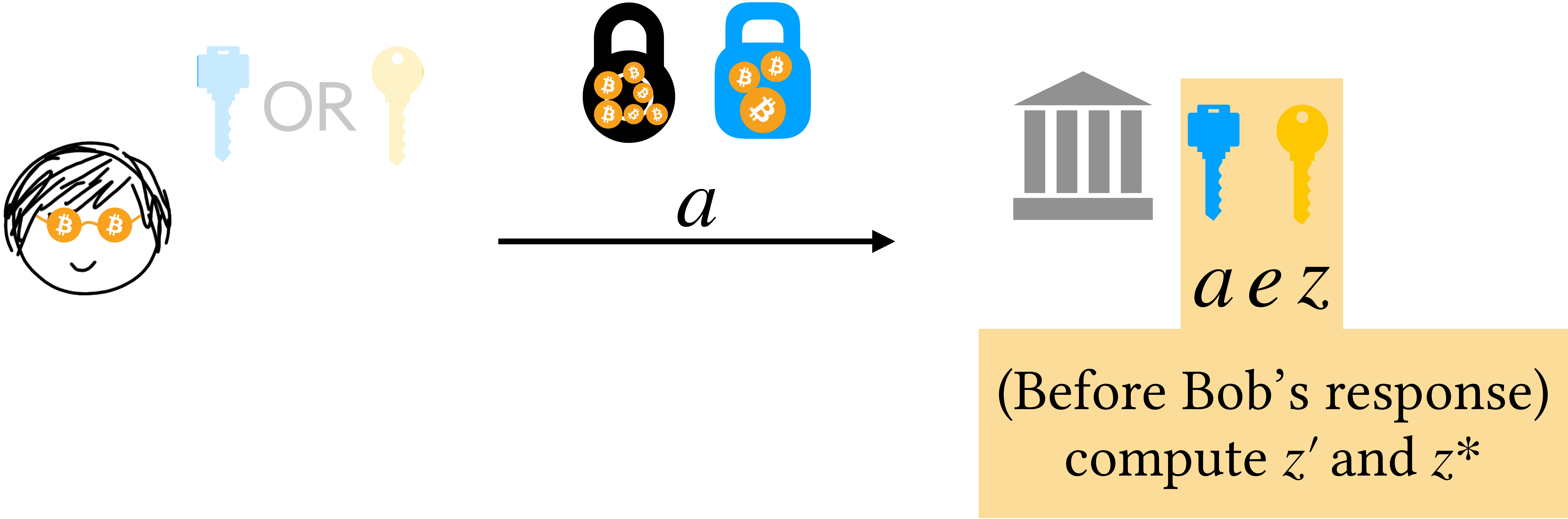
Useful Fact

- Some Σ protocols have the following property:
  (including some multi-witness ones)



Useful Fact

- Some Σ protocols have the following property: (including some multi-witness ones)



OR

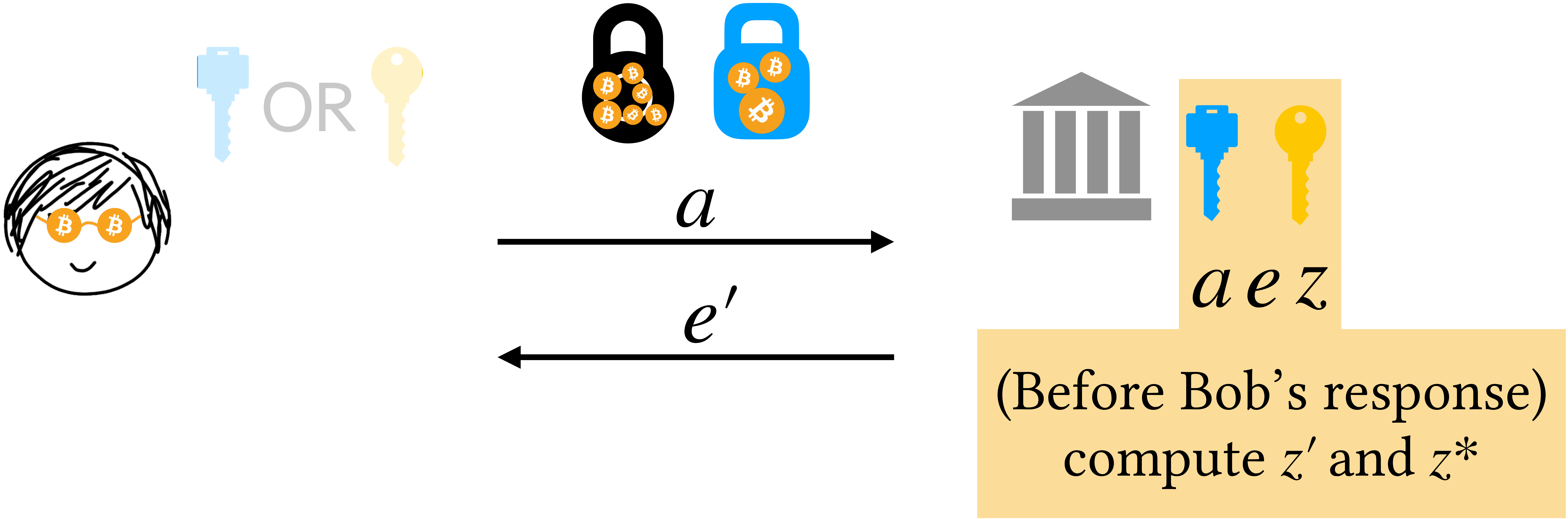
a

$a\ e\ z$

(Before Bob's response) compute z' and z^*

Useful Fact

- Some Σ protocols have the following property: (including some multi-witness ones)



OR

a

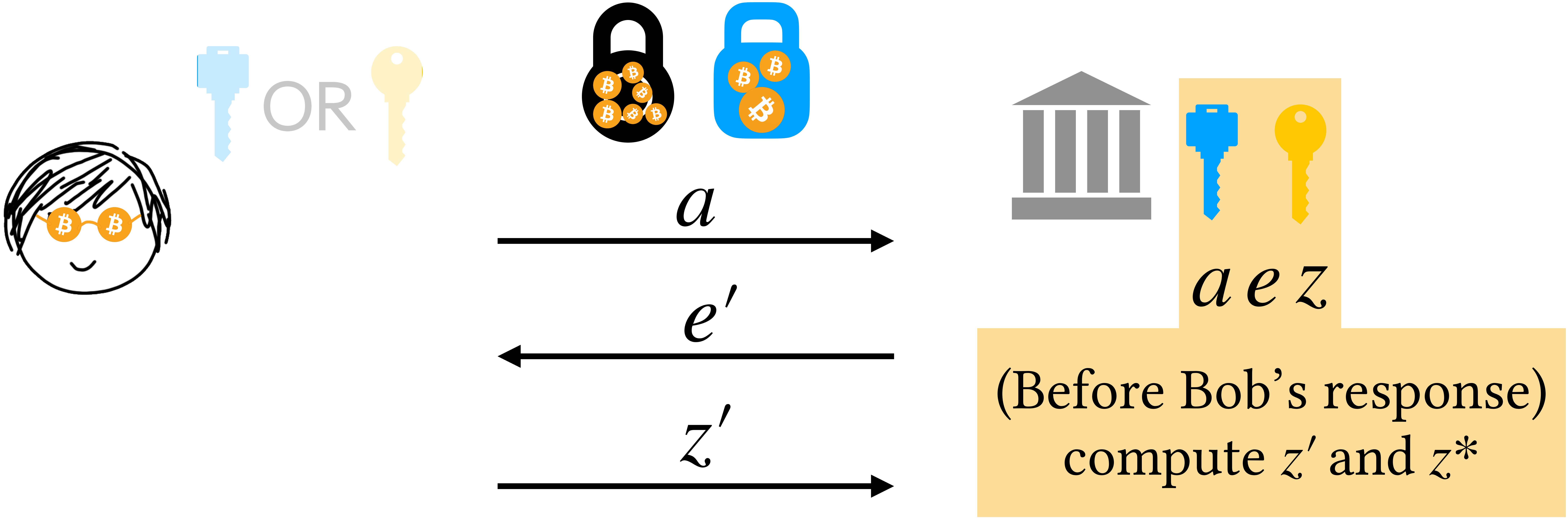
e'

$a \; e \; z$

(Before Bob's response) compute z' and z^*

Useful Fact

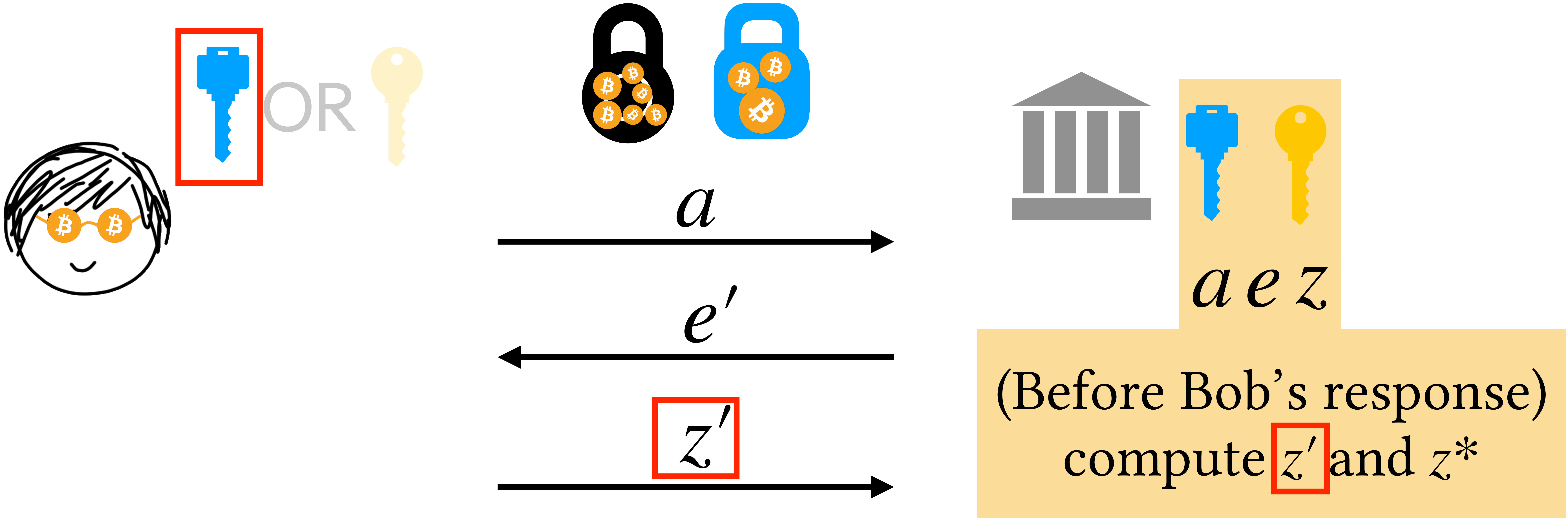
- Some Σ protocols have the following property: (including some multi-witness ones)



(Before Bob's response) compute z' and z^*

Useful Fact

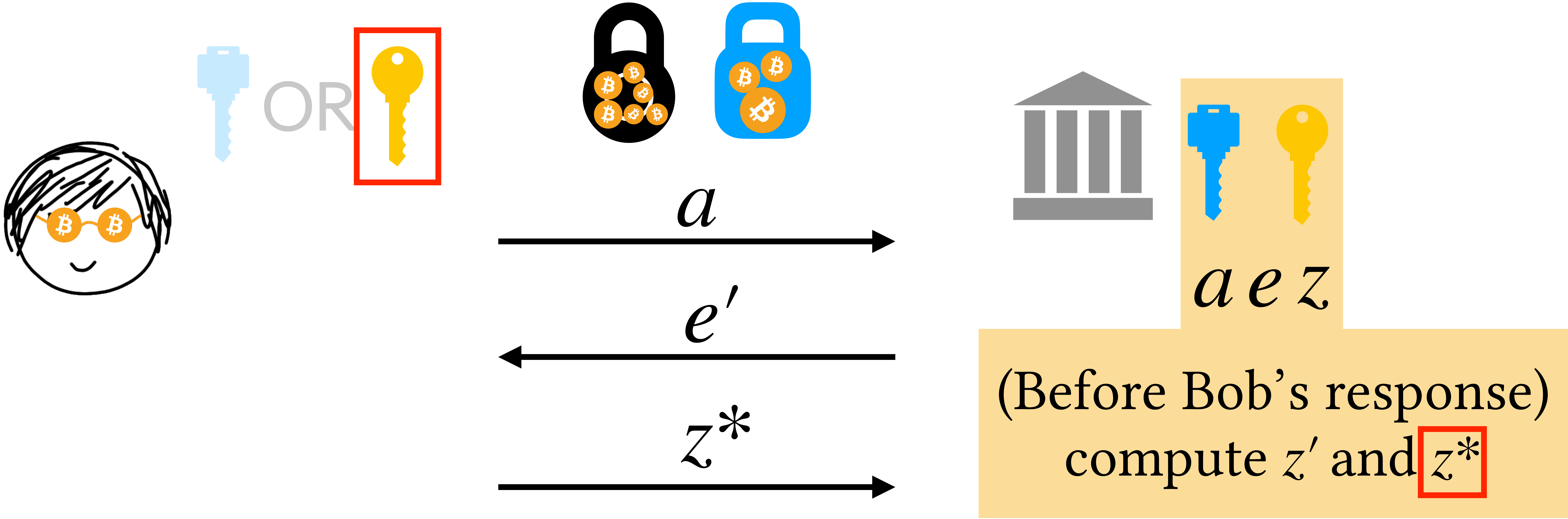
- Some Σ protocols have the following property: (including some multi-witness ones)



(Before Bob's response) compute $\boxed{z'}$ and z^*

Useful Fact

- Some Σ protocols have the following property: (including some multi-witness ones)



a

e'

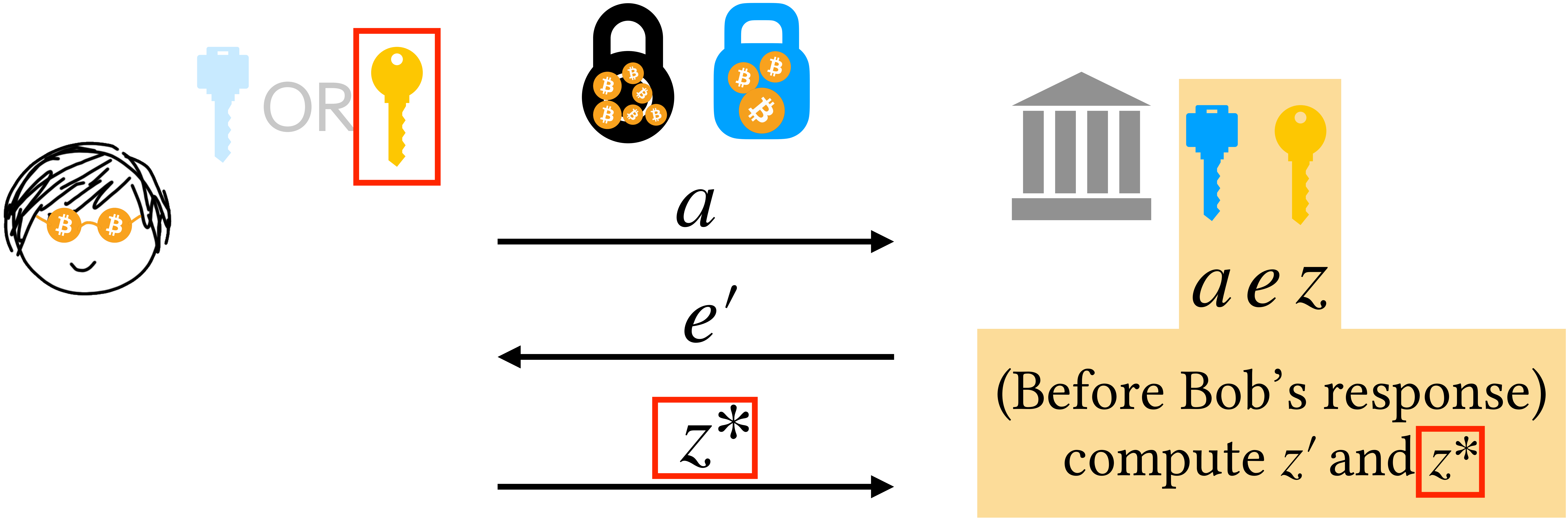
z^*

$a\ e\ z$

(Before Bob's response) compute z' and $\boxed{z^*}$

Useful Fact

- Some Σ protocols have the following property: (including some multi-witness ones)



a

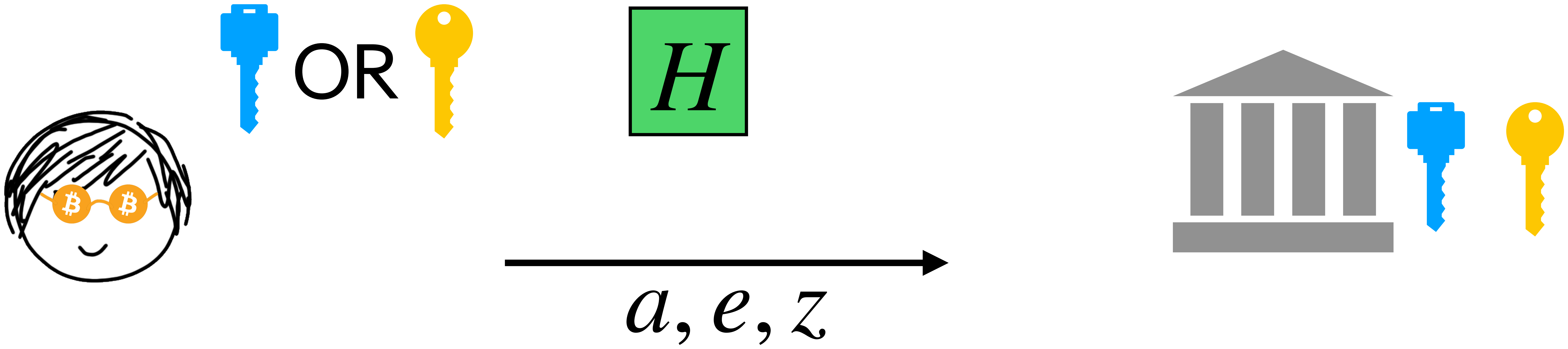
e'

$\boxed{z^*}$

$a \; e \; z$

(Before Bob's response) compute z' and $\boxed{z^*}$

Attack Strategy



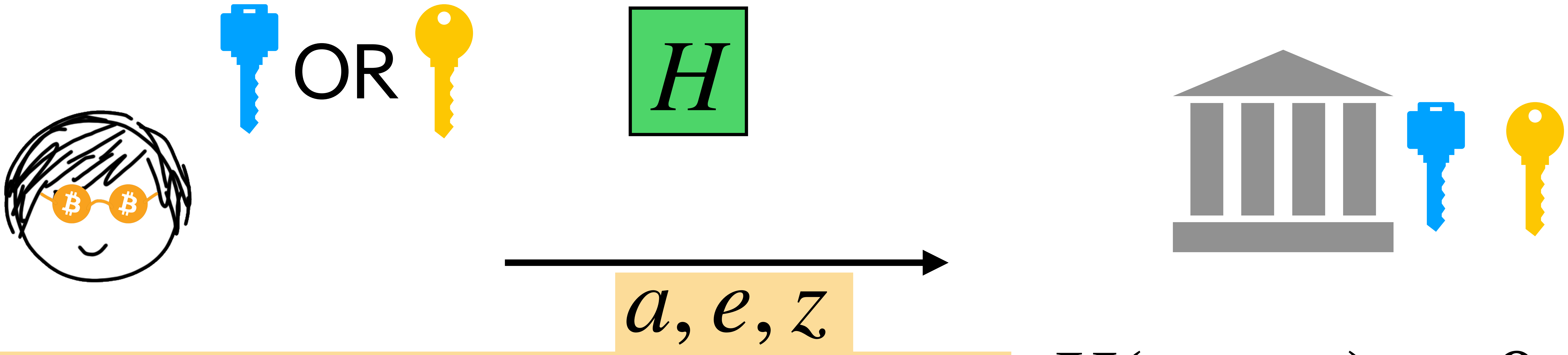
OR

H

a, e, z

$$H(a, e, z) = 0$$

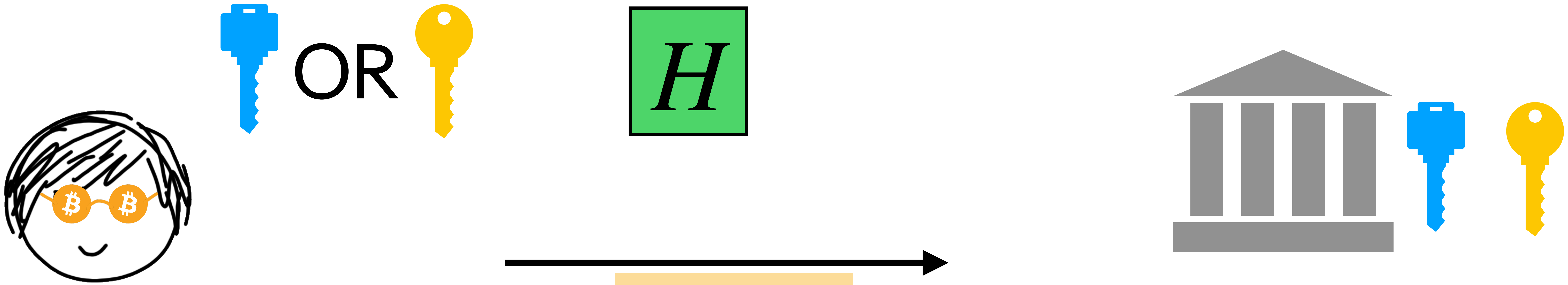
Attack Strategy



Reveals nothing about Bob's key in isolation

$$H(a, e, z) = 0$$

Attack Strategy



a, e, z

Reveals nothing about Bob’s key in isolation

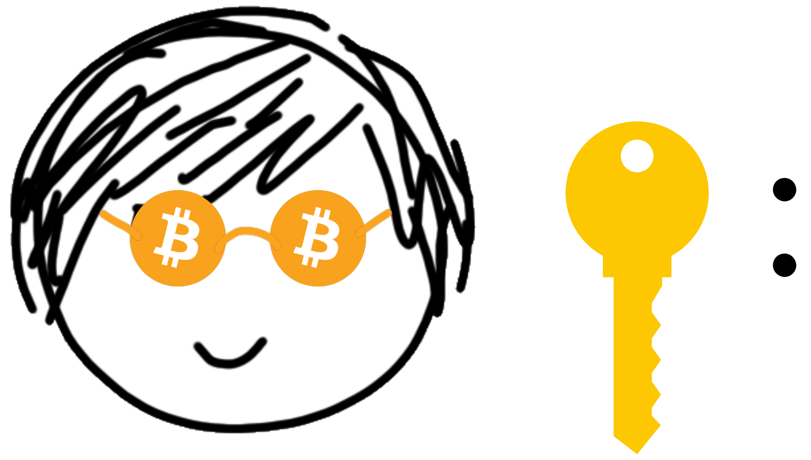
$H(a, e, z) = 0$

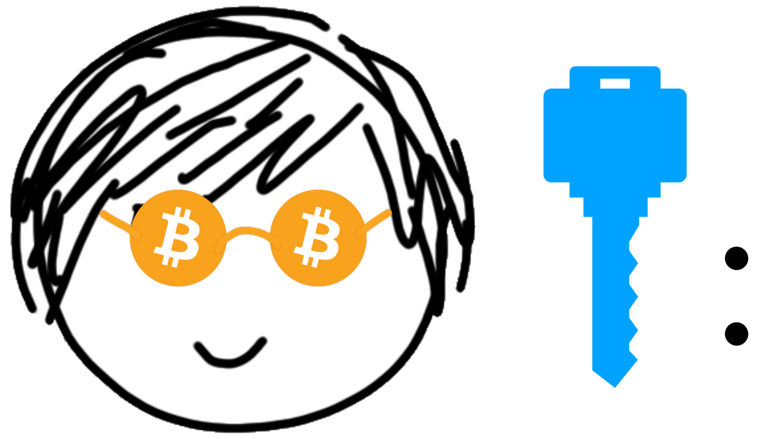
- Imagine we could ask Bob to answer challenge e' …his answer (z' or z^*) would determine which key he has

- Turns out we can achieve this effect by probing H (with no special privileges)

Probing Strategy

Common a

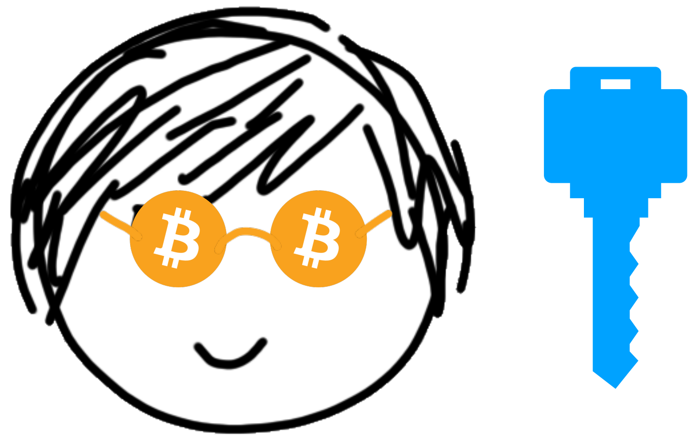




(e, z)

If both possibilities "agree" at e, then they "disagree" at any $e' \neq e$

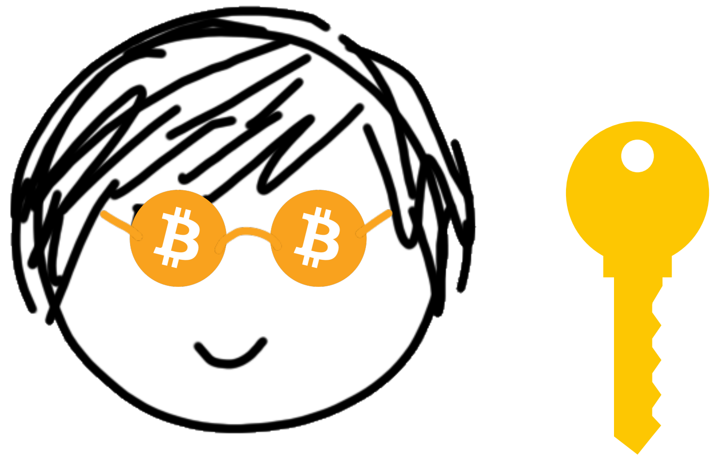
Probing Strategy



:

Common a

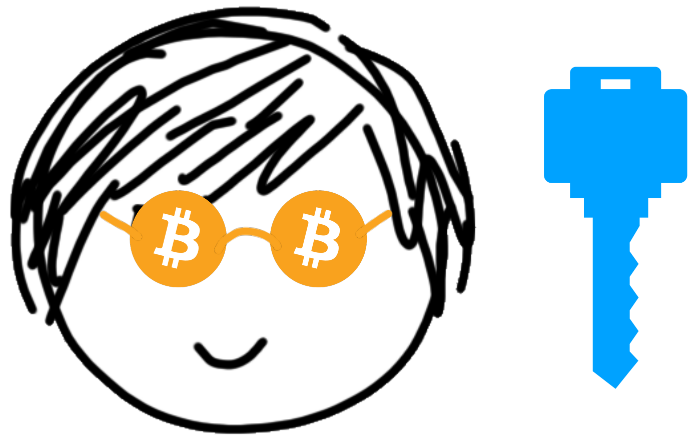
If both possibilities “agree” at e, then they “disagree” at any $e' \neq e$



:

(e, z)

Probing Strategy

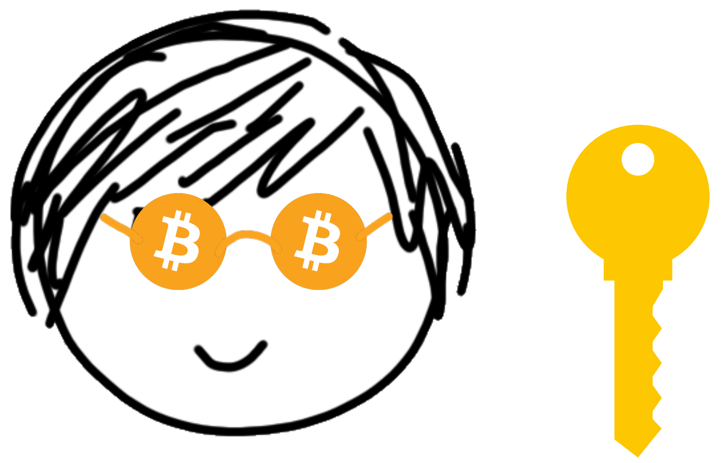


:

$(0,\ z_0')$

Common a

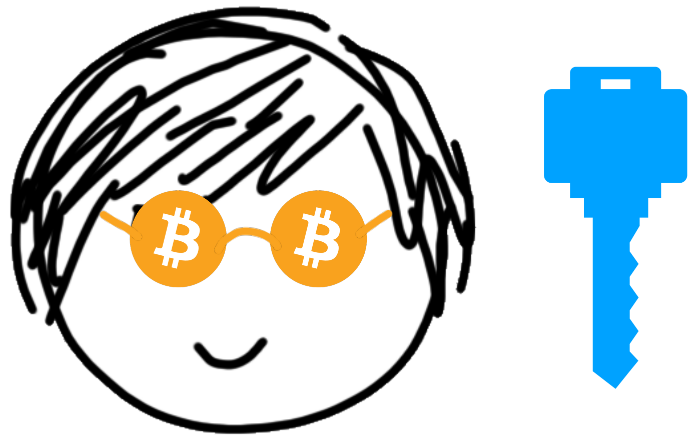
If both possibilities "agree" at e, then they "disagree" at any $e' \neq e$



: $(0,\ z_0^*)$

$(e\,,z)$

Probing Strategy



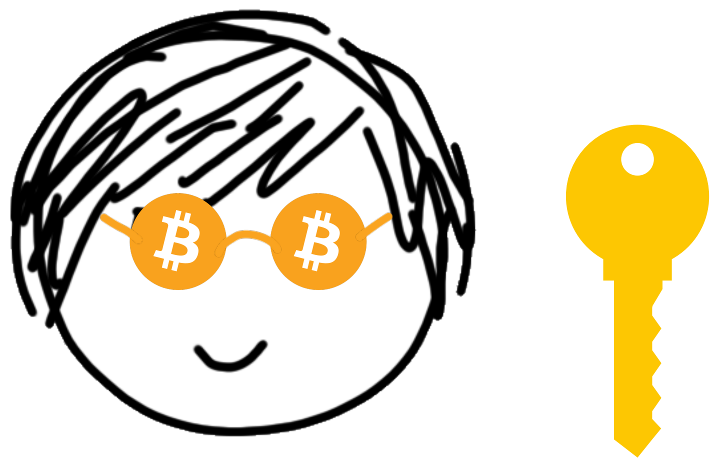
:

$(0,\ z'_0)$

$(1,\ z'_1)$

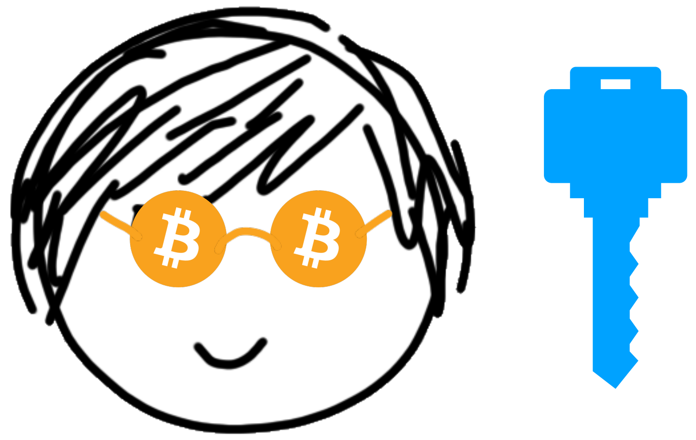
Common a

If both possibilities "agree" at e, then they "disagree" at any $e' \neq e$



: $(0,\ z_0^*)$ $(1,\ z_1^*)$ $(e\,,z)$

Probing Strategy



:

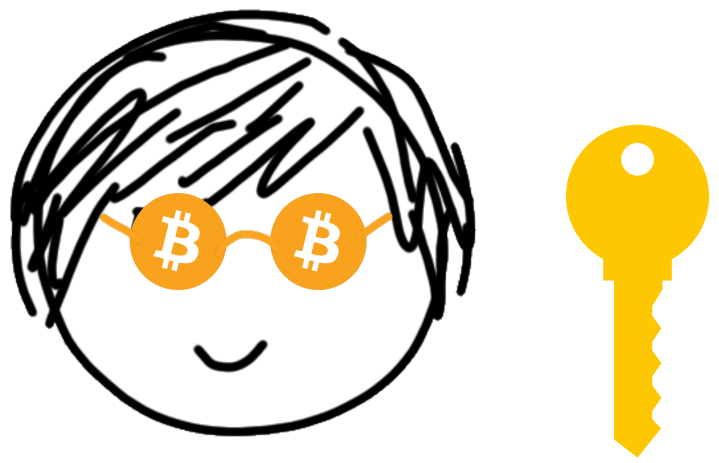
$(0,\ z'_0)$

$(1,\ z'_1)$

$\vdots$

$\boxed{\text{Common } a}$

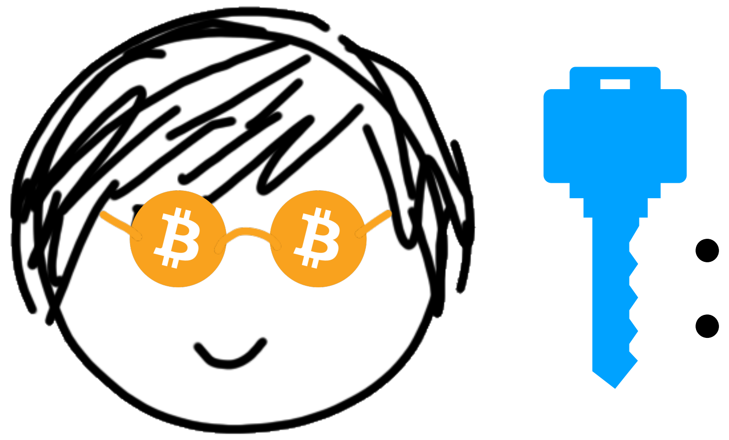
If both possibilities "agree" at e, then they "disagree" at any $e' \neq e$



: $(0,\ z^*_0)$ $(1,\ z^*_1)$ $\cdots$ $(e\ ,z\ )$

Probing Strategy

Given (a, e, z) produced by Fischlin's compiler, we can test which path is "plausible"

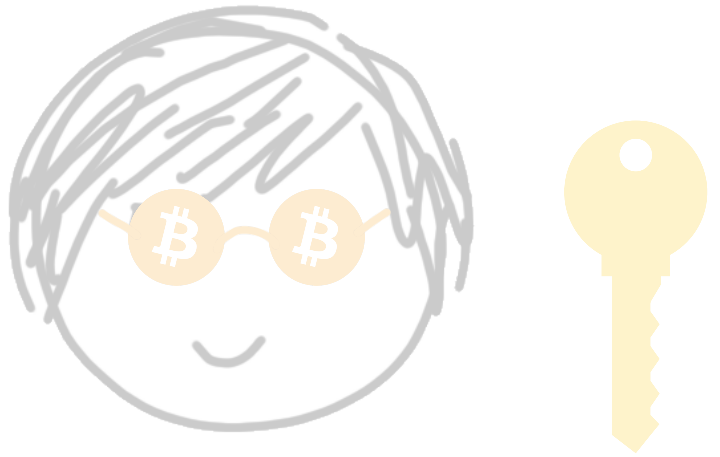


$(0,\ z'_0)$

$(1,\ z'_1)$

$\vdots$

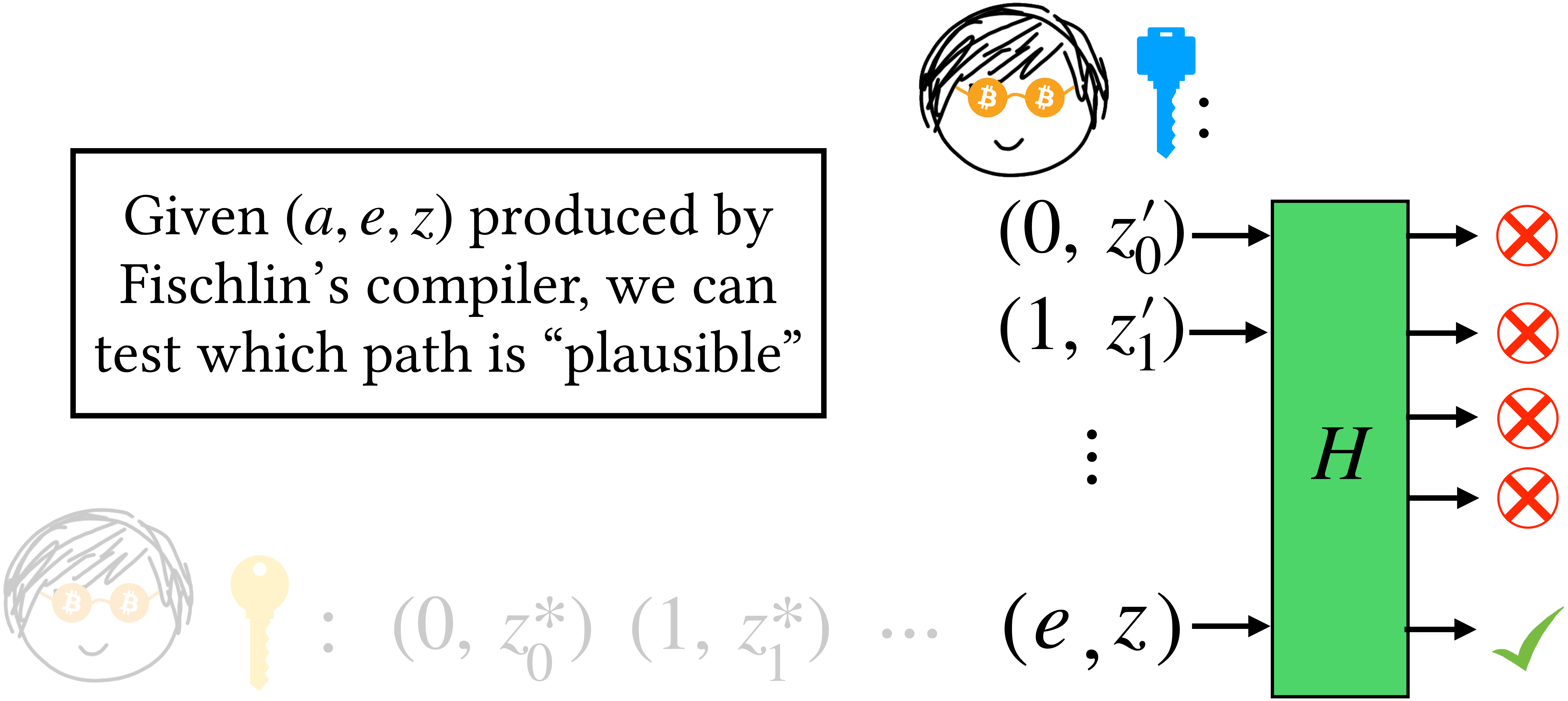
H



$: \quad (0,\ z^*_0)\ (1,\ z^*_1)\ \cdots\ (e, z)$

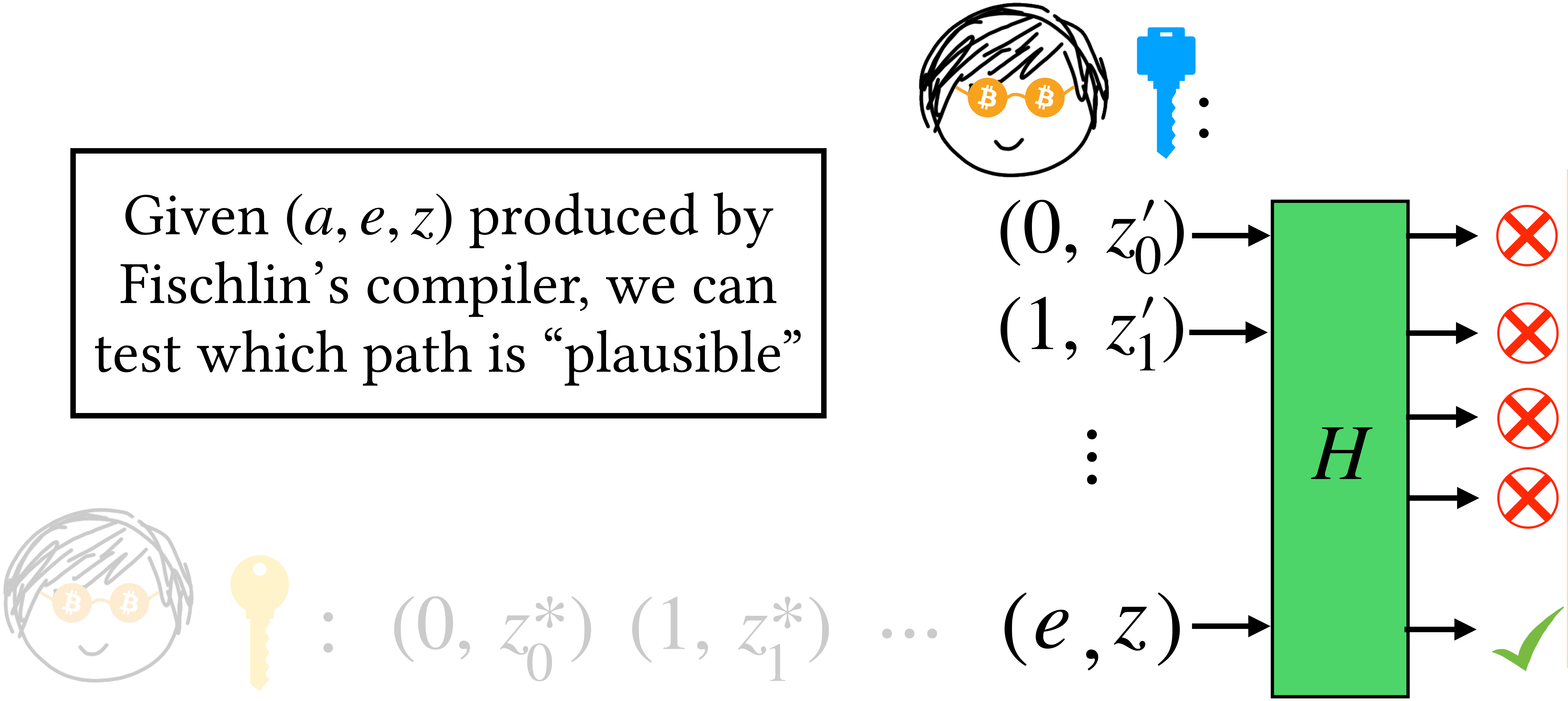
Probing Strategy

Given (a, e, z) produced by Fischlin's compiler, we can test which path is "plausible"



Probing Strategy

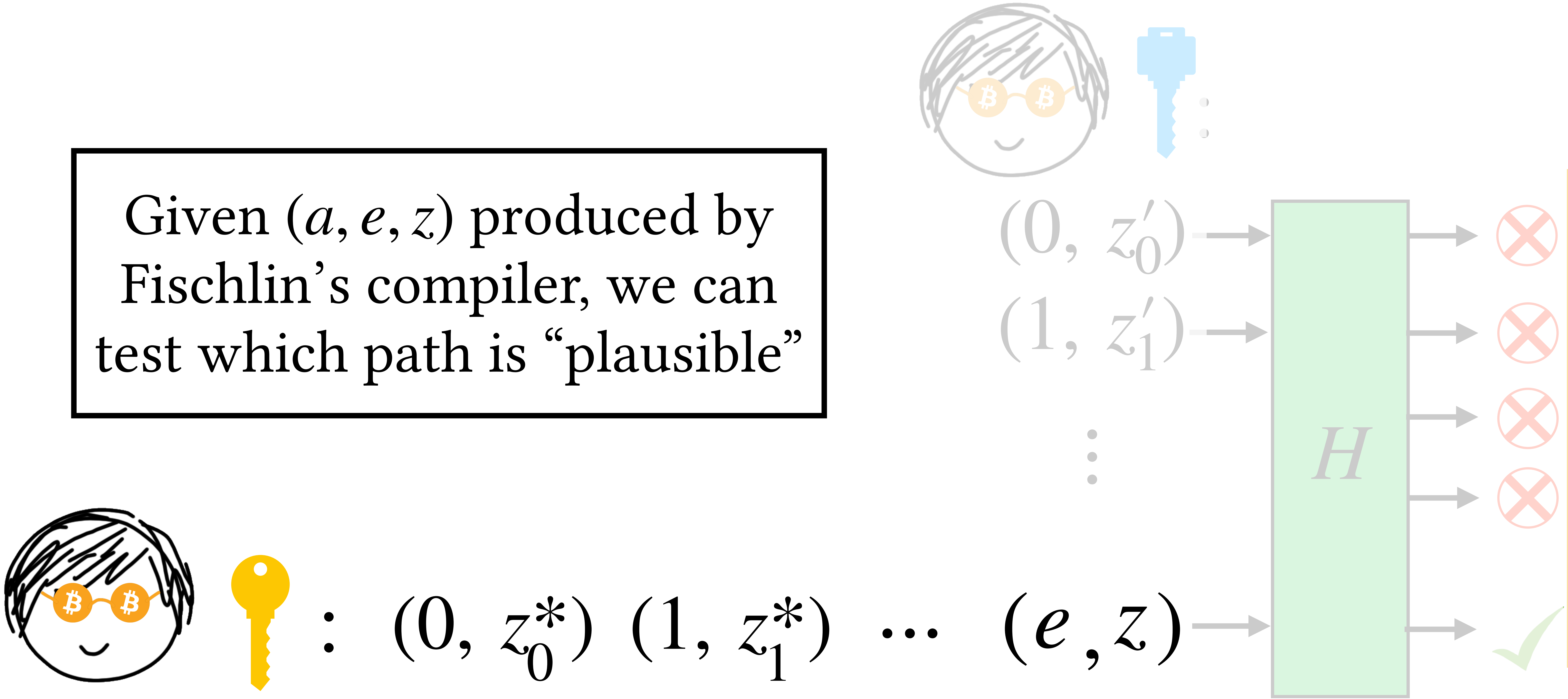
Given (a, e, z) produced by Fischlin's compiler, we can test which path is "plausible"



W.h.p., only one path—induced by one of the two keys—is plausible

Probing Strategy

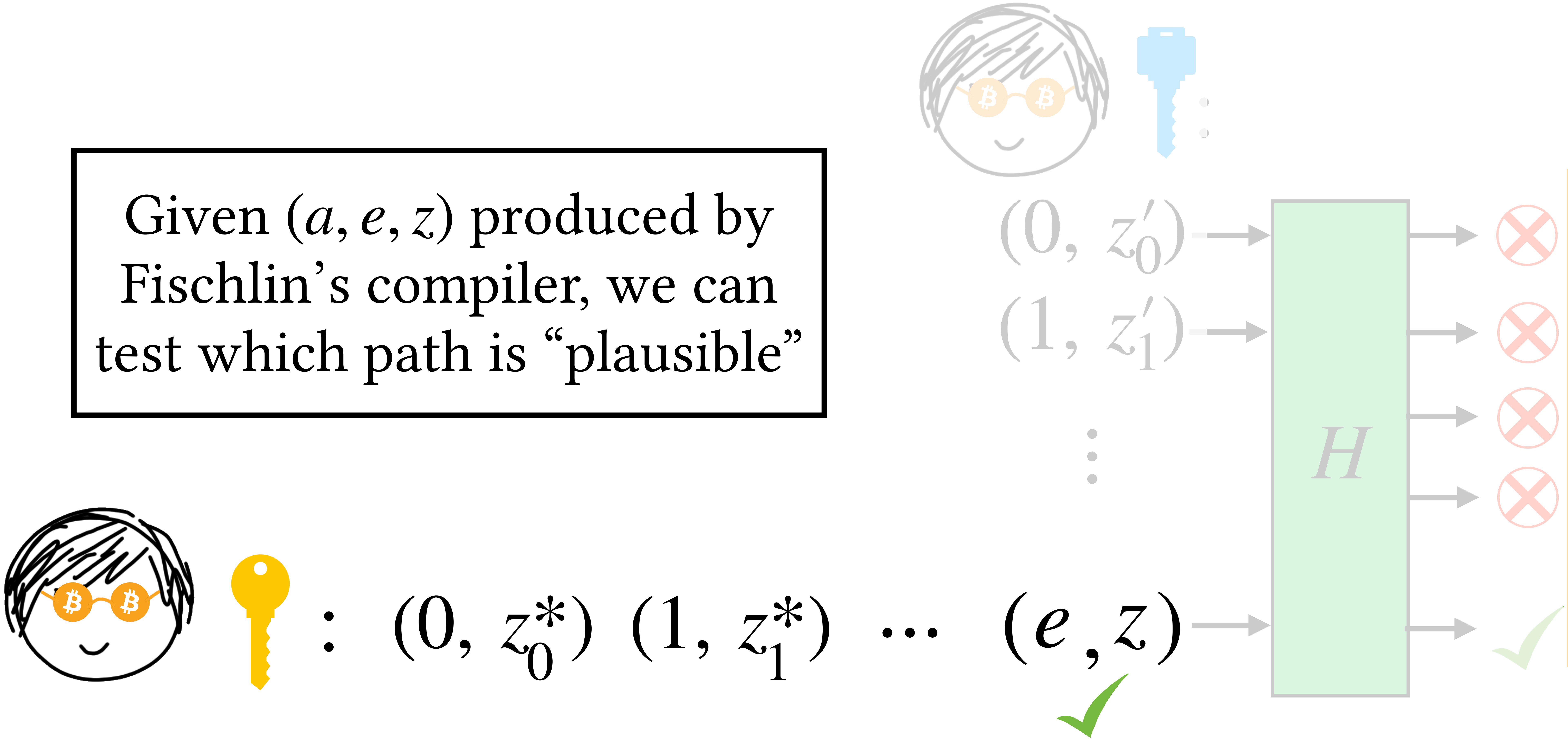
Given (a, e, z) produced by Fischlin's compiler, we can test which path is "plausible"



W.h.p., only one path—induced by one of the two keys—is plausible

Probing Strategy

Given (a, e, z) produced by Fischlin's compiler, we can test which path is "plausible"

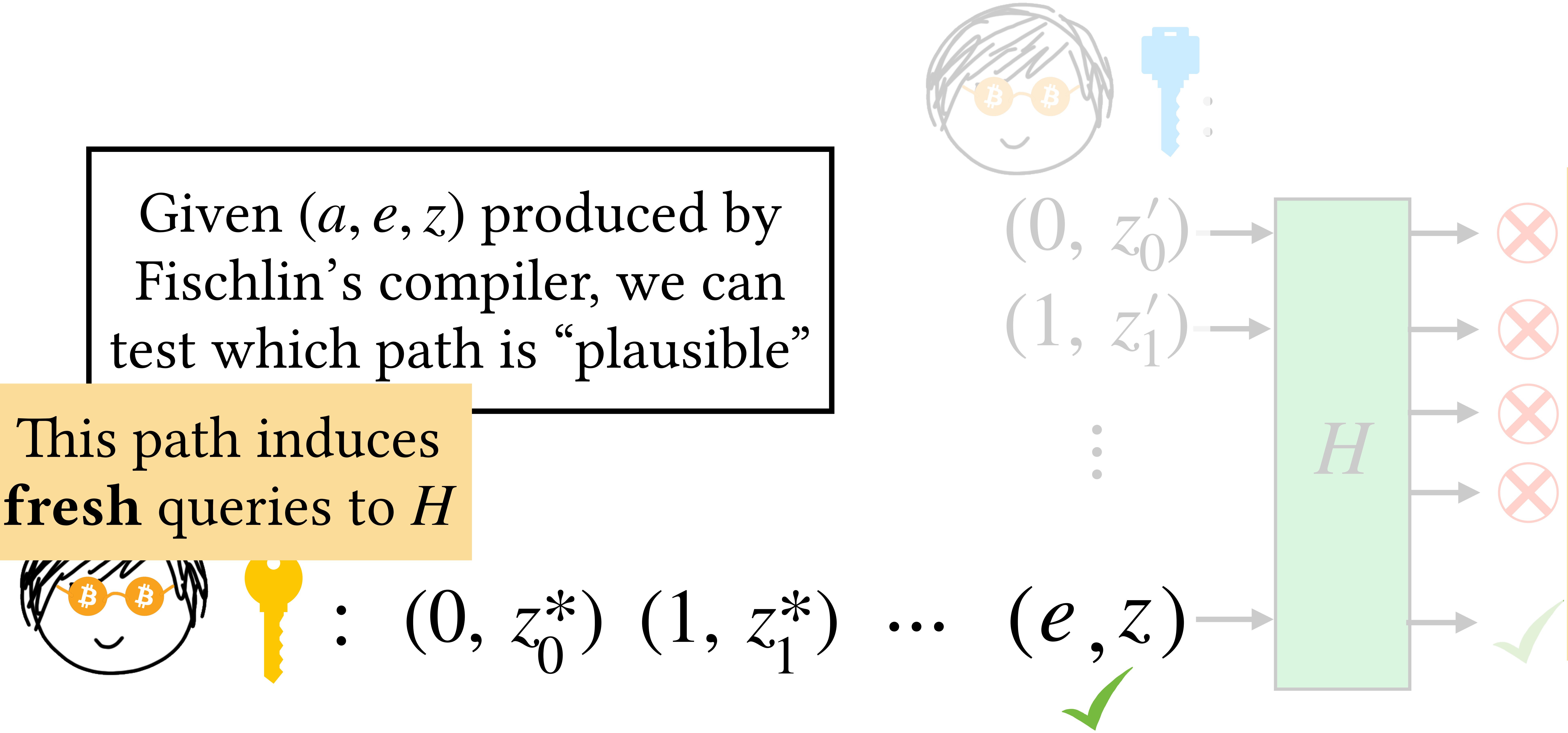


W.h.p., only one path—induced by one of the two keys—is plausible

Probing Strategy

Given (a, e, z) produced by Fischlin's compiler, we can test which path is "plausible"

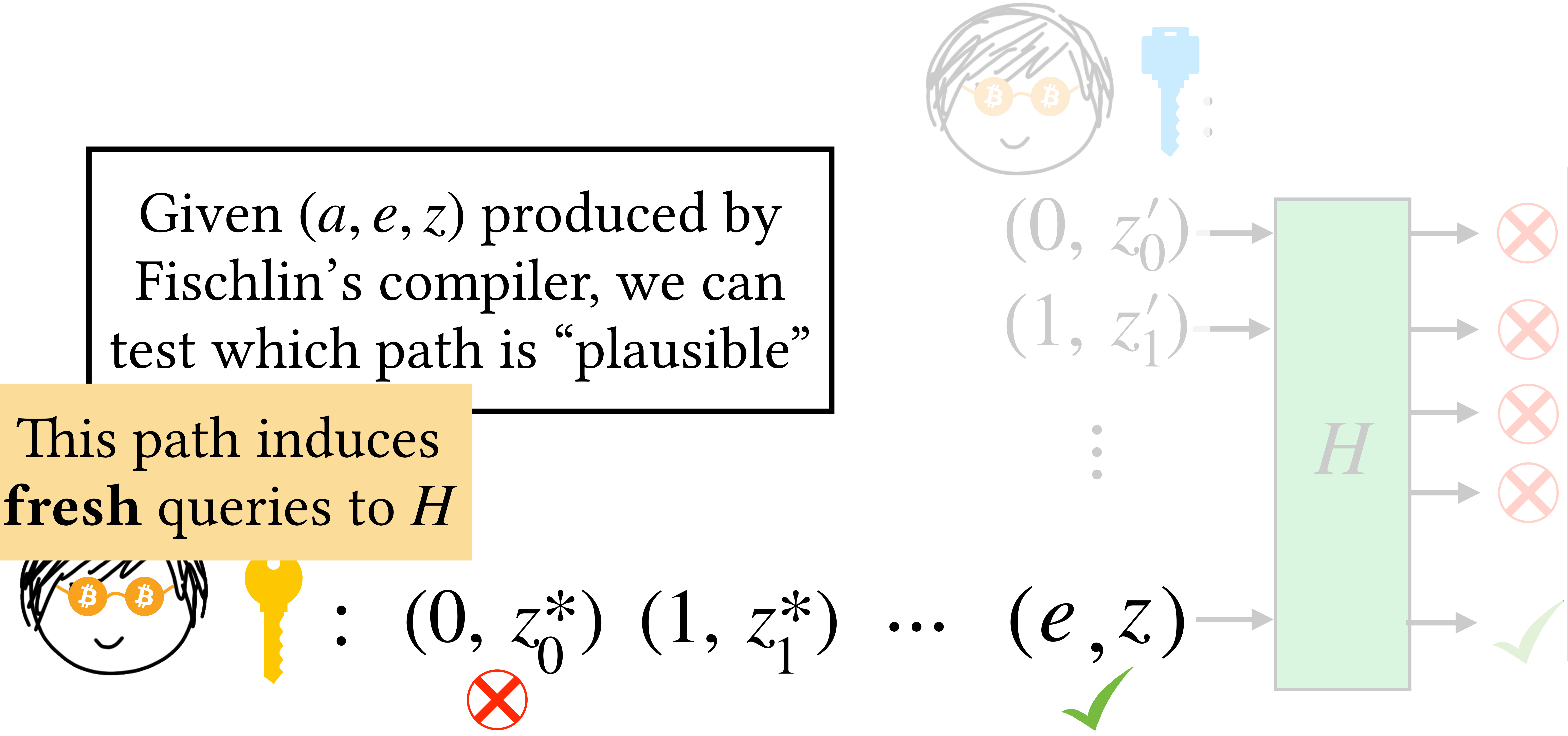
This path induces **fresh** queries to H



W.h.p., only one path— induced by one of the two keys — is plausible

Probing Strategy

Given (a, e, z) produced by Fischlin's compiler, we can test which path is "plausible"

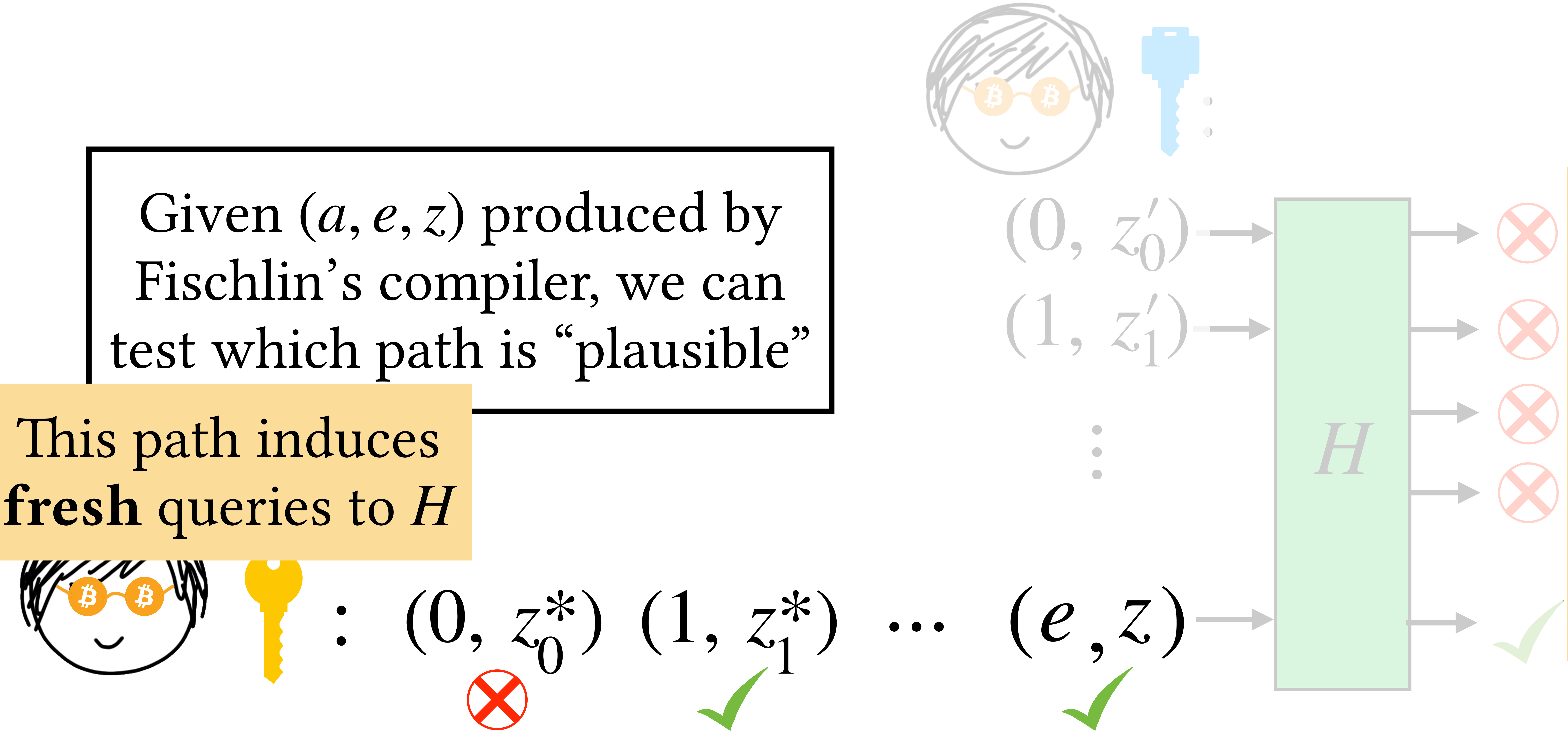


This path induces **fresh** queries to H

W.h.p., only one path— induced by one of the two keys — is plausible

Probing Strategy

Given (a, e, z) produced by Fischlin's compiler, we can test which path is "plausible"

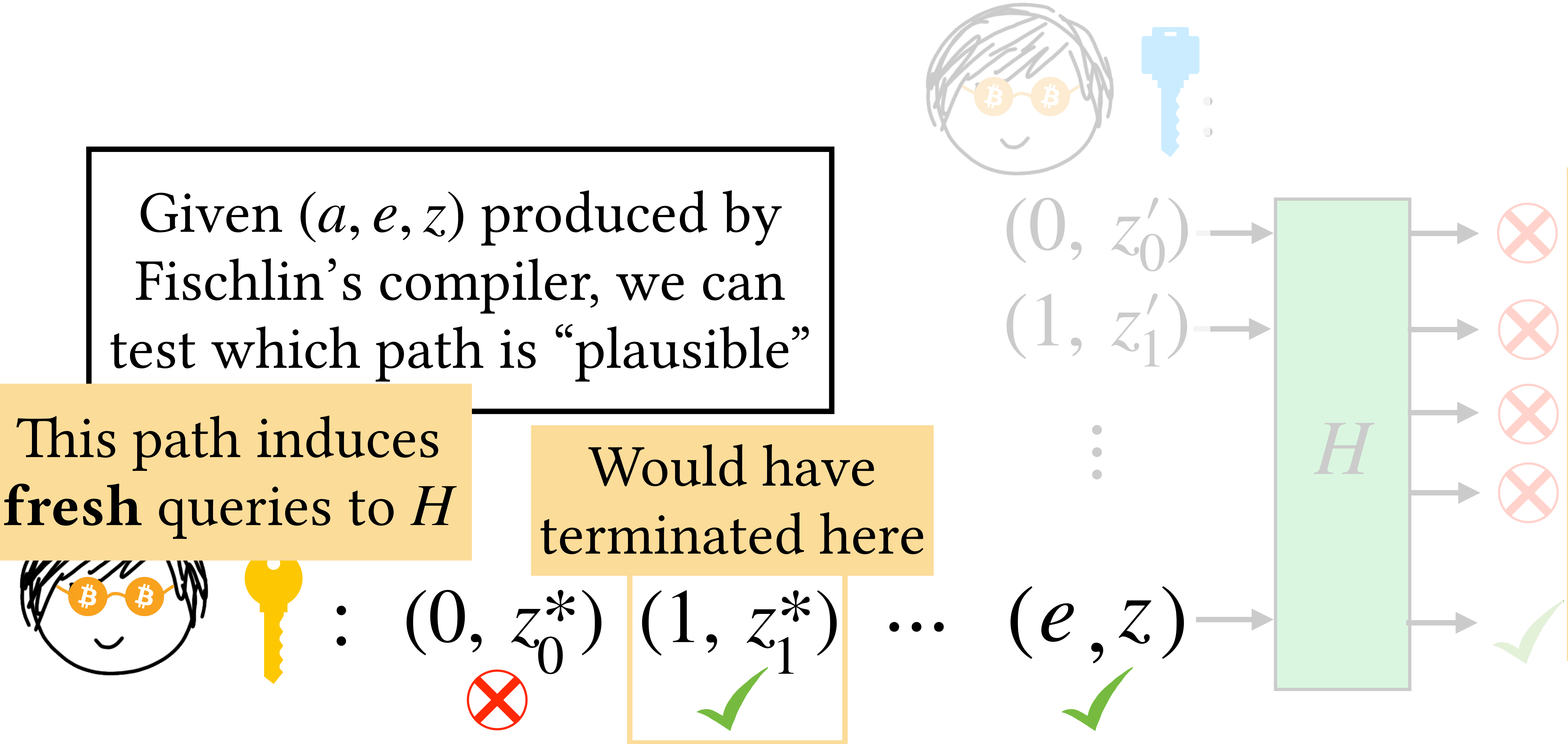


This path induces **fresh** queries to H

W.h.p., only one path—induced by one of the two keys—is plausible

Probing Strategy

Given (a, e, z) produced by Fischlin's compiler, we can test which path is "plausible"



This path induces **fresh** queries to H

Would have terminated here

W.h.p., only one path—induced by one of the two keys—is plausible

How to Fix it? [Ks 22]

- The probing strategy very strongly depends on being able to "re-trace" the Prover's steps
  - This is enabled by the deterministic nature of Fischlin's compiler

- We showed that randomizing the order in which the Prover tries challenges will fix the problem

- We strengthen Fischlin's technique to be good enough to apply to most useful Sigma protocols

In Summary

- We saw what non-interactive zero-knowledge proofs of knowledge are, how they can be used

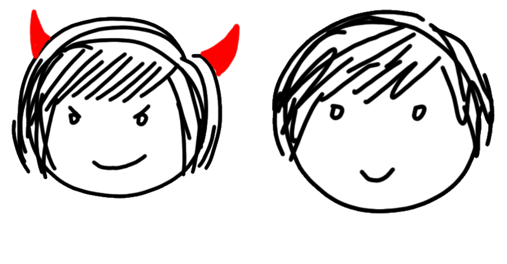
- We got a taste for how they are designed and analysed, and how to understand security guarantees like concurrent composition and ROM

- We uncovered a gap in the literature that was glossed over as folklore, and saw how it turned out to be a vulnerability
  (and briefly discussed how it's now fixed)

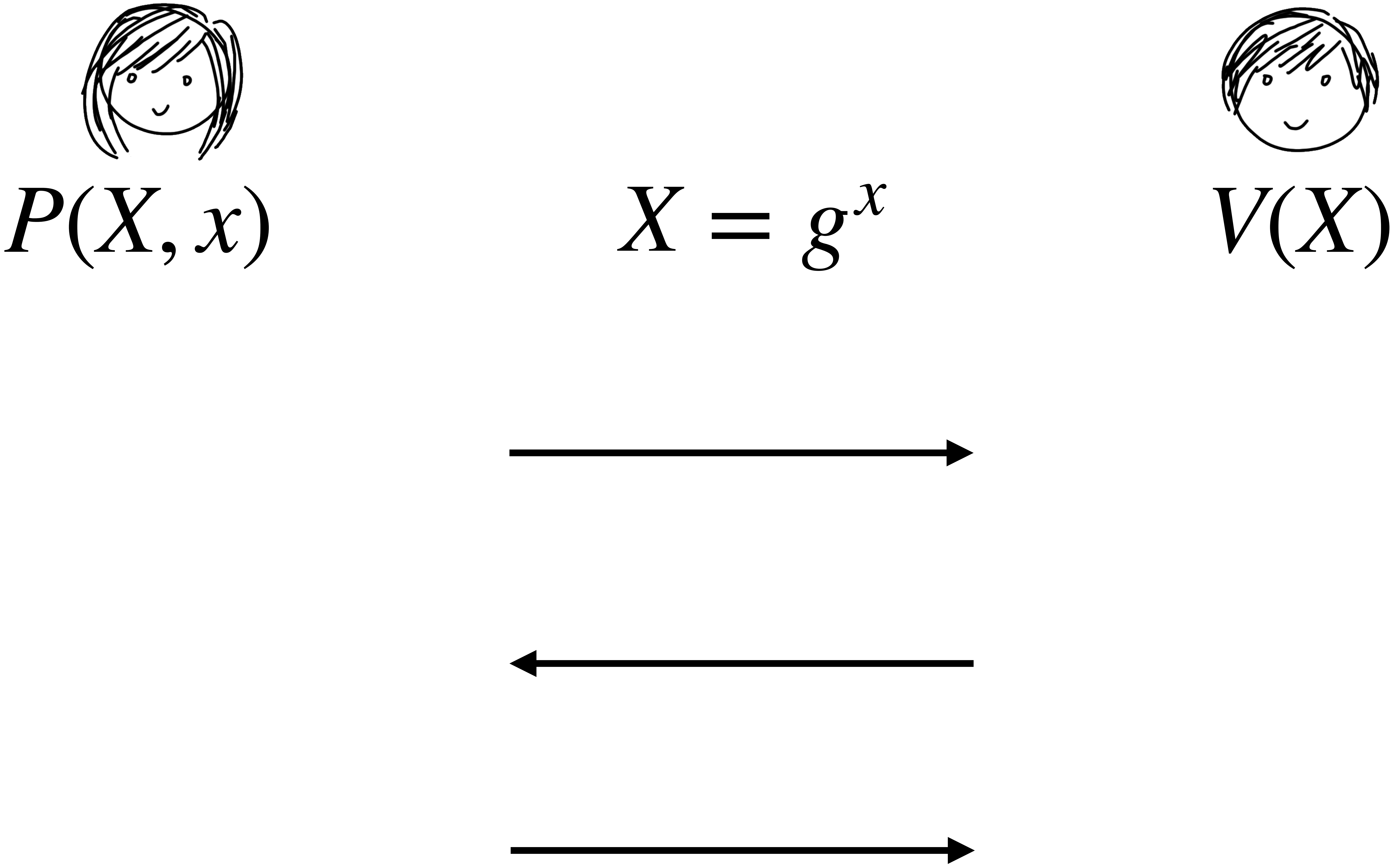
Questions?

eprint.iacr.org/2022/393

Thanks **Eysa Lee** for



Example: Schnorr PoK of Discrete Logarithm

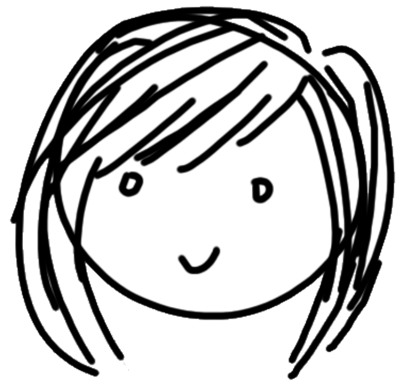


$P(X, x)$

$X = g^x$

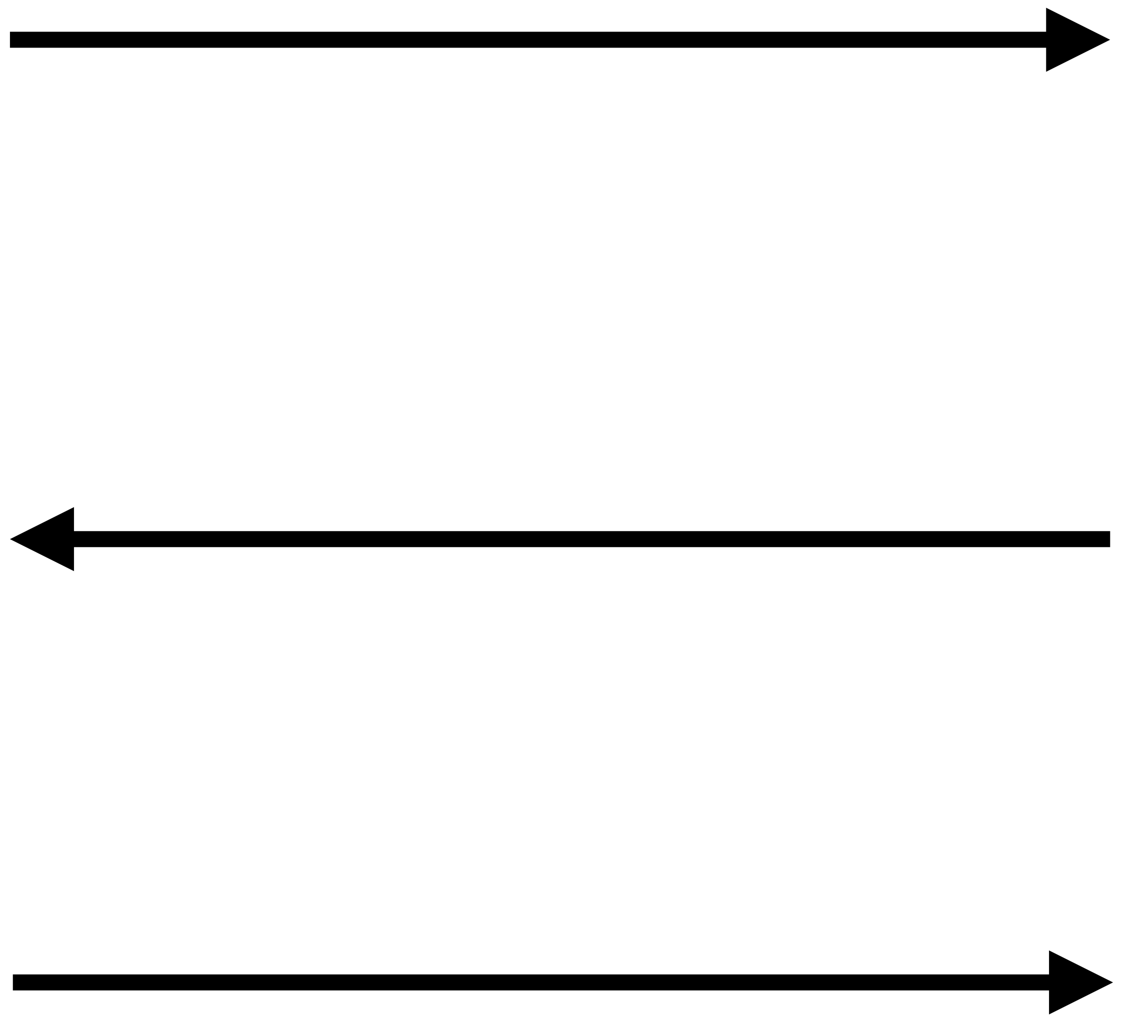
$V(X)$

Example: Schnorr PoK of Discrete Logarithm

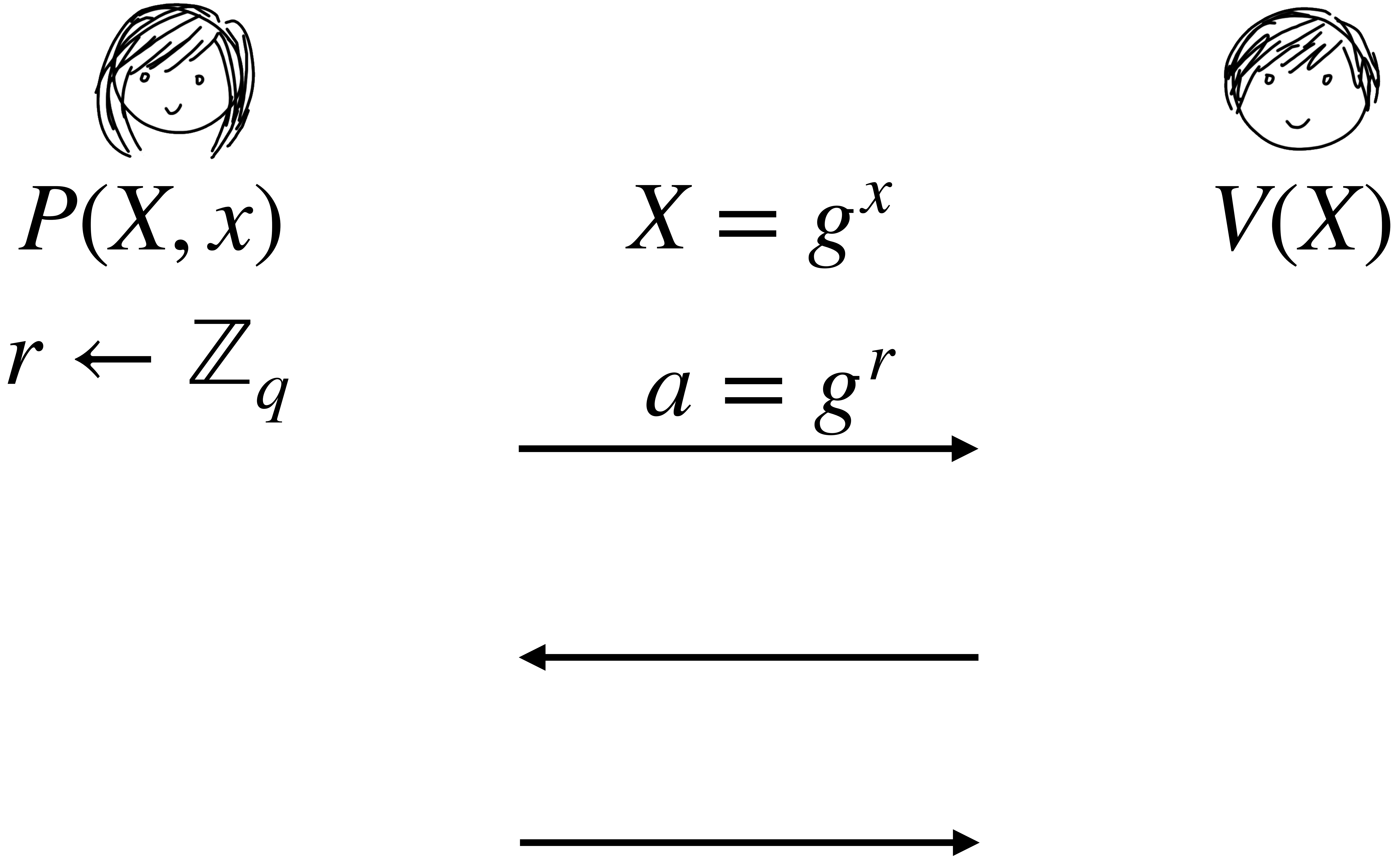


$P(X, x)$ $\quad\quad$ $X = g^x$ $\quad\quad$ $V(X)$

$r \leftarrow \mathbb{Z}_q$



Example: Schnorr PoK of Discrete Logarithm



$P(X, x)$

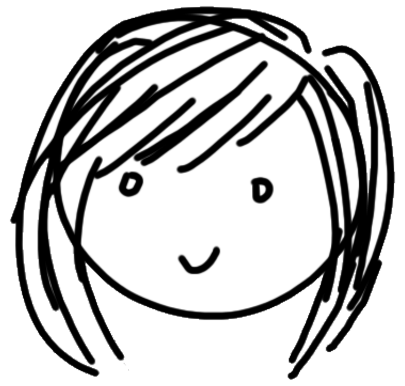
$X = g^x$

$V(X)$

$r \leftarrow \mathbb{Z}_q$

$a = g^r$

Example: Schnorr PoK of Discrete Logarithm



$P(X, x)$ $\qquad$ $X = g^x$ $\qquad$ $V(X)$

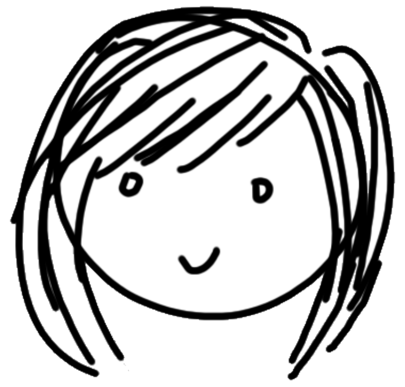
$r \leftarrow \mathbb{Z}_q$

$$\xrightarrow{\qquad a = g^r \qquad}$$

$$\xleftarrow{\qquad e \in \mathbb{Z}_q \qquad}$$

$$\xrightarrow{\qquad\qquad\qquad}$$

Example: Schnorr PoK of Discrete Logarithm



$P(X, x)$ $\quad\quad X = g^x \quad\quad$ $V(X)$

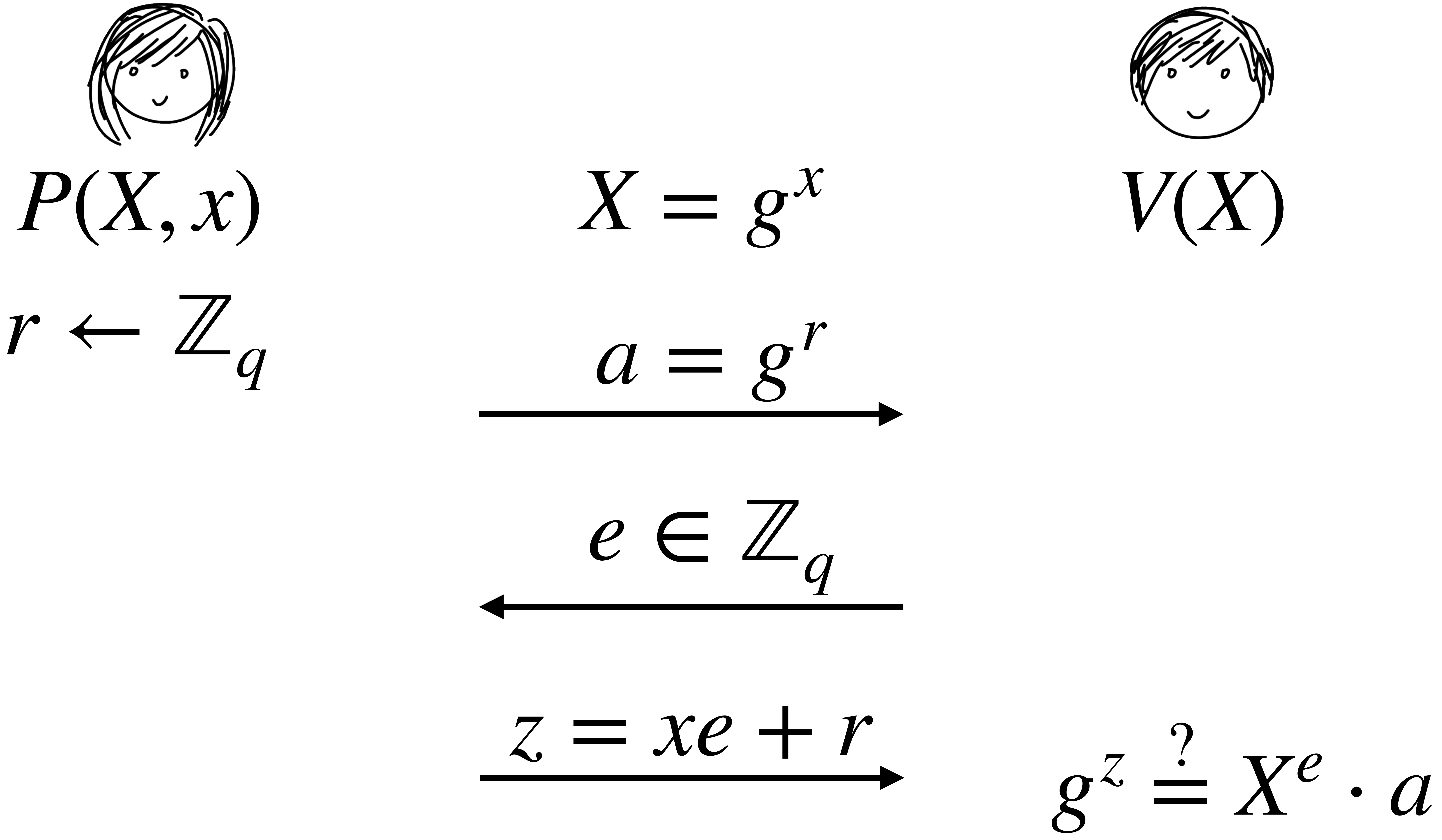
$r \leftarrow \mathbb{Z}_q$

$$\xrightarrow{\quad a = g^r \quad}$$

$$\xleftarrow{\quad e \in \mathbb{Z}_q \quad}$$

$$\xrightarrow{\quad z = xe + r \quad}$$

Example: Schnorr PoK of Discrete Logarithm



$P(X, x)$ $\quad X = g^x \quad$ $V(X)$

$r \leftarrow \mathbb{Z}_q$

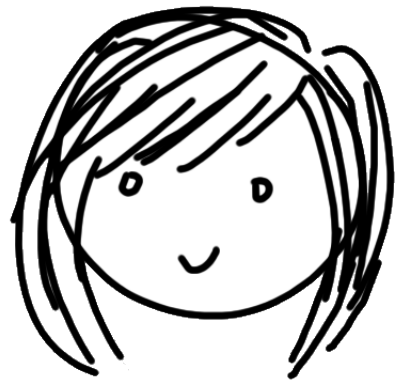
$$\xrightarrow{a = g^r}$$

$$\xleftarrow{e \in \mathbb{Z}_q}$$

$$\xrightarrow{z = xe + r}$$

$$g^z \overset{?}{=} X^e \cdot a$$

Example: Schnorr PoK of Discrete Logarithm



$P(X, x)$ $\qquad X = g^x \qquad$ $V(X)$

$r \leftarrow \mathbb{Z}_q$

$$\xrightarrow{a = g^r}$$

$$\xleftarrow{e \in \mathbb{Z}_q}$$

$$\xrightarrow{z = xe + r}$$

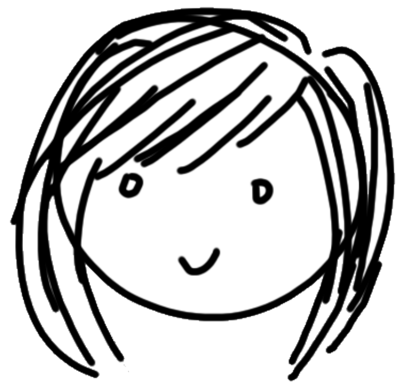
$$g^z \stackrel{?}{=} X^e \cdot a$$

Ext$(a, (e, z), (e', z'))$:

$x = (z' - z)/(e' - e)$

Output x

Example: Schnorr PoK of Discrete Logarithm



$P(X, x)$ $\quad X = g^x \quad$ $V(X)$

$r \leftarrow \mathbb{Z}_q$

$$\xrightarrow{\quad a = g^r \quad}$$

<u>HVZK $\mathcal{S}(e)$:</u>

$z \leftarrow \mathbb{Z}_q$

$a = g^z / X^e$

Output (a, z)

<u>Ext$(a, (e, z), (e', z'))$:</u>

$x = (z' - z)/(e' - e)$

Output x

$$\xleftarrow{\quad e \in \mathbb{Z}_q \quad}$$

$$\xrightarrow{\quad z = xe + r \quad}$$

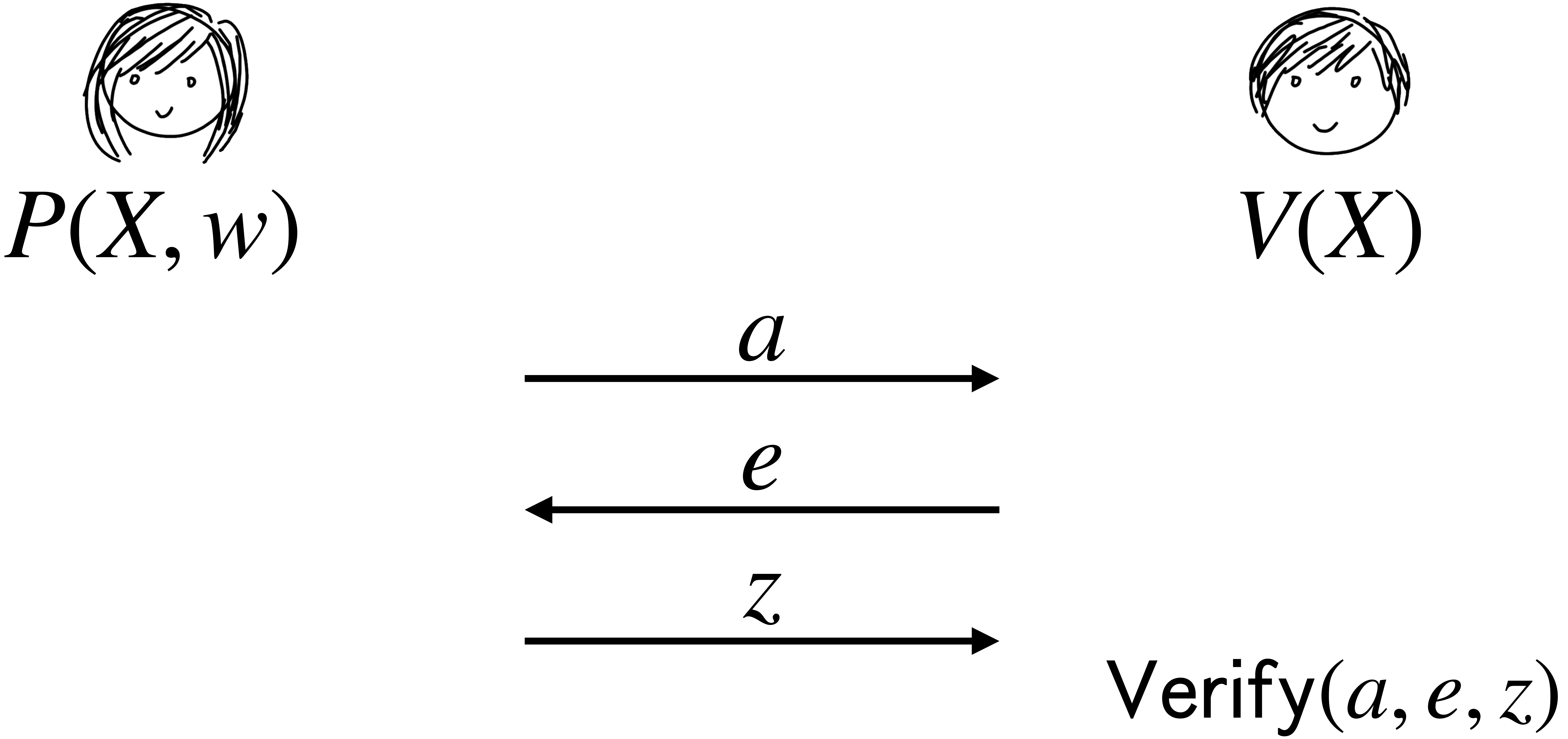
$$g^z \overset{?}{=} X^e \cdot a$$

The Fiat-Shamir Transform

- [Fiat Shamir 87] provides a simple method to compile any public-coin protocol to a non-interactive proof, given a suitably chosen hash function

The Fiat-Shamir Transform

- [Fiat Shamir 87] provides a simple method to compile any public-coin protocol to a non-interactive proof, given a suitably chosen hash function



$P(X, w)$

$V(X)$

a

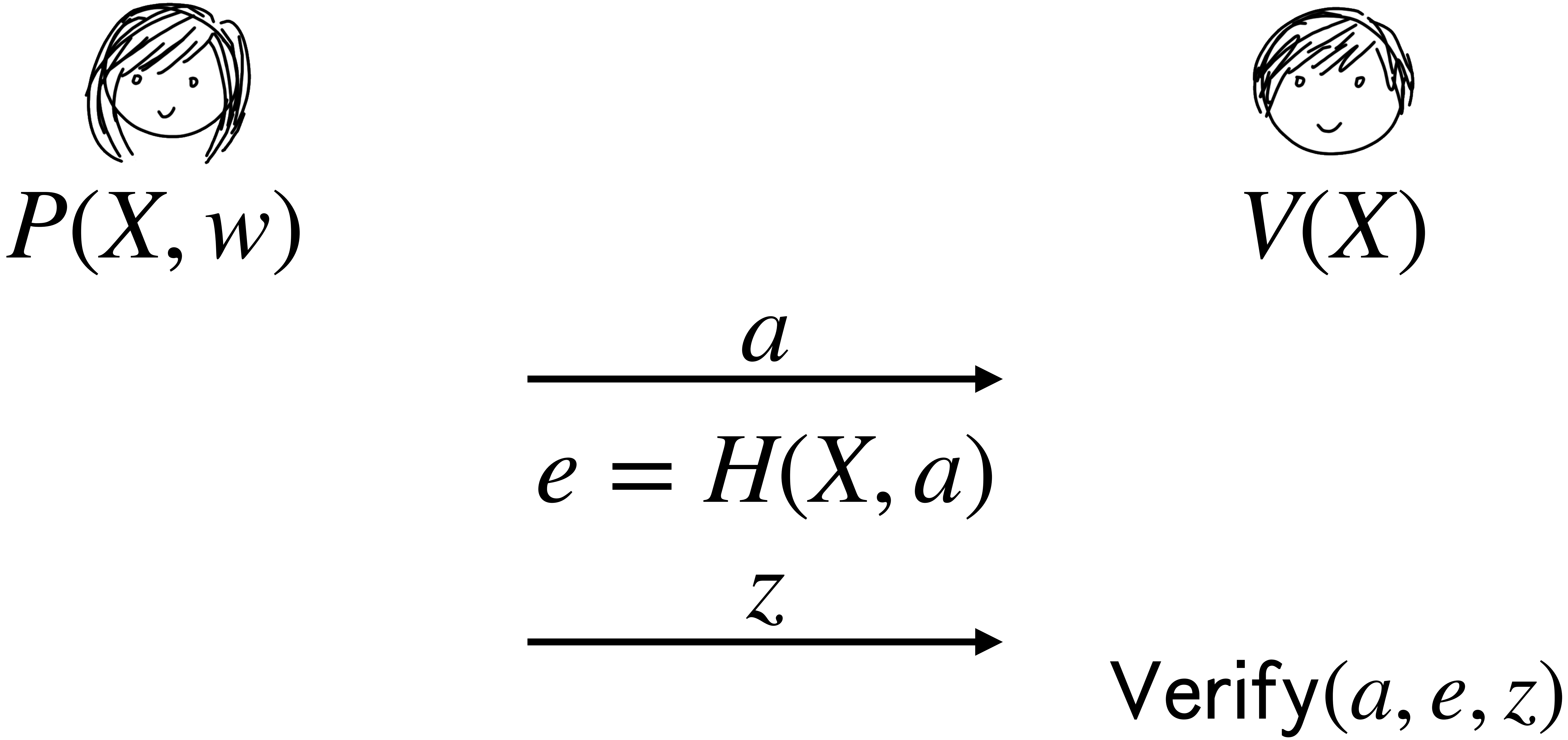
e

z

$\mathsf{Verify}(a, e, z)$

The Fiat-Shamir Transform

- [Fiat Shamir 87] provides a simple method to compile any public-coin protocol to a non-interactive proof, given a suitably chosen hash function



$P(X, w)$

$V(X)$

$$P(X,w) \xrightarrow{a} V(X)$$

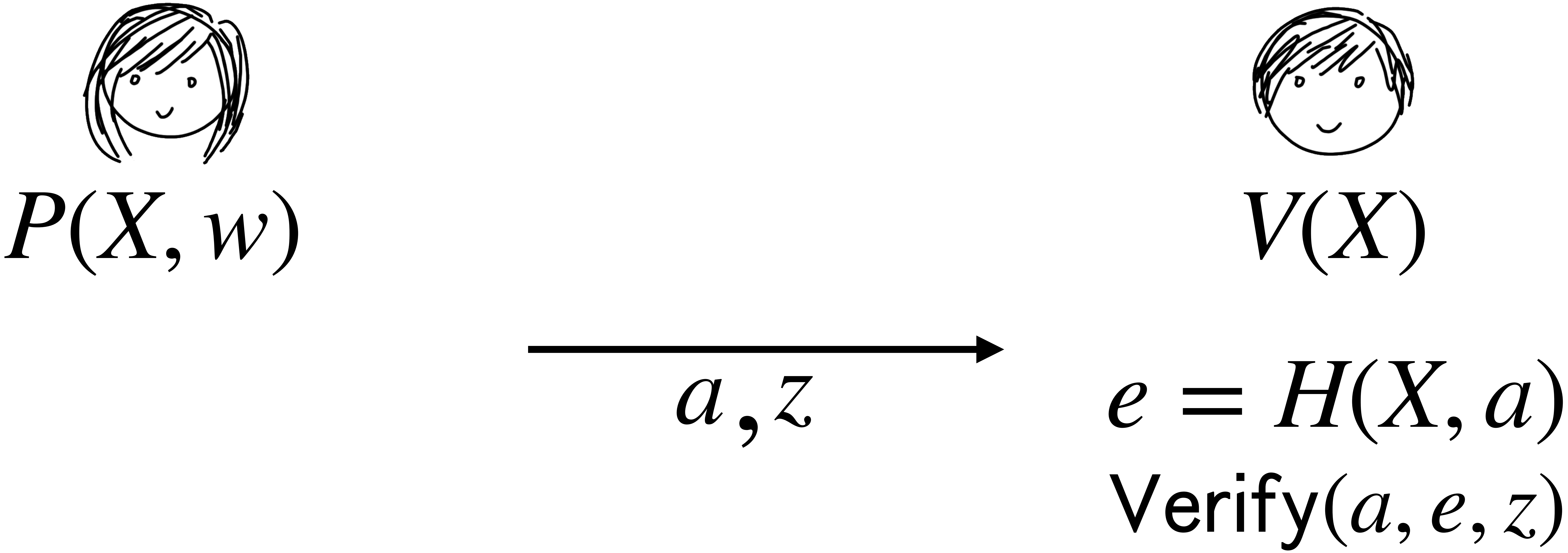
$$e = H(X, a)$$

$$P(X,w) \xrightarrow{z} V(X)$$

$\mathsf{Verify}(a, e, z)$

The Fiat-Shamir Transform

- [Fiat Shamir 87] provides a simple method to compile any public-coin protocol to a non-interactive proof, given a suitably chosen hash function



$P(X, w)$

$V(X)$

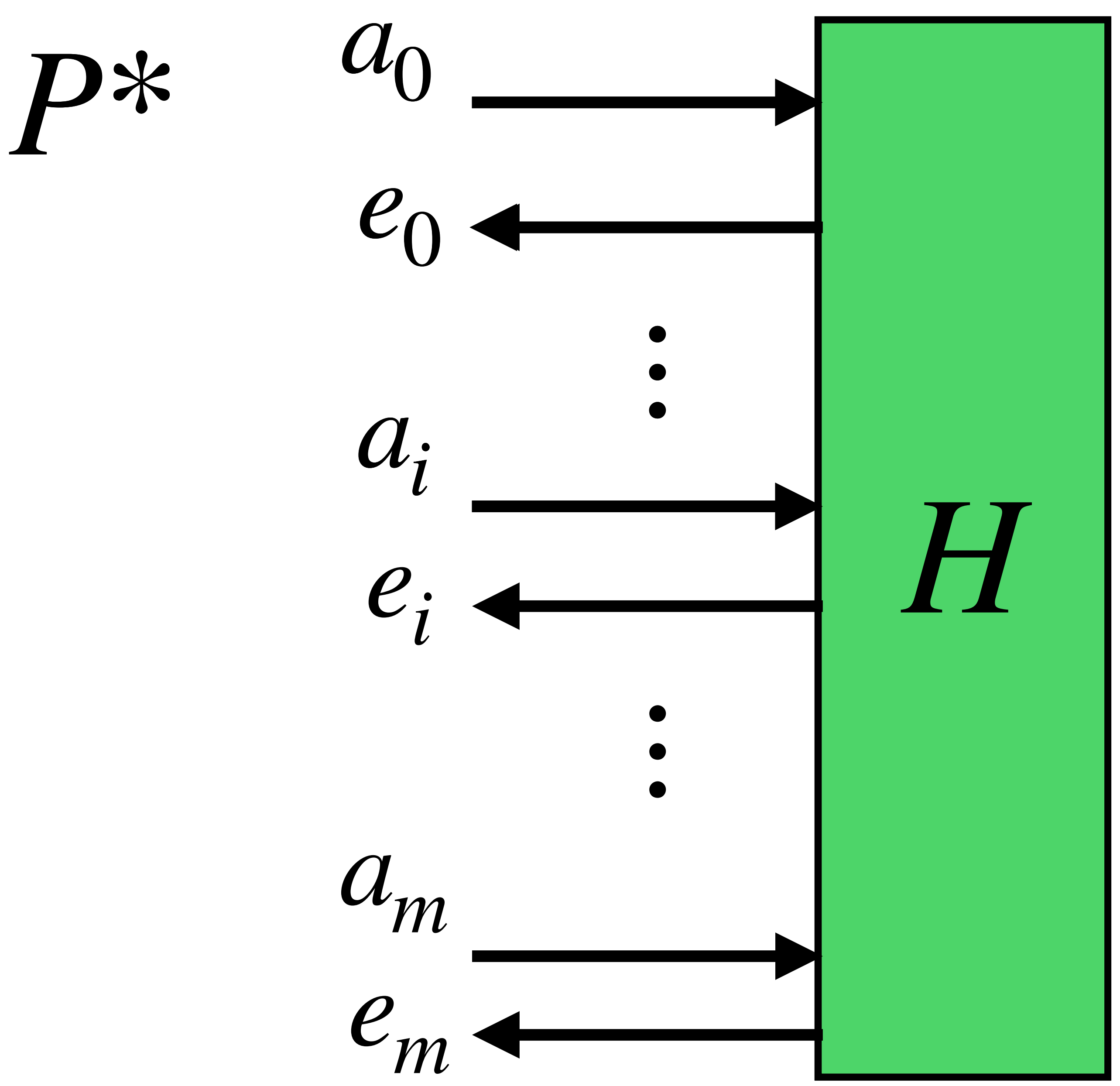
$$\xrightarrow[\quad a,z \quad]{}$$

$e = H(X, a)$

$\mathsf{Verify}(a, e, z)$

Fiat-Shamir: Security

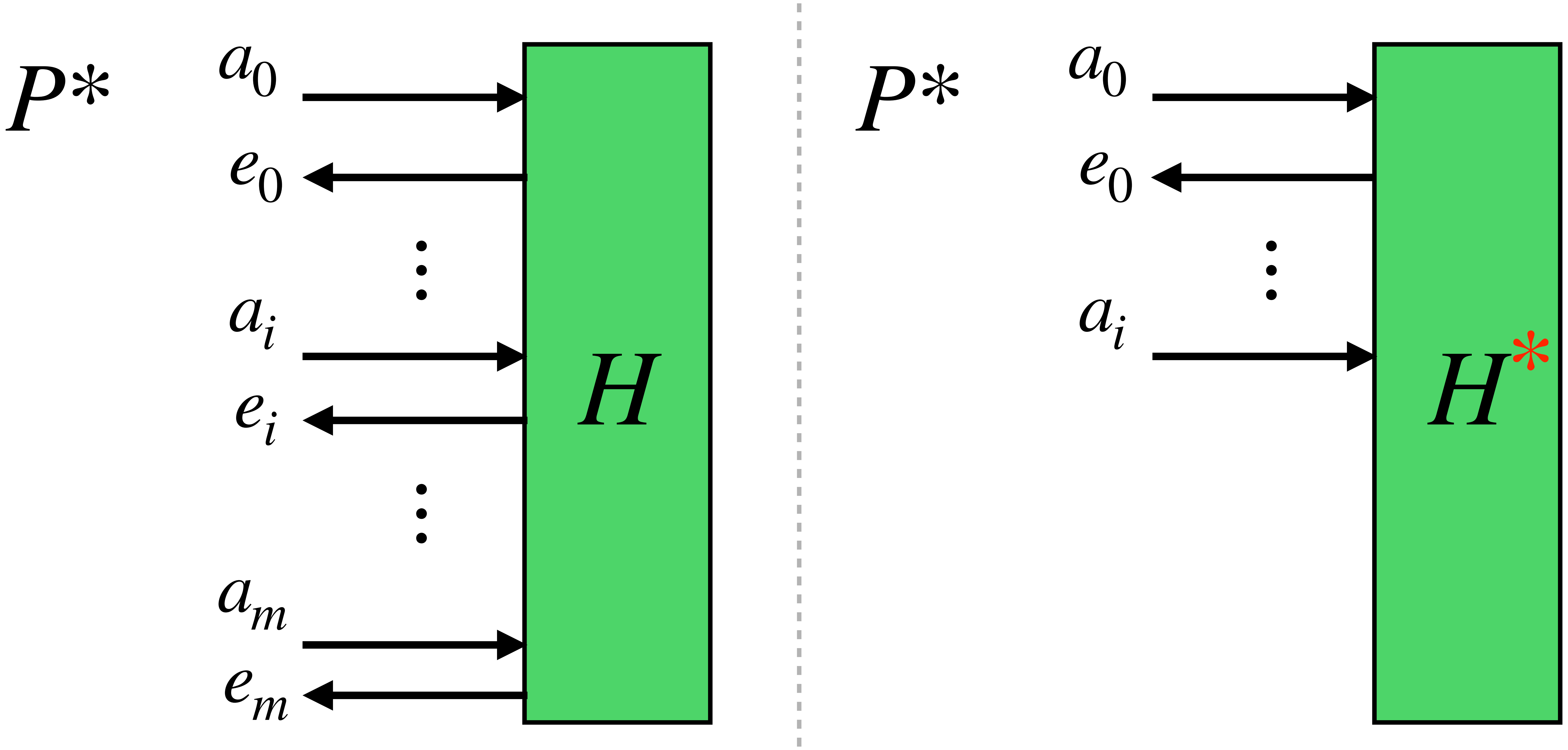
- "Forking" extraction strategy in Random Oracle Model [Pointcheval Stern 96]:



Output (a_i, e_i, z_i)

Fiat-Shamir: Security

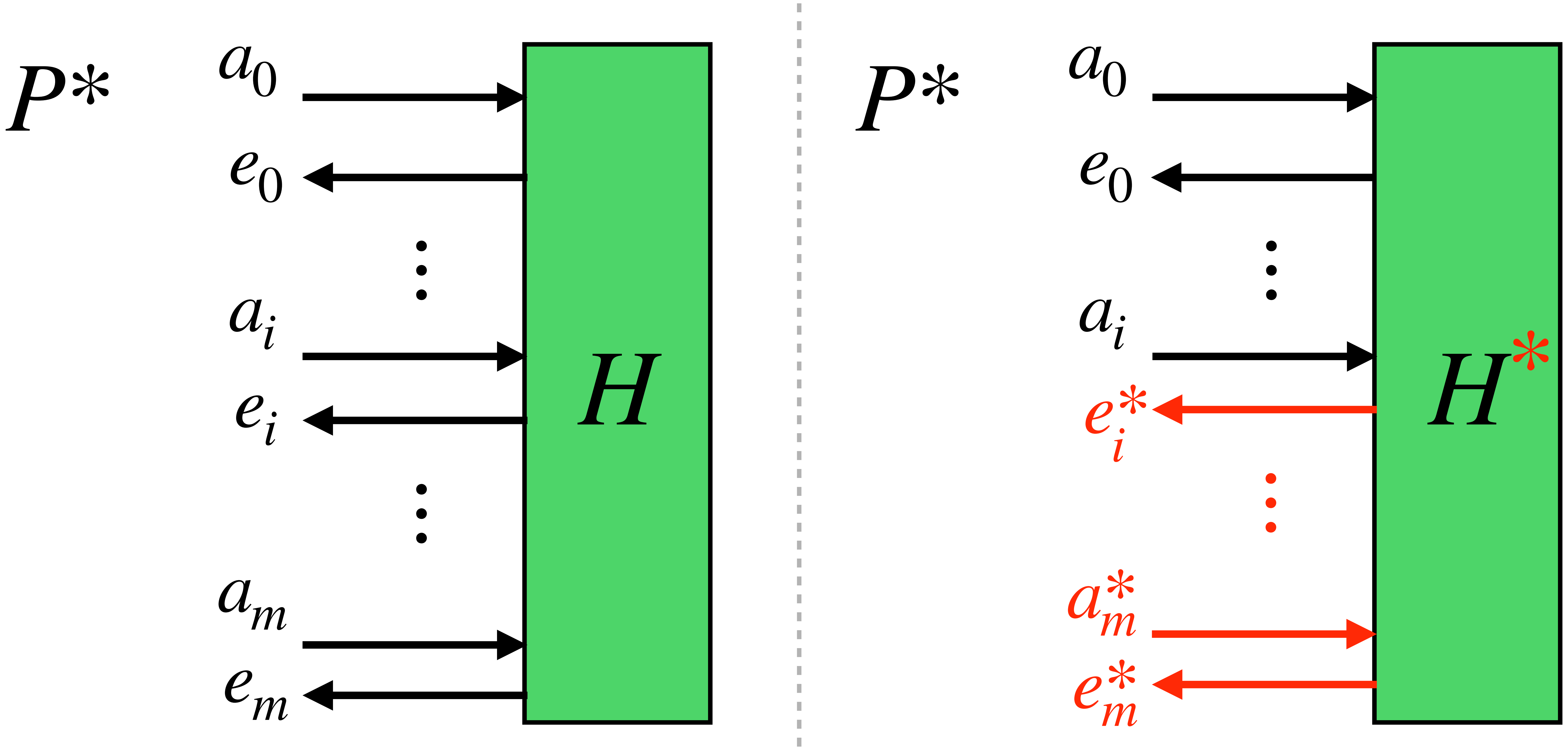
- "Forking" extraction strategy in Random Oracle Model [Pointcheval Stern 96]:



Output (a_i, e_i, z_i)

Fiat-Shamir: Security

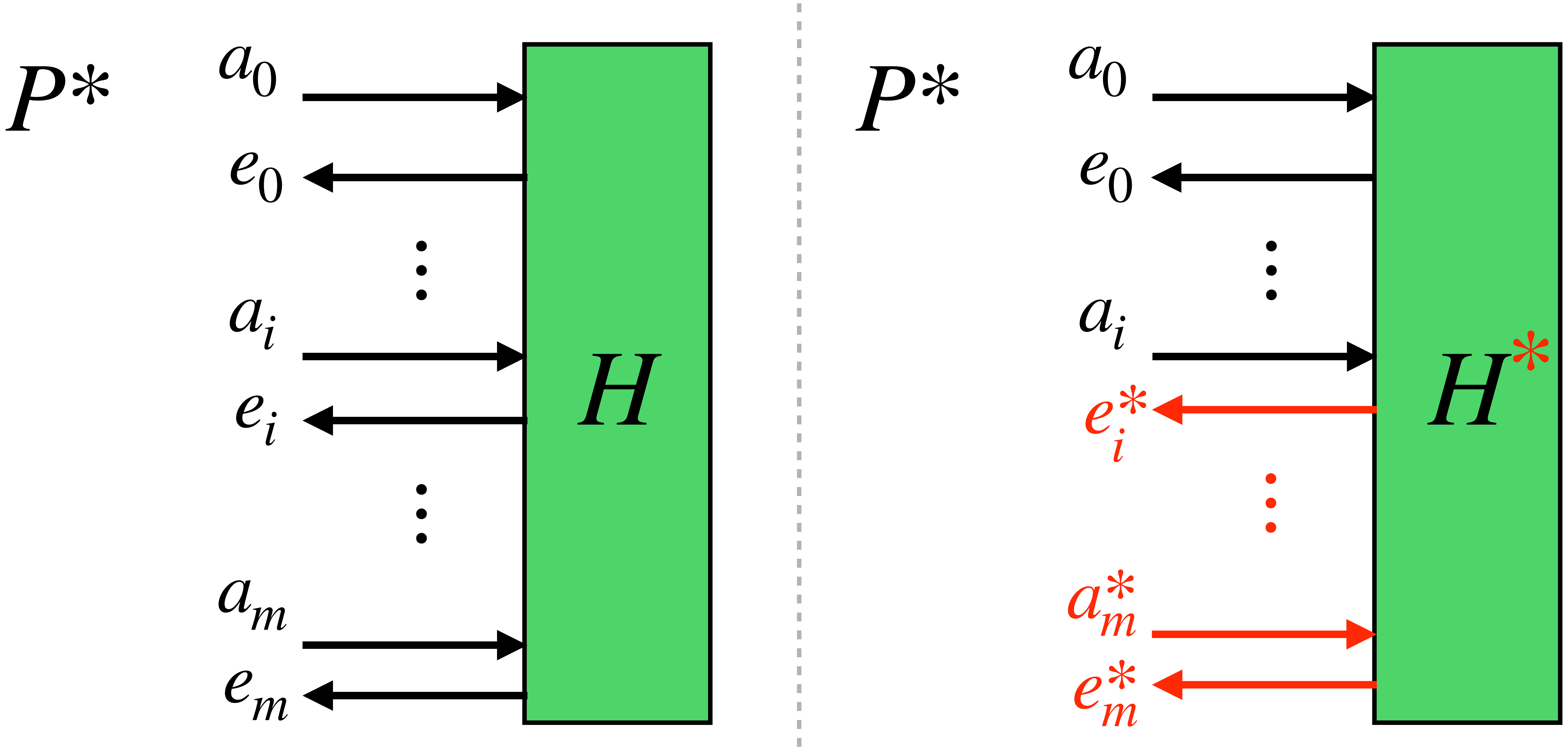
- “Forking” extraction strategy in Random Oracle Model [Pointcheval Stern 96]:



Output (a_i, e_i, z_i)

Fiat-Shamir: Security

- “Forking” extraction strategy in Random Oracle Model [Pointcheval Stern 96]:

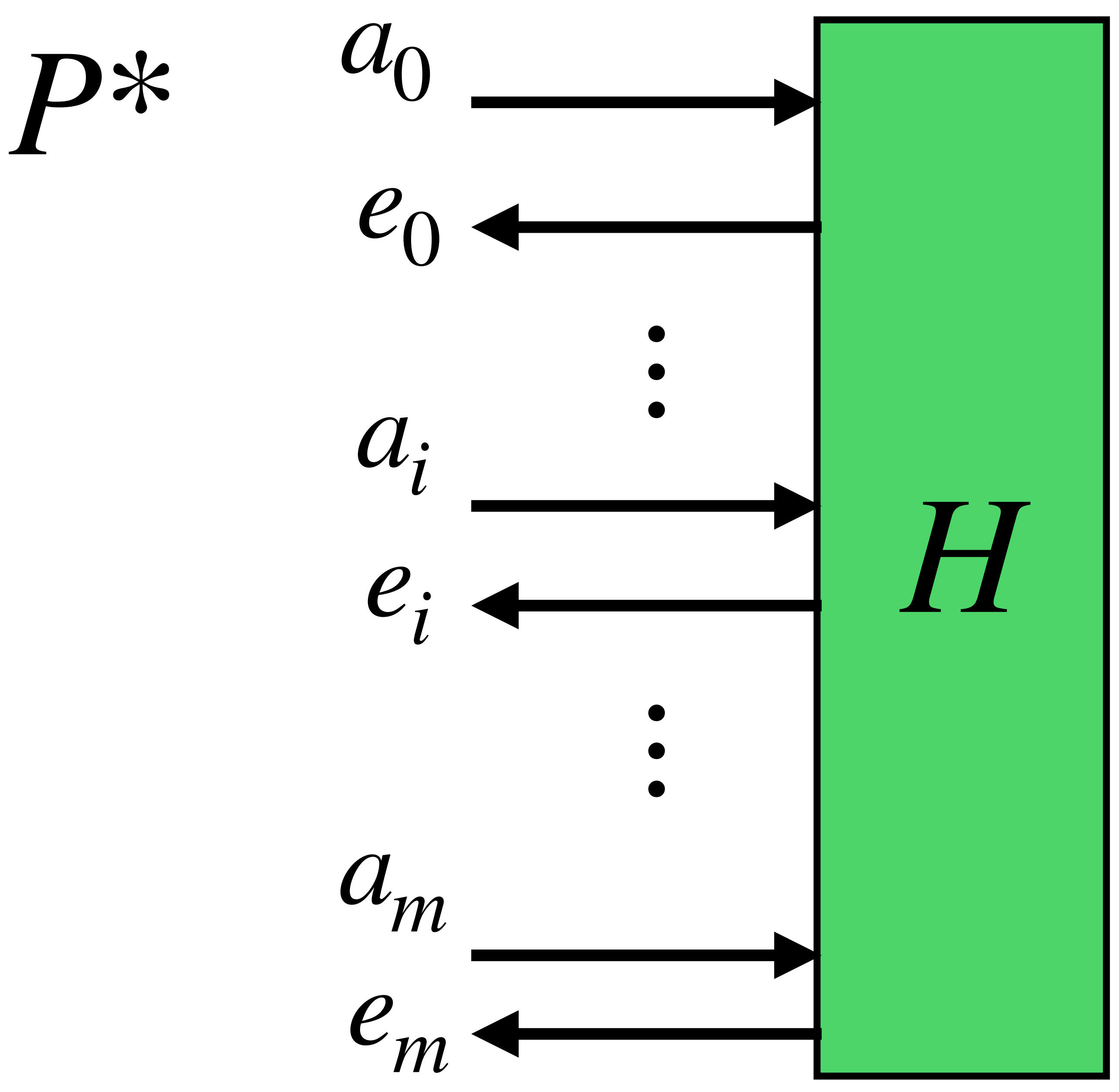


Output (a_i, e_i, z_i)

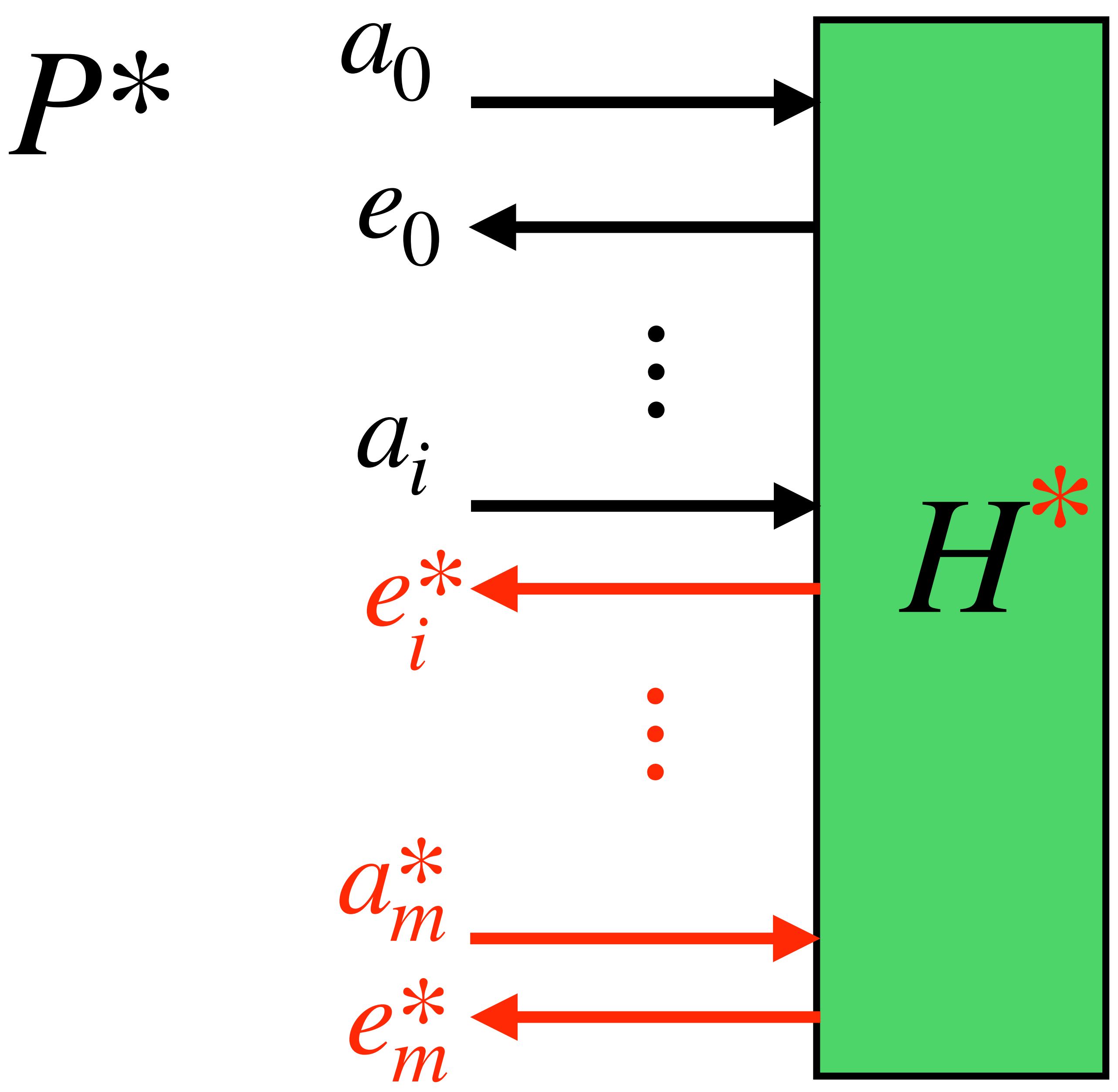
Output (a_i, e_i^*, z_i^*)

Fiat-Shamir: Security

- “Forking” extraction strategy in Random Oracle Model [Pointcheval Stern 96]:



Output (a_i, e_i, z_i)



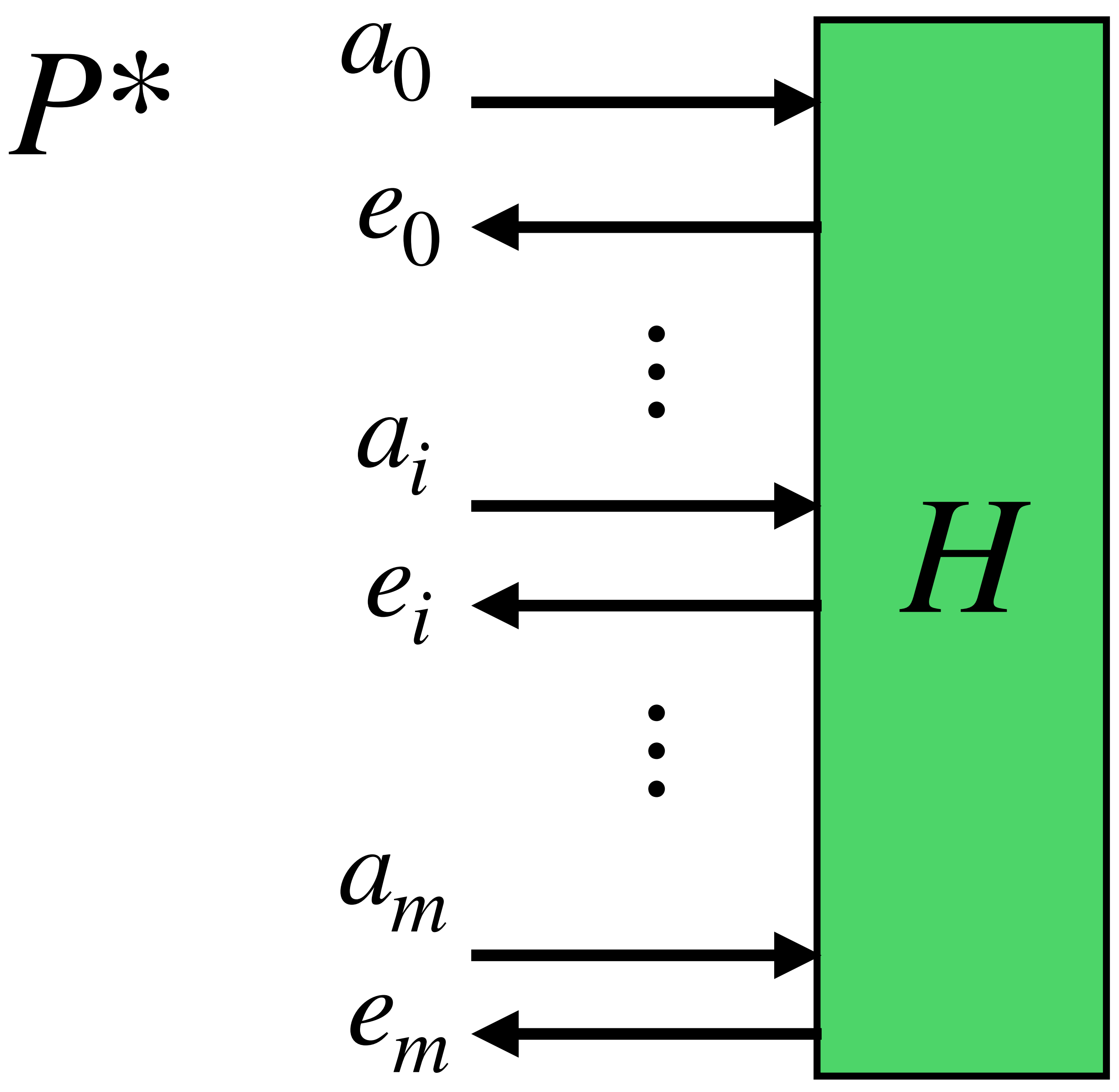
Output (a_i, e_i^*, z_i^*)

$$\mathsf{Ext}\begin{pmatrix}(a_i, e_i) & (a_i, e_i) \\ \multicolumn{2}{c}{z_i, z_i^*}\end{pmatrix}$$

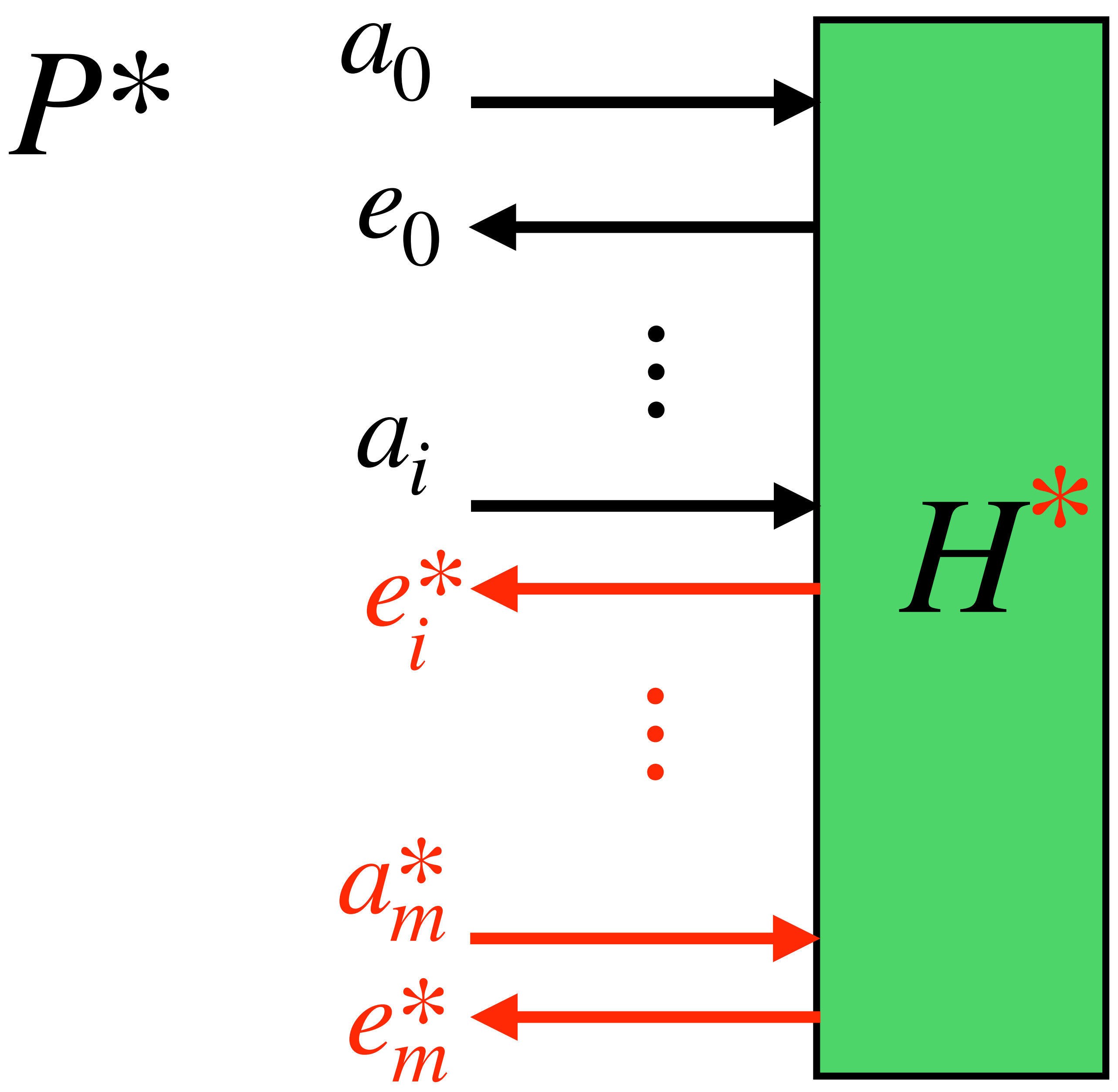
Outputs witness w

Fiat-Shamir: Security

- “Forking” extraction strategy in Random Oracle Model [Pointcheval Stern 96]:



Output (a_i, e_i, z_i)



Output (a_i, e_i^*, z_i^*)

$$\mathsf{Ext}\begin{pmatrix} (a_i, e_i) \;\; (a_i, e_i) \\ z_i, z_i^* \end{pmatrix}$$

Outputs witness w

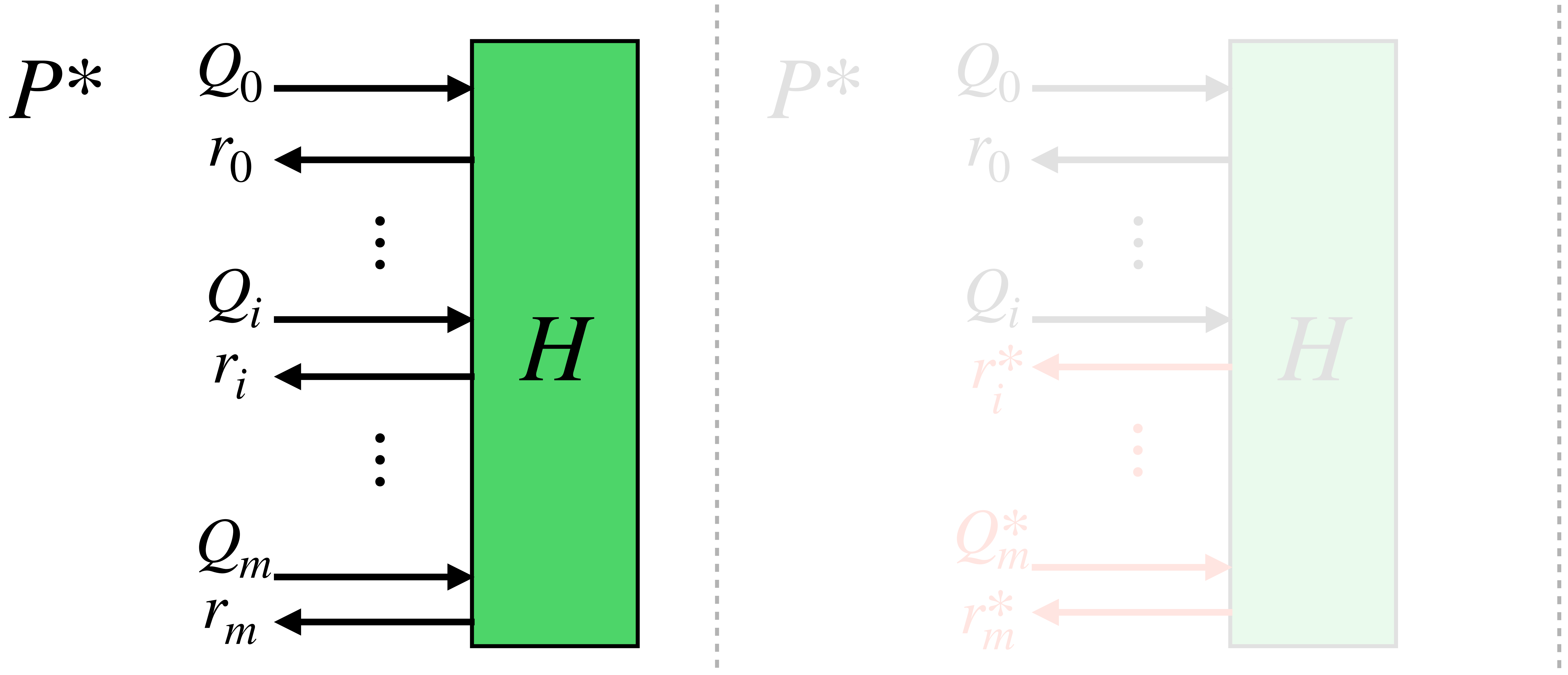
Probability of success: p $\qquad$ p $\qquad$ $\approx p^2$

Fiat-Shamir Compilation

- Advantages:
  - Simple to describe/implement
  - Very efficient; proving, verification cost exactly the same as input Σ-protocol
- Downsides:
  - Forking strategy does not compose; unclear how to prove concurrent security
  - Quadratic security loss

Straight-line Extraction

- Formalized by [Pass 03] in the Random Oracle Model:



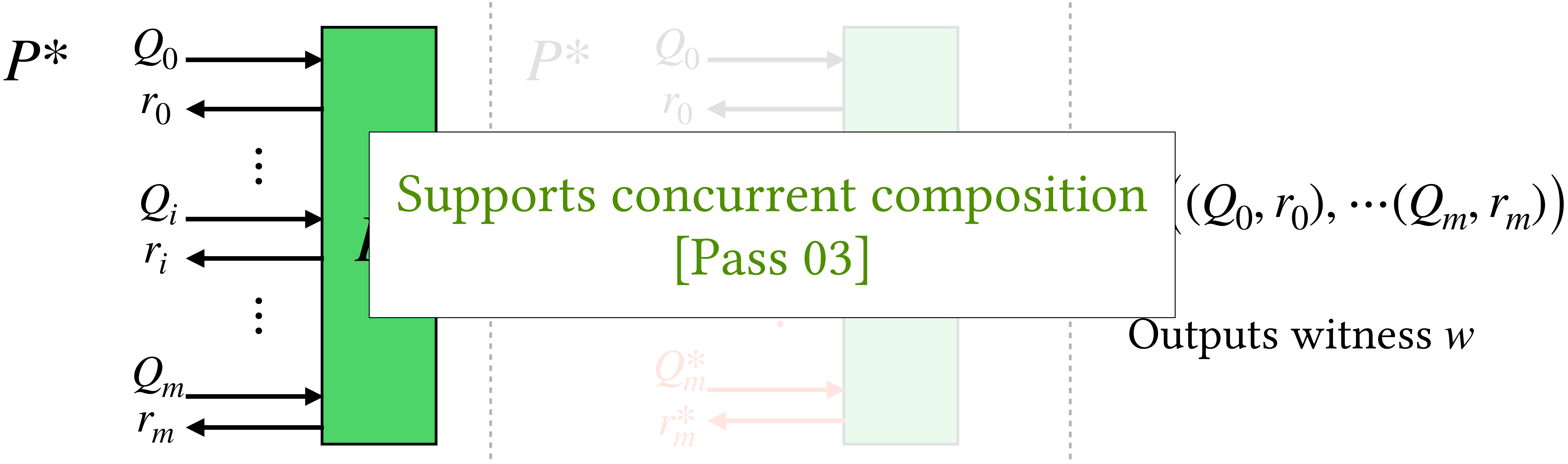
$\mathsf{Ext}\left((Q_0, r_0), \cdots (Q_m, r_m)\right)$

Outputs witness w

Probability of success: p p $\approx p$

Straight-line Extraction

- Formalized by [Pass 03] in the Random Oracle Model:



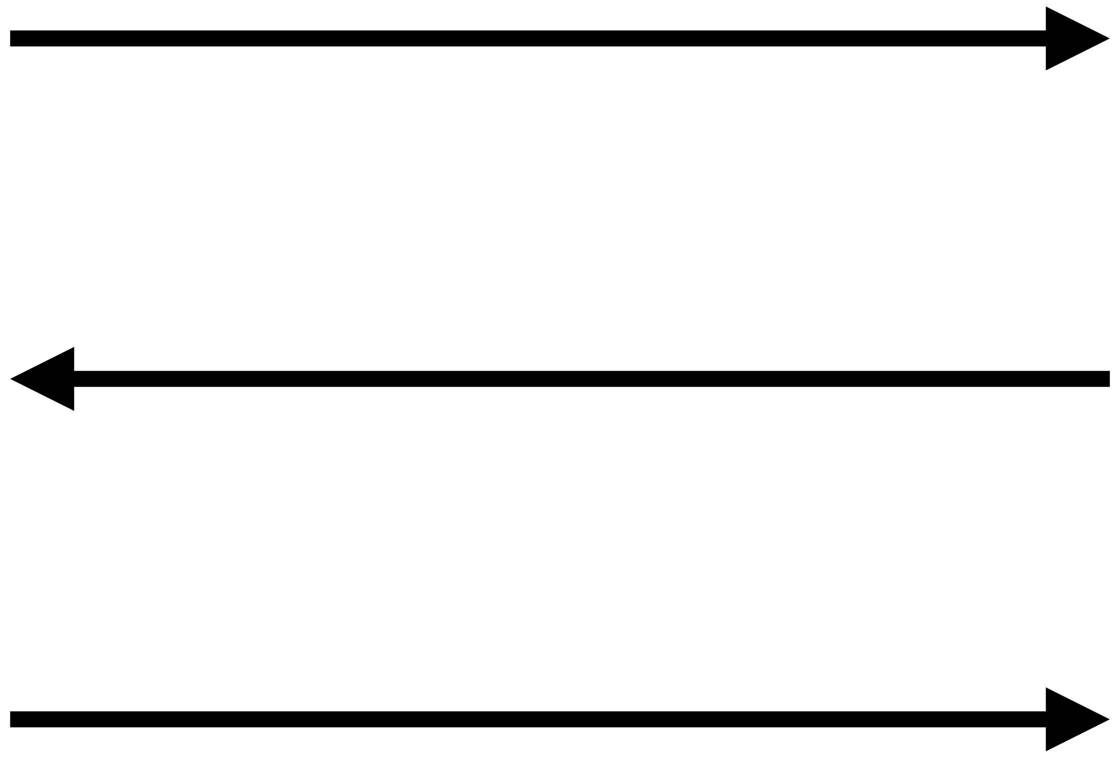
Supports concurrent composition [Pass 03]

Probability of success: p p $\approx p$

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]

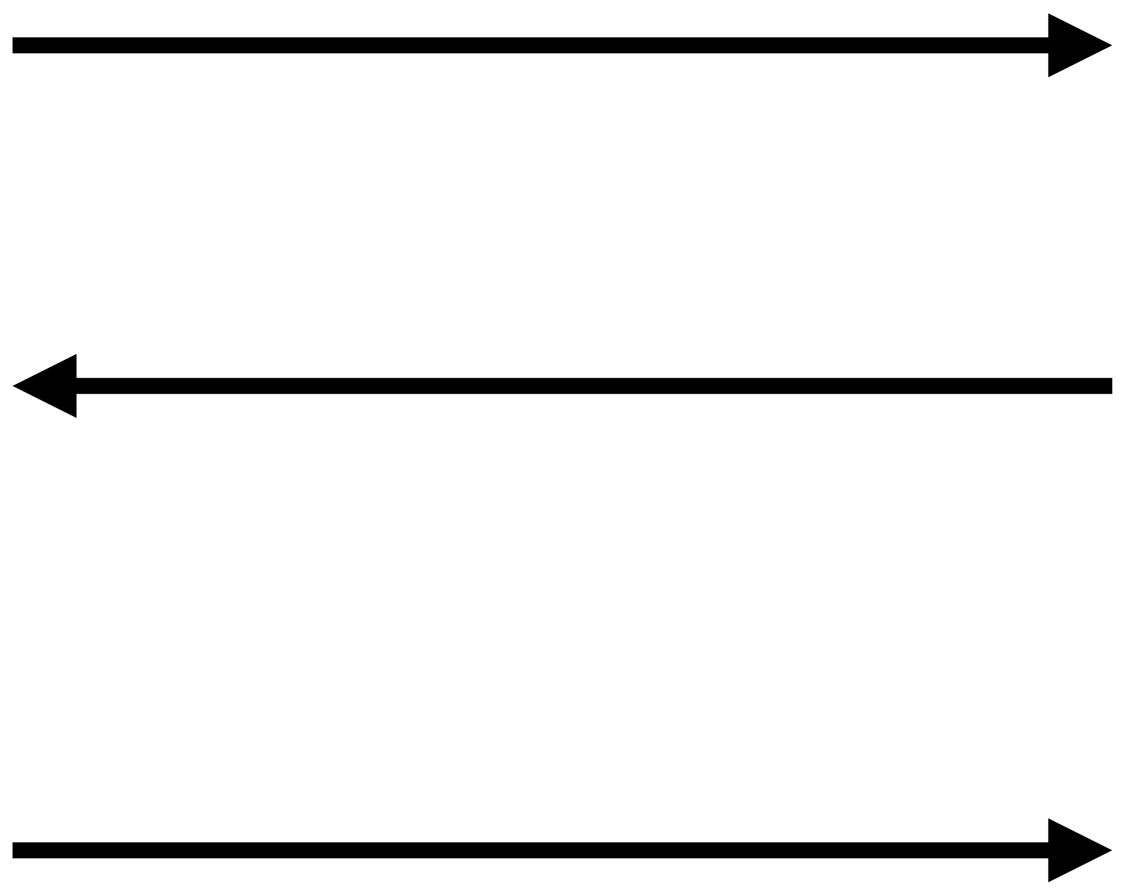
$P_{\mathsf{OR}}(w_b)$ x_0, x_1 V



Logical OR-Composition of Σ Protocols

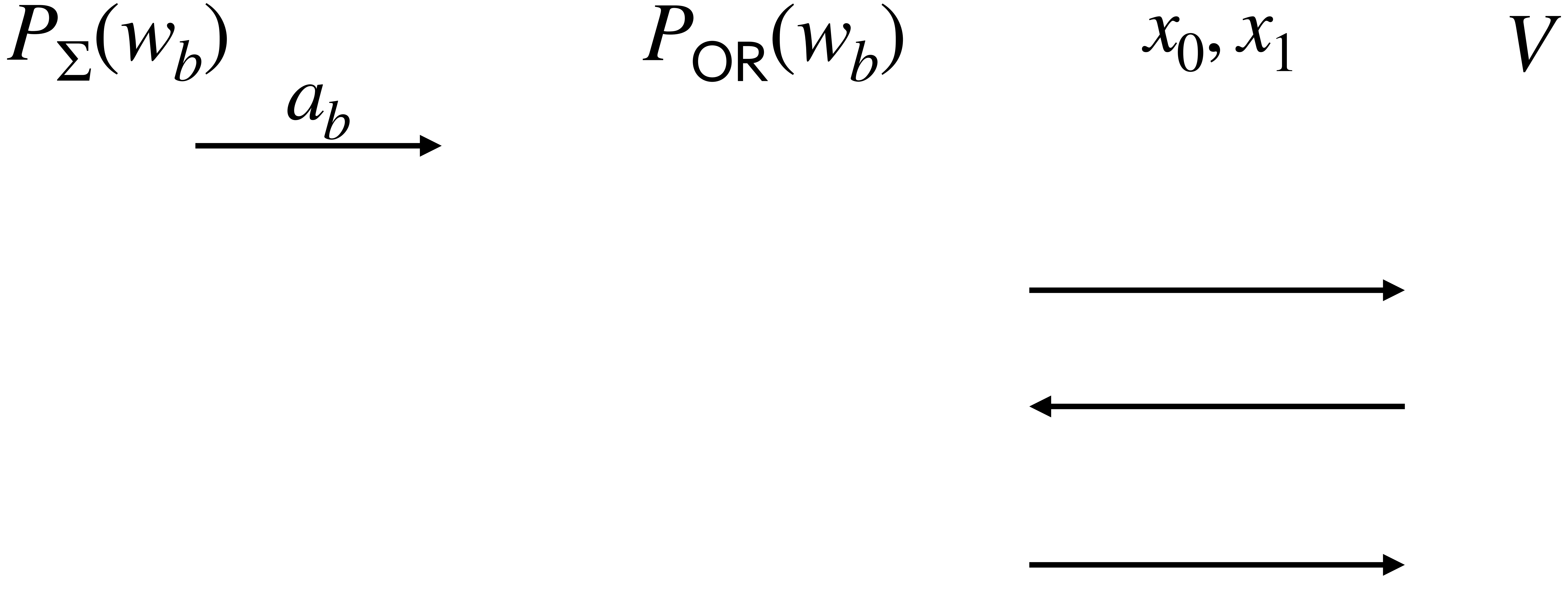
[Cramer Damgård Schoenmakers 94]

$P_{\Sigma}(w_b)$ $P_{\mathsf{OR}}(w_b)$ x_0, x_1 V



Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_{\Sigma}(w_b)$

a_b

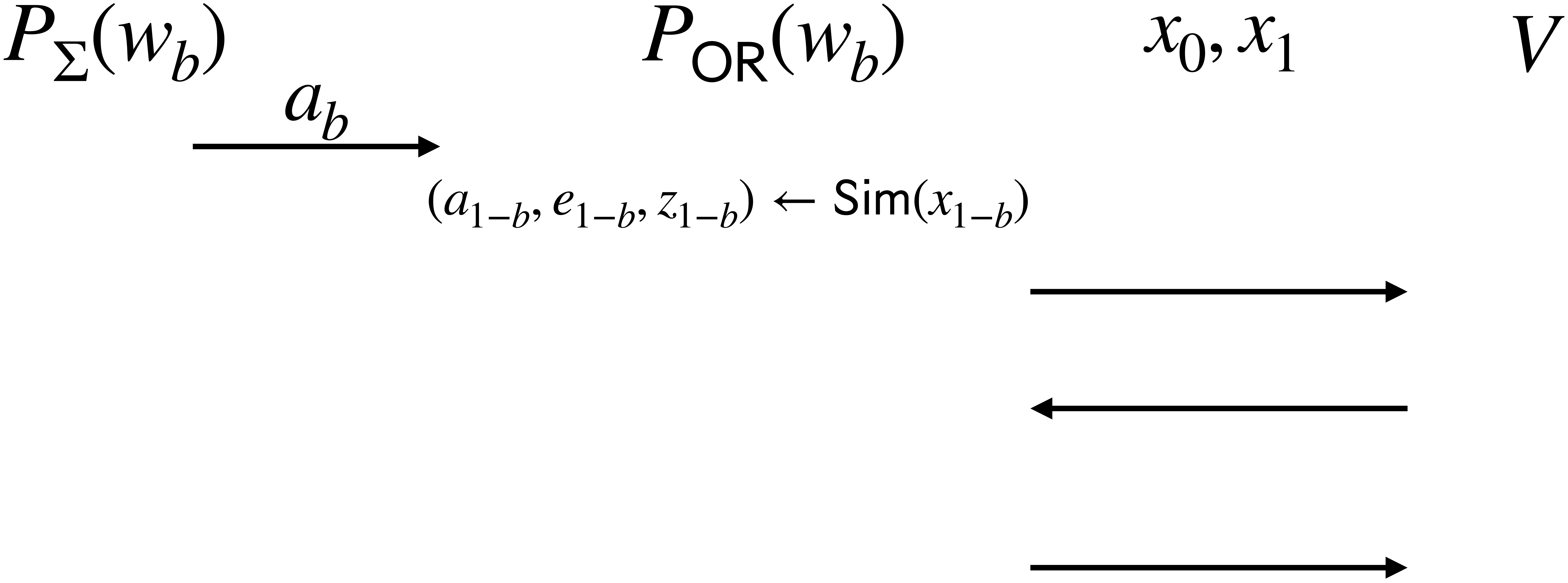
$P_{\mathsf{OR}}(w_b)$

x_0, x_1

V

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



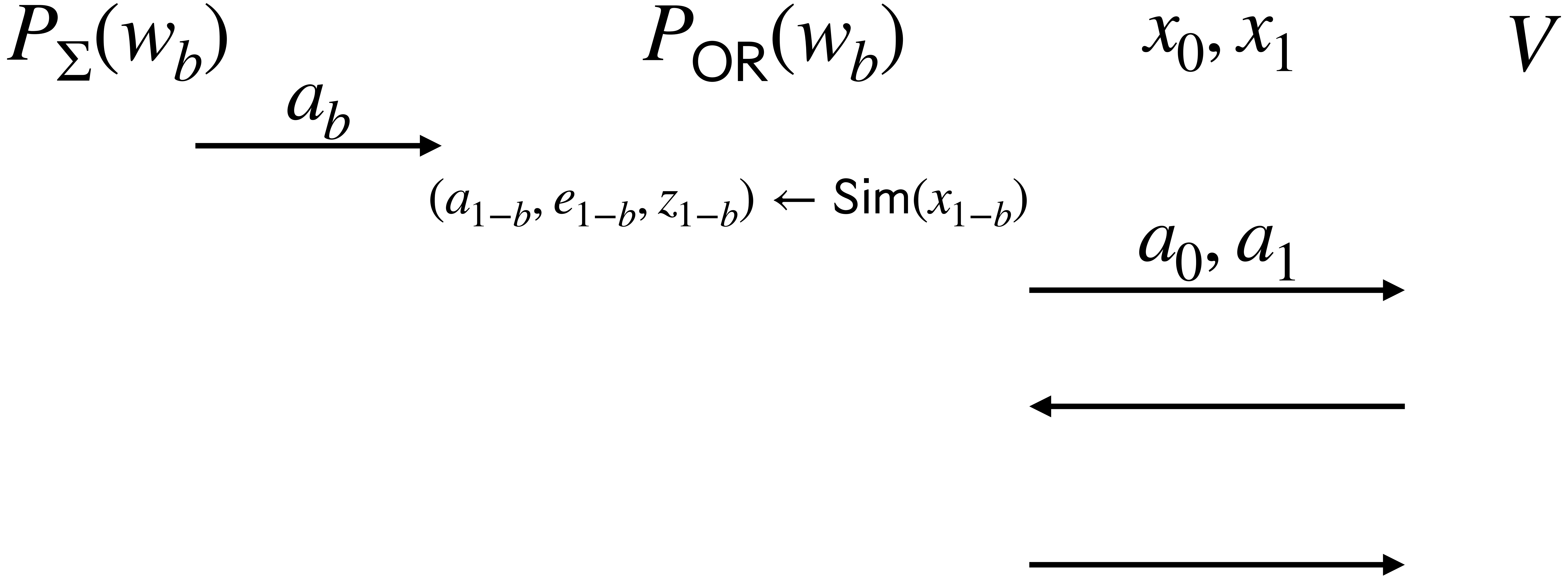
$P_\Sigma(w_b)$ $\quad$ $P_{\mathsf{OR}}(w_b)$ $\quad$ x_0, x_1 $\quad$ V

$\xrightarrow{a_b}$

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_\Sigma(w_b)$

$P_{\mathsf{OR}}(w_b)$

x_0, x_1

V

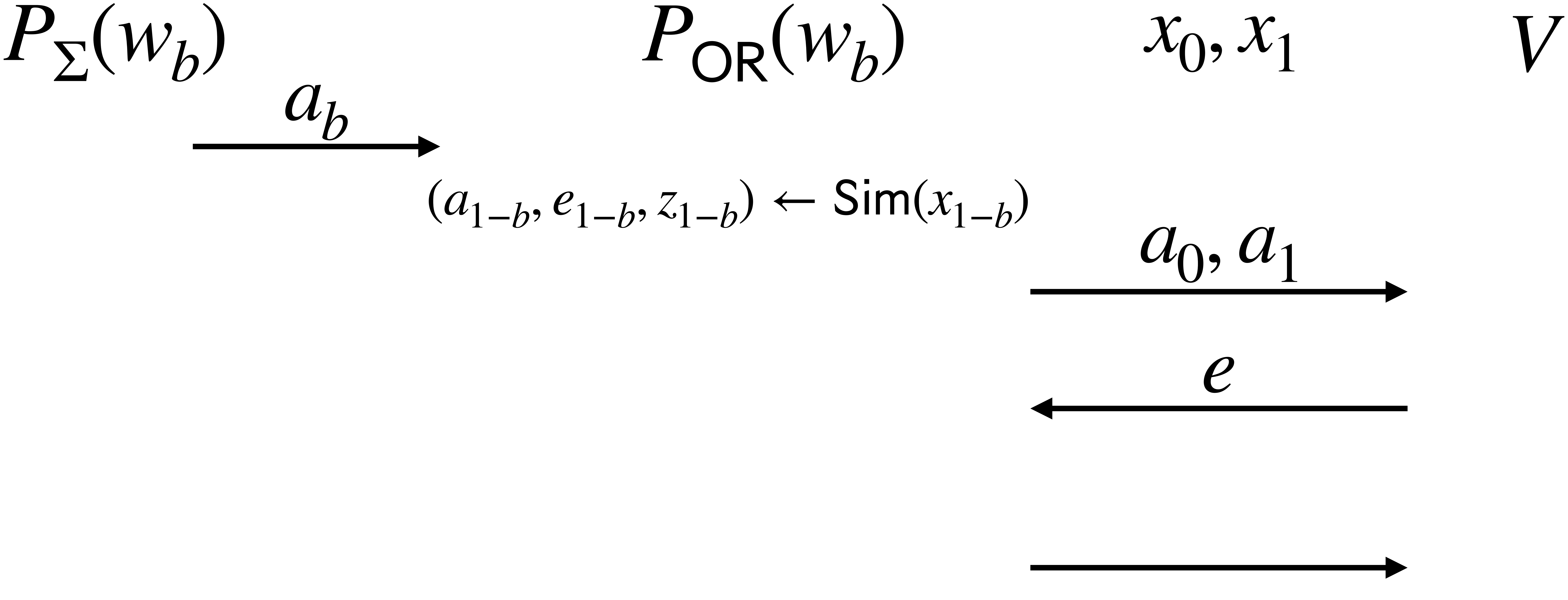
a_b

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

a_0, a_1

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_\Sigma(w_b)$ $\quad$ $P_{\mathsf{OR}}(w_b)$ $\quad$ x_0, x_1 $\quad$ V

$\xrightarrow{a_b}$

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

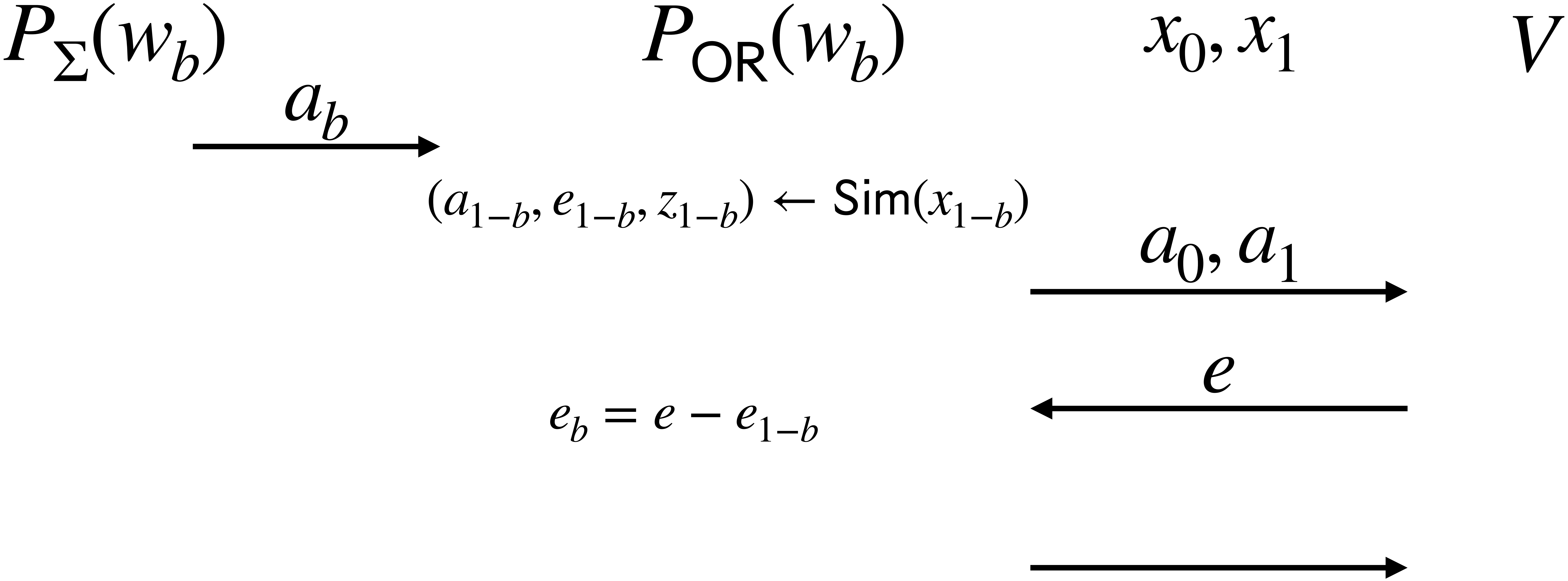
$\xrightarrow{a_0, a_1}$

$\xleftarrow{e}$

$\xrightarrow{\quad}$

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_{\Sigma}(w_b)$ $P_{\mathsf{OR}}(w_b)$ x_0, x_1 V

$\xrightarrow{a_b}$

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

$\xrightarrow{a_0, a_1}$

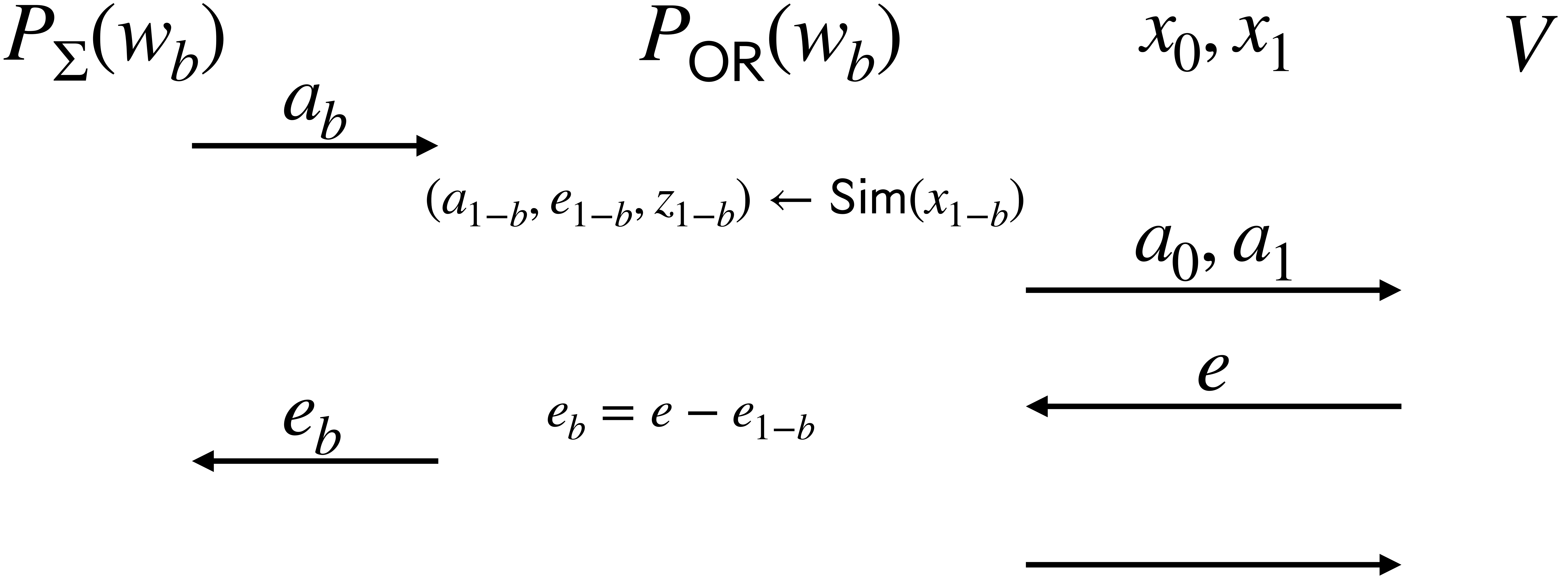
$\xleftarrow{e}$

$e_b = e - e_{1-b}$

$\longrightarrow$

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_\Sigma(w_b)$

$P_{\mathsf{OR}}(w_b)$

x_0, x_1

V

a_b

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

a_0, a_1

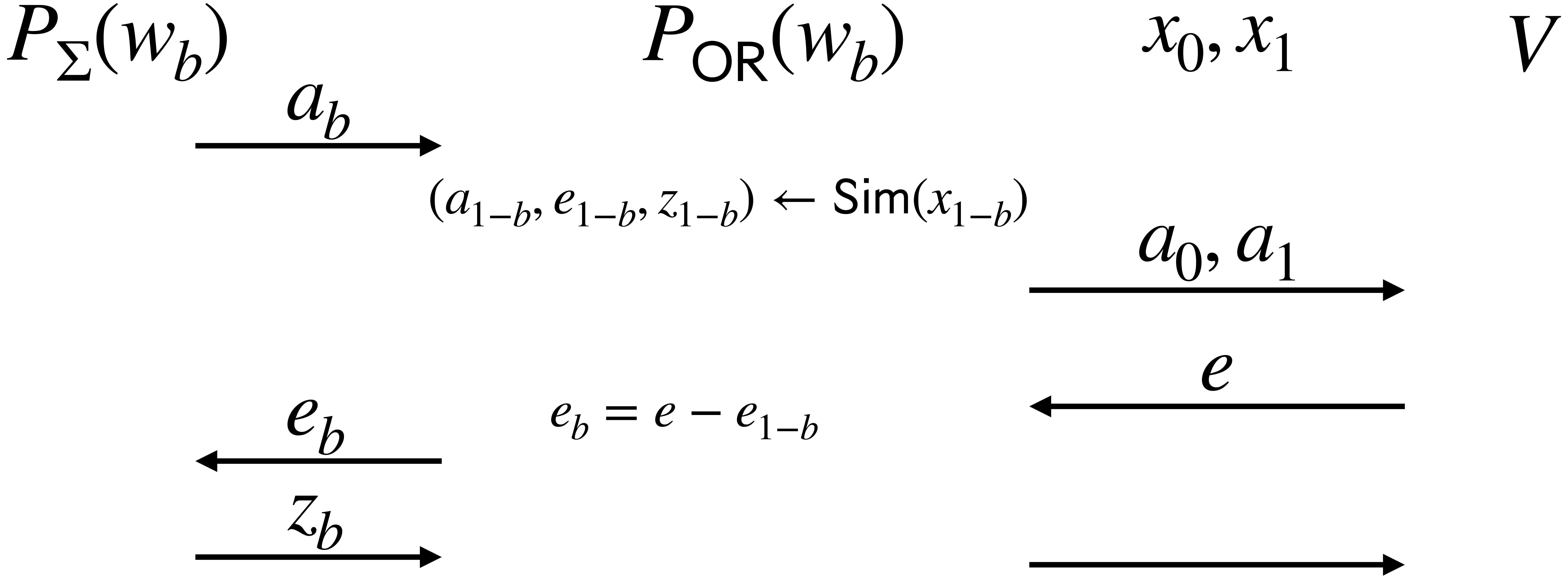
e

e_b

$e_b = e - e_{1-b}$

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_{\Sigma}(w_b)$ $P_{\mathsf{OR}}(w_b)$ x_0, x_1 V

$\xrightarrow{a_b}$

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

$\xrightarrow{a_0, a_1}$

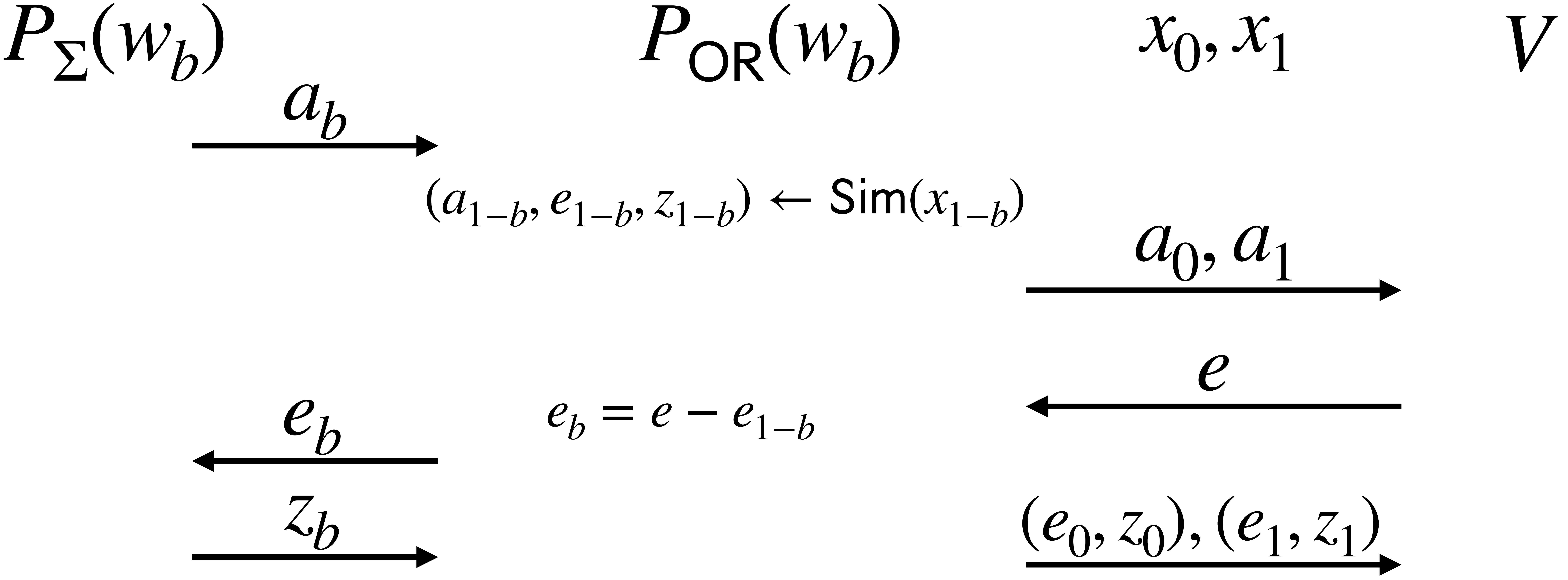
$\xleftarrow{e}$

$\xleftarrow{e_b}$ $e_b = e - e_{1-b}$

$\xrightarrow{z_b}$ $\xrightarrow{}$

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_\Sigma(w_b)$ $\quad P_{\mathsf{OR}}(w_b)$ $\quad x_0, x_1$ $\quad V$

$$P_\Sigma(w_b) \xrightarrow{a_b} P_{\mathsf{OR}}(w_b)$$

$$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$$

$$P_{\mathsf{OR}}(w_b) \xrightarrow{a_0, a_1} V$$

$$P_{\mathsf{OR}}(w_b) \xleftarrow{e} V$$

$$e_b = e - e_{1-b}$$

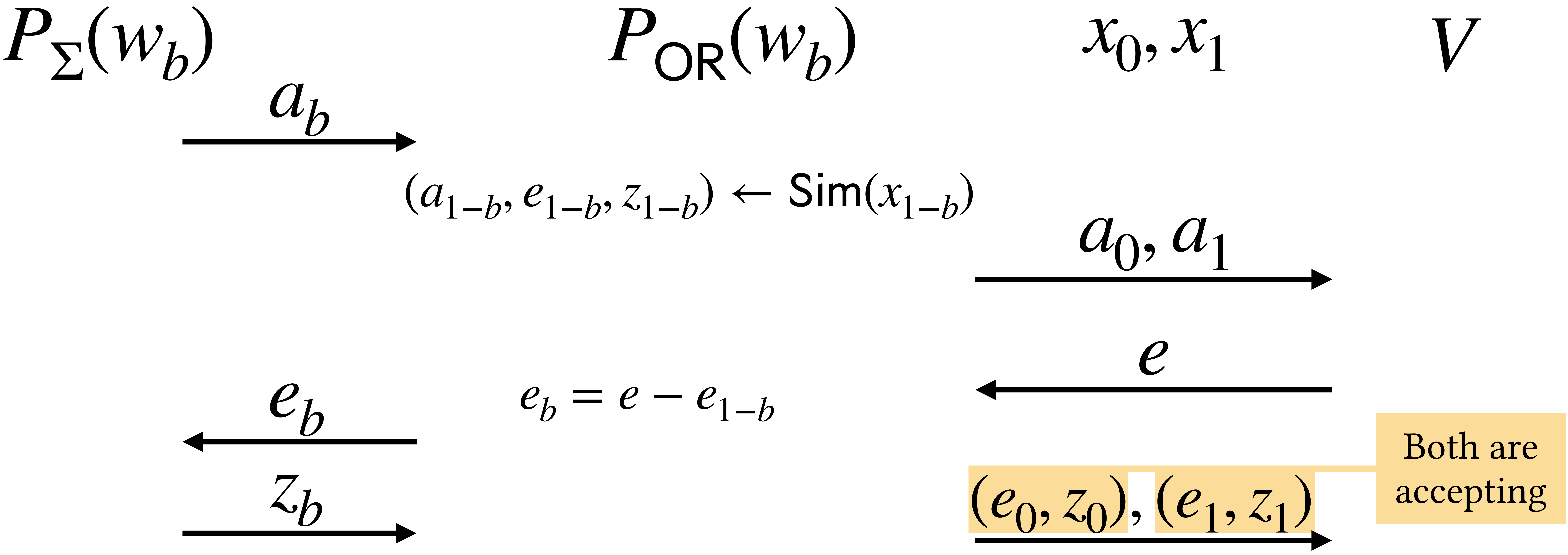
$$P_\Sigma(w_b) \xleftarrow{e_b} P_{\mathsf{OR}}(w_b)$$

$$P_\Sigma(w_b) \xrightarrow{z_b} P_{\mathsf{OR}}(w_b)$$

$$P_{\mathsf{OR}}(w_b) \xrightarrow{(e_0, z_0), (e_1, z_1)} V$$

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_{\Sigma}(w_b)$

$P_{\mathsf{OR}}(w_b)$

x_0, x_1

V

a_b

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

a_0, a_1

e

e_b

$e_b = e - e_{1-b}$

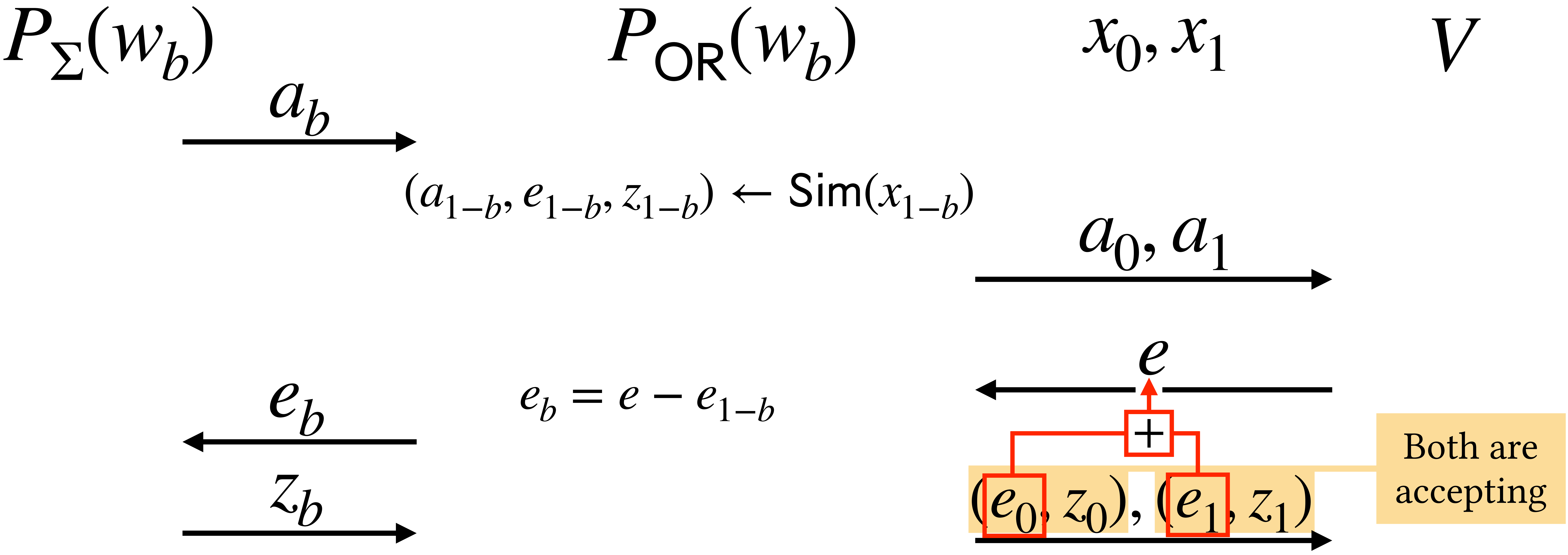
z_b

$(e_0, z_0), (e_1, z_1)$

Both are accepting

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_\Sigma(w_b)$ $\qquad$ $P_{\mathsf{OR}}(w_b)$ $\qquad$ x_0, x_1 $\qquad$ V

$$P_\Sigma(w_b) \xrightarrow{a_b} P_{\mathsf{OR}}(w_b)$$

$$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$$

$$P_{\mathsf{OR}}(w_b) \xrightarrow{a_0, a_1} V$$

$$P_{\mathsf{OR}}(w_b) \xleftarrow{e} V$$

$$P_\Sigma(w_b) \xleftarrow{e_b} P_{\mathsf{OR}}(w_b)$$

$$e_b = e - e_{1-b}$$

$$P_\Sigma(w_b) \xrightarrow{z_b} P_{\mathsf{OR}}(w_b)$$

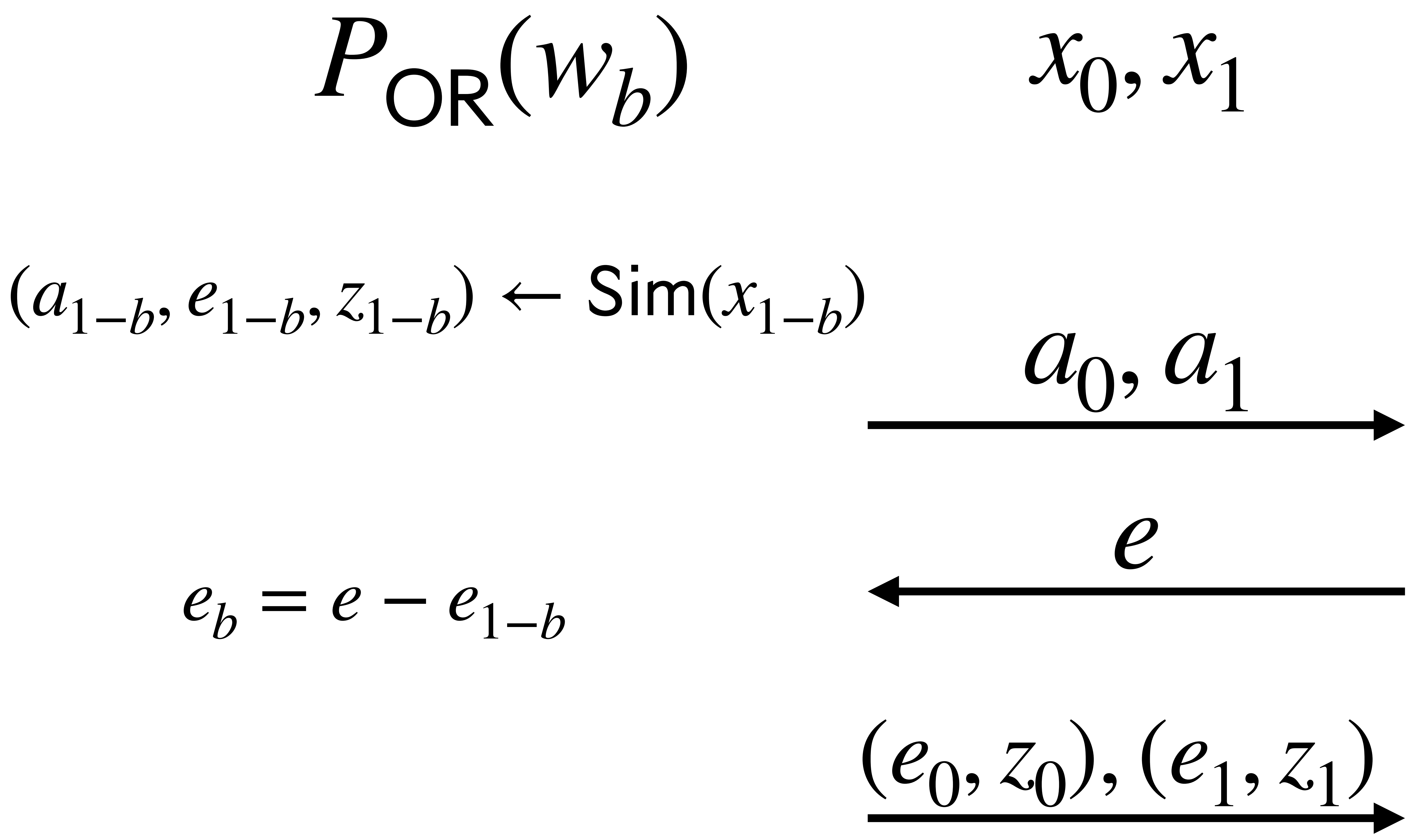
$$P_{\mathsf{OR}}(w_b) \xrightarrow{(\boxed{e_0}, z_0), (\boxed{e_1}, z_1)} V$$

$$e = e_0 + e_1$$

Both are accepting

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_{\mathsf{OR}}(w_b)$ x_0, x_1 V

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

$\xrightarrow{a_0, a_1}$

$\xleftarrow{e}$

$e_b = e - e_{1-b}$

$\xrightarrow{(e_0, z_0), (e_1, z_1)}$

Recall: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ violates unique responses

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]

$P_{\mathsf{OR}}(w_b)$ $\qquad$ x_0, x_1 $\qquad$ V

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

$$\xrightarrow{\quad a_0, a_1 \quad}$$

$$\xleftarrow{\quad e \quad}$$

$e_b = e - e_{1-b}$

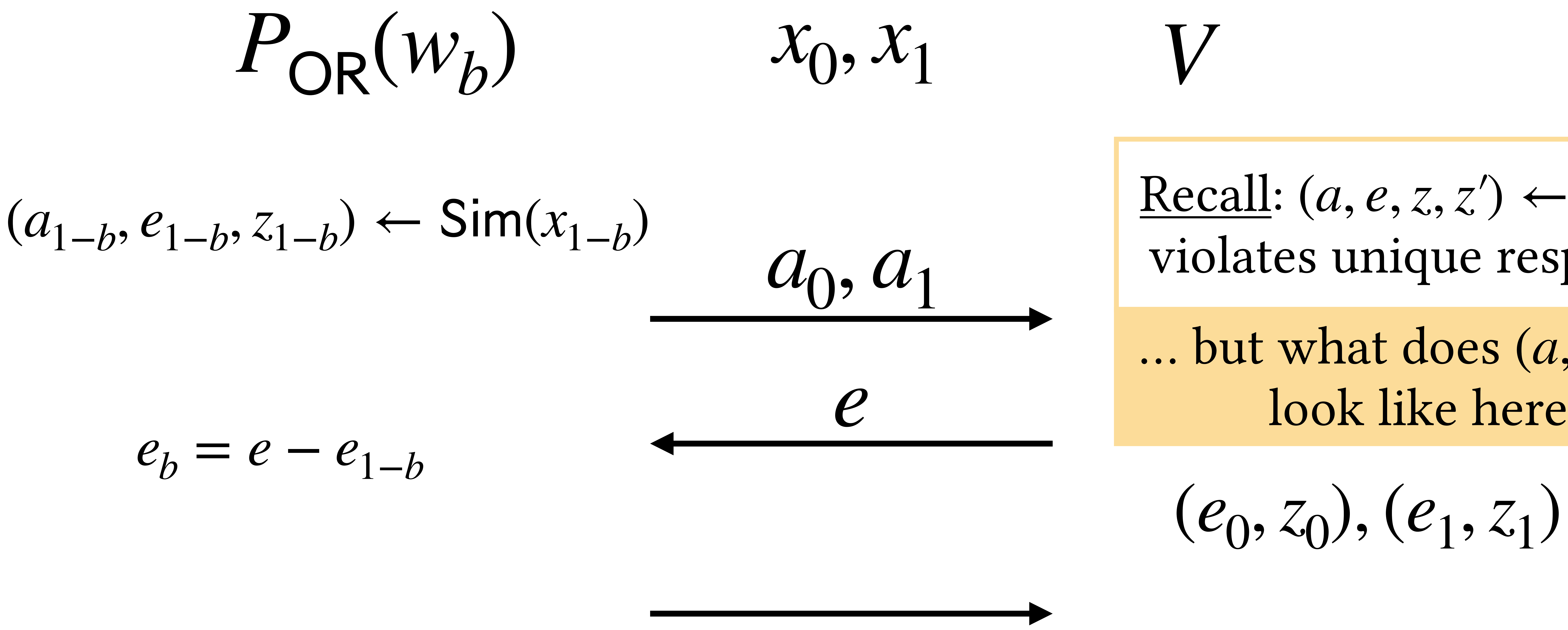
$$\xrightarrow{\quad (e_0, z_0), (e_1, z_1) \quad}$$

Recall: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ violates unique responses

… but what does (a, e, z, z') look like here?

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_{\mathsf{OR}}(w_b)$ $\quad$ x_0, x_1 $\quad$ V

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

a_0, a_1

Recall: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ violates unique responses

… but what does (a, e, z, z') look like here?

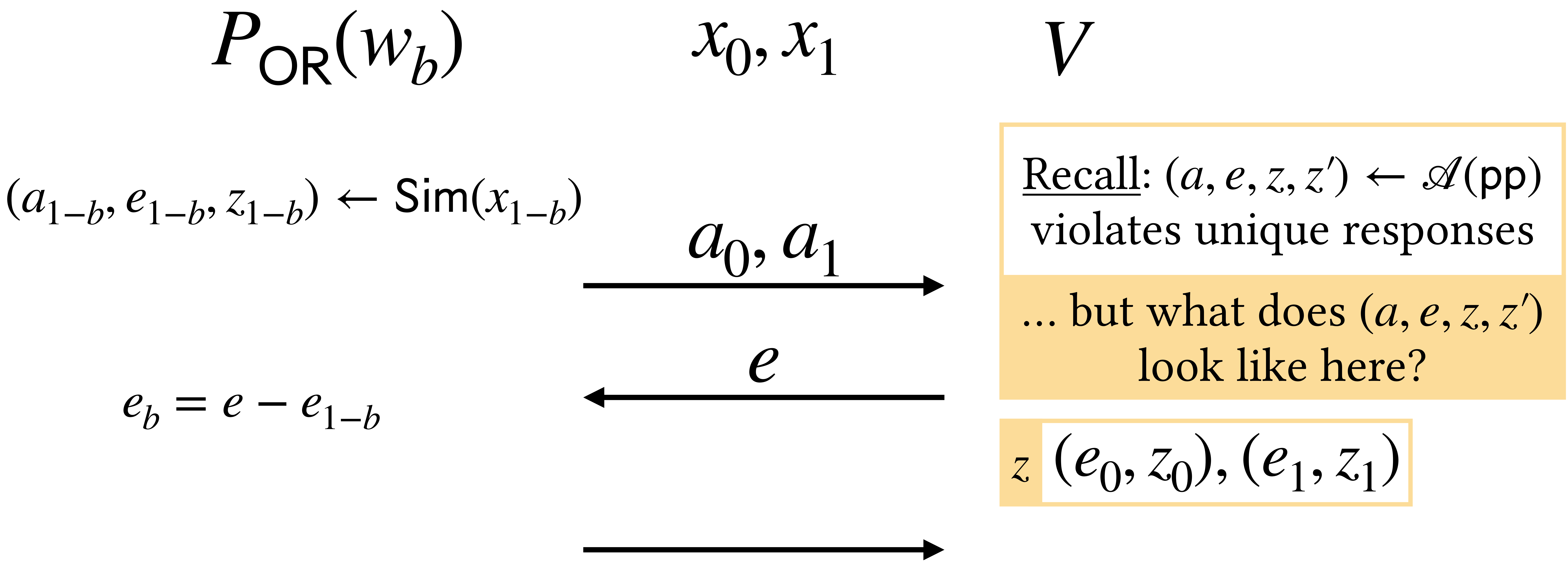
e

$e_b = e - e_{1-b}$

$(e_0, z_0), (e_1, z_1)$

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



$P_{\mathsf{OR}}(w_b)$

x_0, x_1

V

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

$\xrightarrow{a_0, a_1}$

<u>Recall</u>: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ violates unique responses

... but what does (a, e, z, z') look like here?

$\xleftarrow{e}$

$e_b = e - e_{1-b}$

z $(e_0, z_0), (e_1, z_1)$

$\longrightarrow$

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]

$P_{\mathsf{OR}}(w_b)$ $\quad x_0, x_1 \quad$ V

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

$$\xrightarrow{\quad a_0, a_1 \quad}$$

$$\xleftarrow{\quad e \quad}$$

$e_b = e - e_{1-b}$

$$\xrightarrow{(e'_0, z'_0), (e'_1, z'_1) \quad z'}$$

$z \quad (e_0, z_0), (e_1, z_1)$

Recall: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ violates unique responses

… but what does (a, e, z, z') look like here?

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]

$P_{\mathsf{OR}}(w_b)$ $\qquad$ x_0, x_1 $\qquad$ V

$(a_{1-b}, e_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

$$\xrightarrow{\quad a_0, a_1 \quad}$$

$$\xleftarrow{\quad e \quad}$$

$e_b = e - e_{1-b}$

$$\xrightarrow{\quad (e_0', z_0'), (e_1', z_1') \quad} \; z'$$

Recall: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ violates unique responses

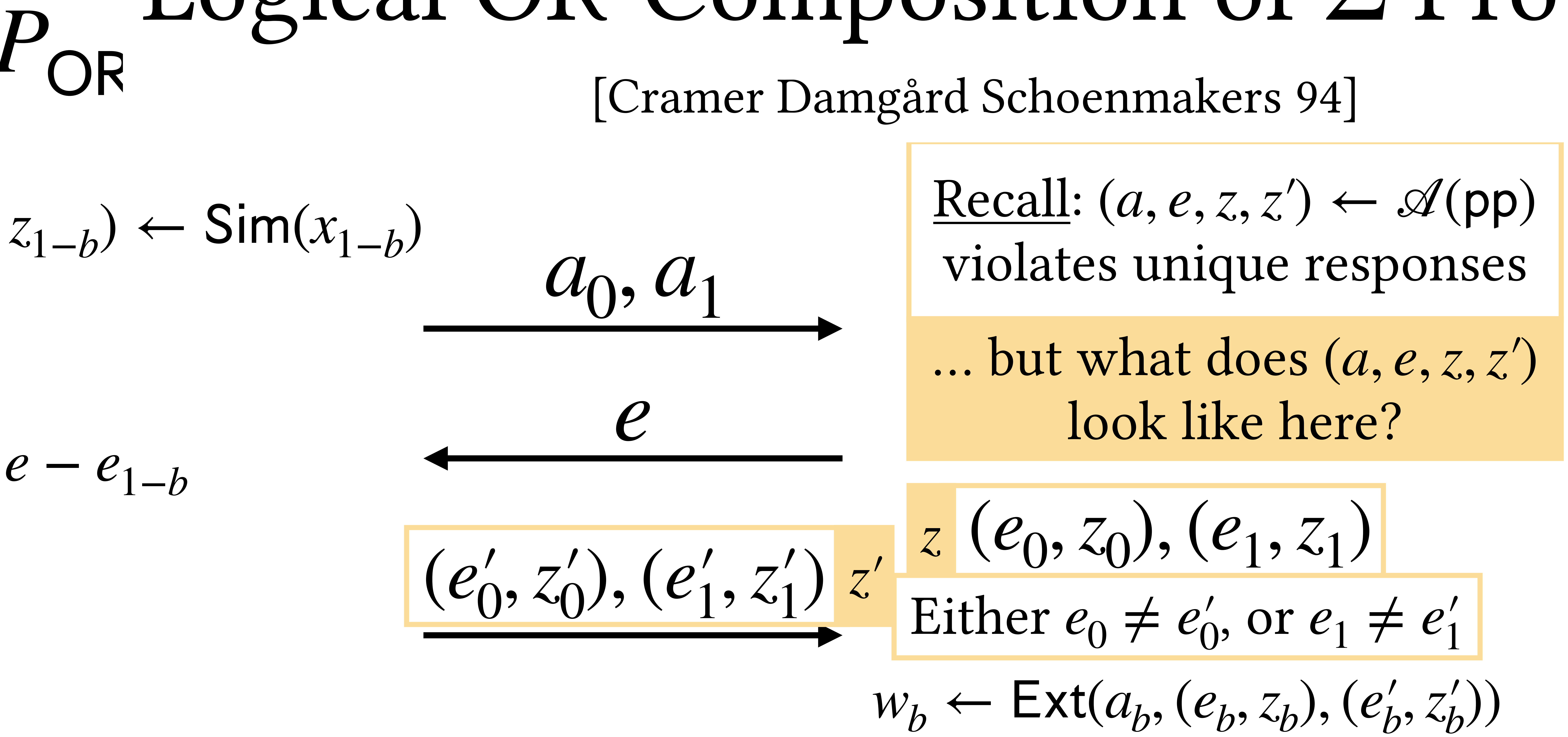
... but what does (a, e, z, z') look like here?

z $\quad (e_0, z_0), (e_1, z_1)$

Either $e_0 \neq e_0'$, or $e_1 \neq e_1'$

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]



P_{OR}

$z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

a_0, a_1

e

$e - e_{1-b}$

$(e_0', z_0'), (e_1', z_1')$ z'

<u>Recall</u>: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ violates unique responses

... but what does (a, e, z, z') look like here?

z $(e_0, z_0), (e_1, z_1)$

Either $e_0 \neq e_0'$, or $e_1 \neq e_1'$

$w_b \leftarrow \mathsf{Ext}(a_b, (e_b, z_b), (e_b', z_b'))$

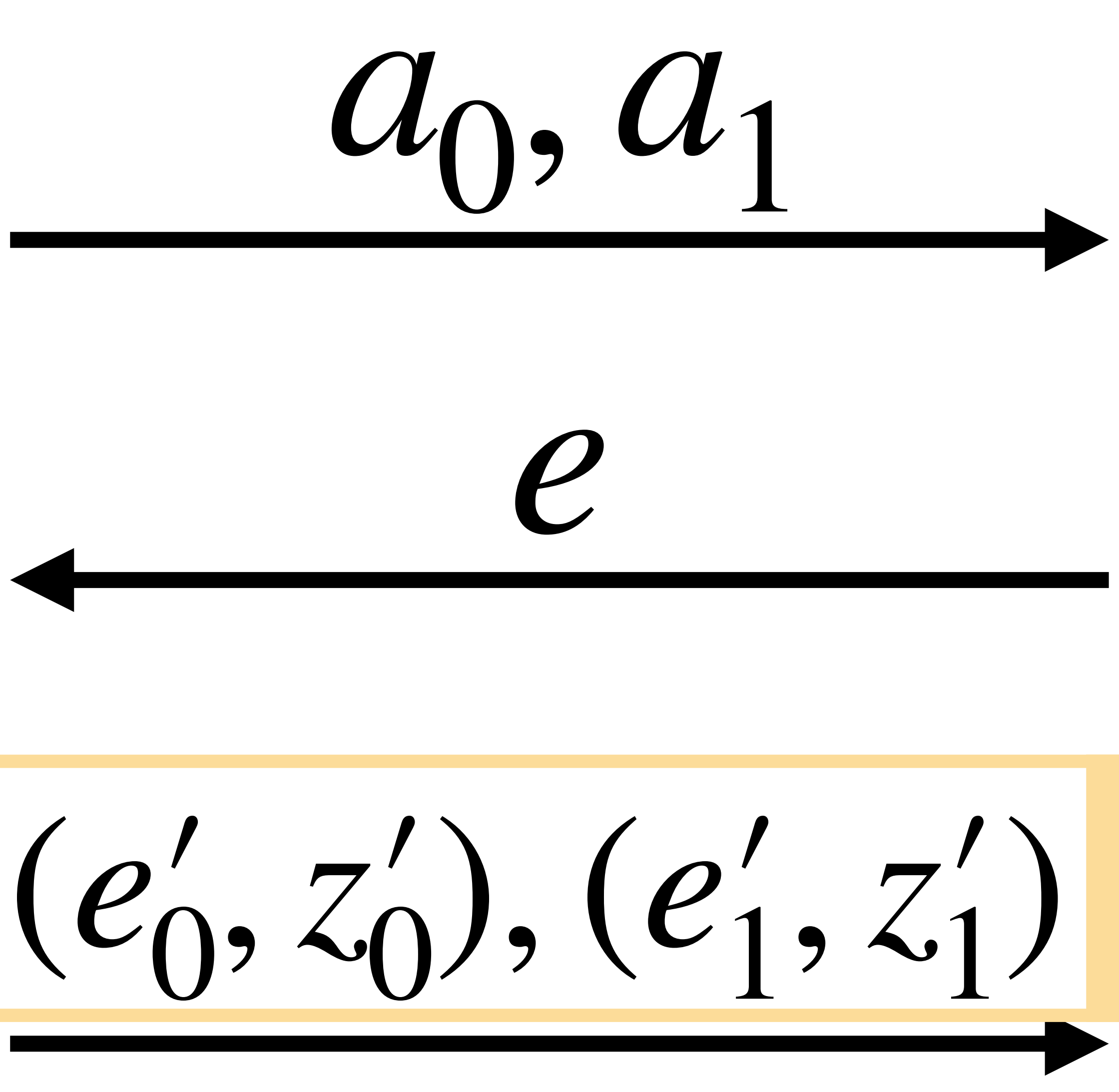
P_{OR}

Logical OR-Composition of Σ Protocols

[Cramer Damgård Schoenmakers 94]

$z_{1-b}) \leftarrow \mathsf{Sim}(x_{1-b})$

$e - e_{1-b}$



a_0, a_1

e

$(e'_0, z'_0), (e'_1, z'_1)$ z'

Recall: $(a, e, z, z') \leftarrow \mathscr{A}(\mathsf{pp})$ violates unique responses

... but what does (a, e, z, z') look like here?

z $(e_0, z_0), (e_1, z_1)$

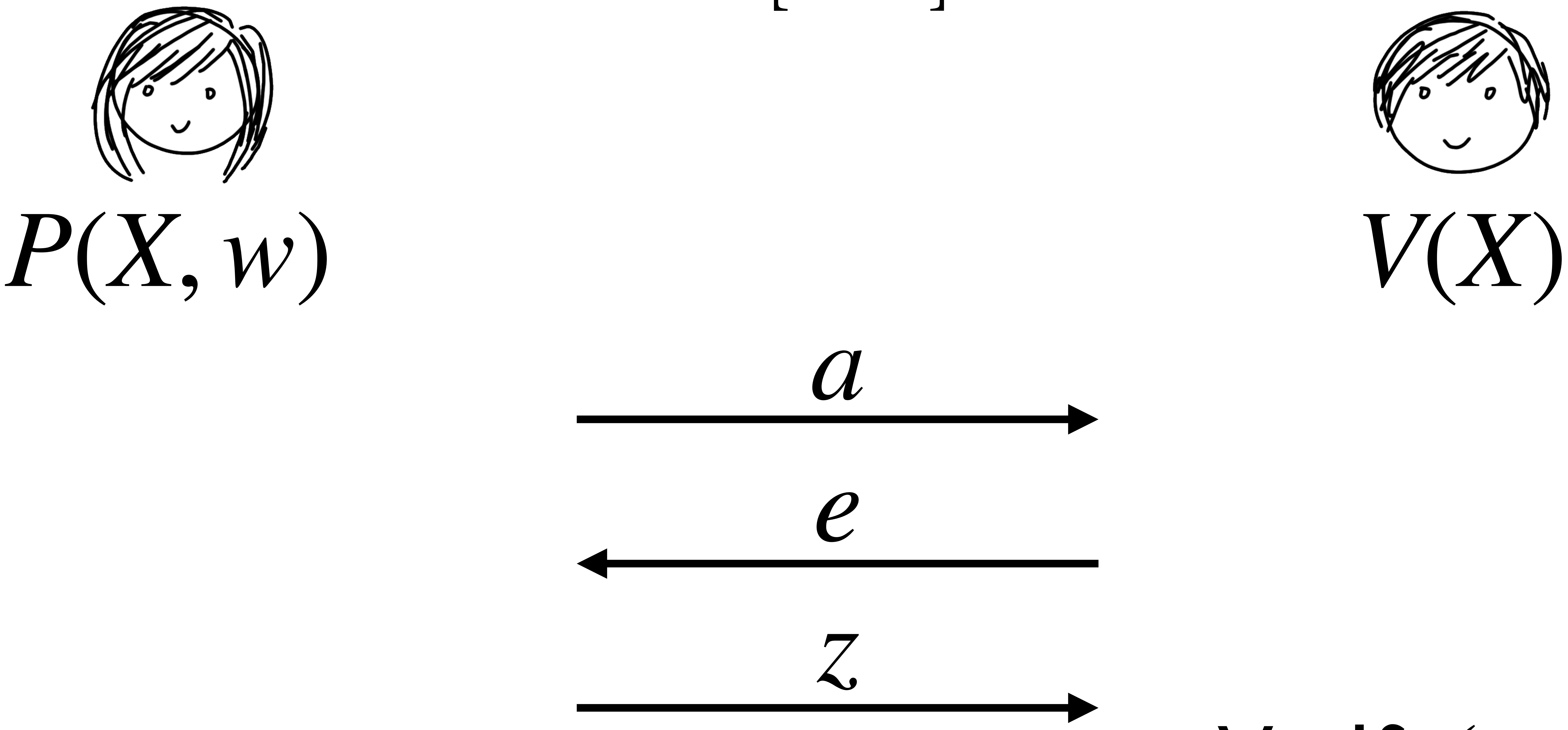
Either $e_0 \neq e'_0$, or $e_1 \neq e'_1$

$w_b \leftarrow \mathsf{Ext}(a_b, (e_b, z_b), (e'_b, z'_b))$

Quasi-unique responses not *strictly* necessary for extraction (folklore)

Tightening Conditions for Extraction

[Ks 22]



$P(X, w)$

$V(X)$

$\xrightarrow{a}$

$\xleftarrow{e}$

$\xrightarrow{z}$

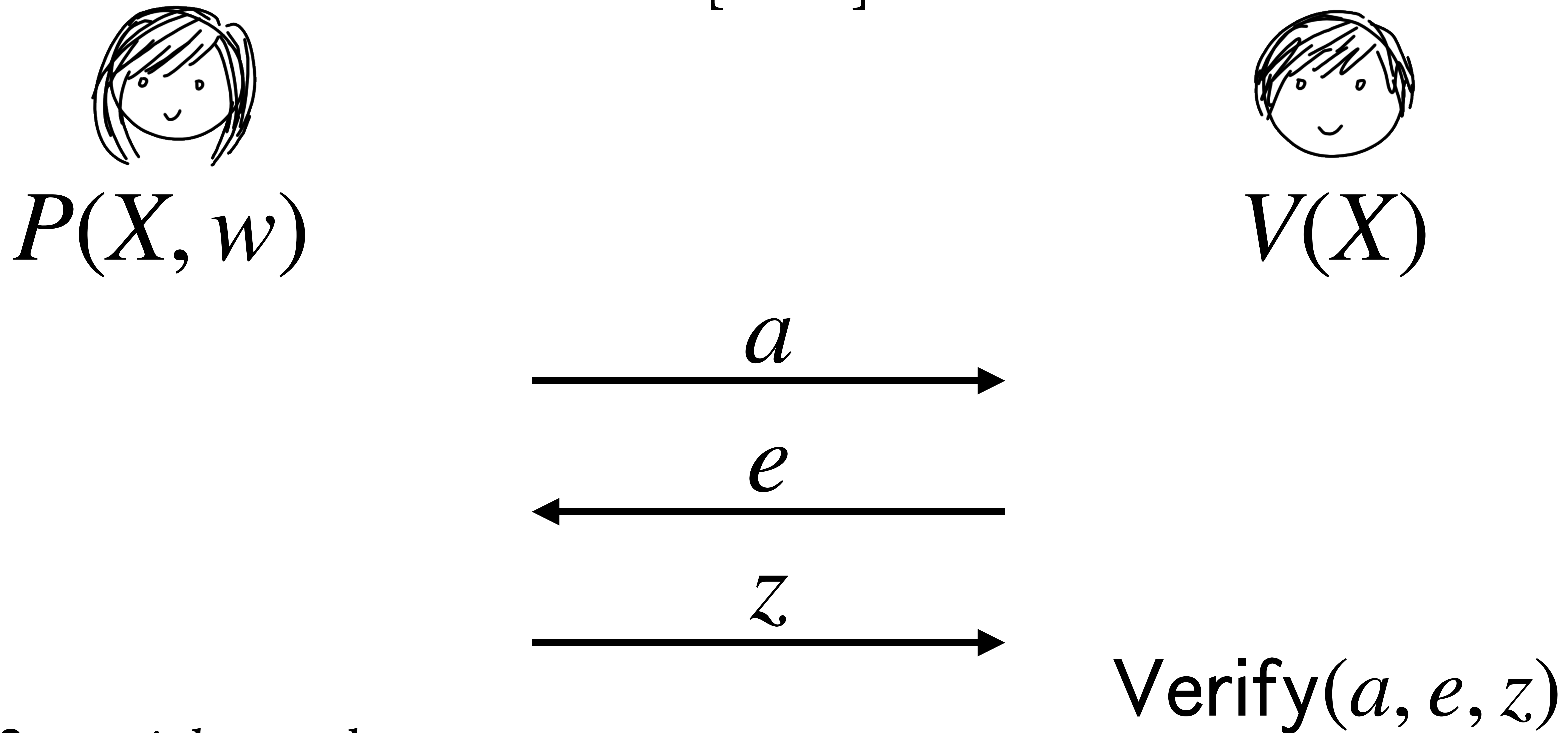
$\mathsf{Verify}(a, e, z)$

2-special soundness:

$$w \leftarrow \mathsf{Ext}(X, a, (e_1, z_1), (e_2, z_2)) \text{ such that } R(X, w) = 1$$

Tightening Conditions for Extraction

[Ks 22]



$P(X, w)$

$V(X)$

$\xrightarrow{a}$

$\xleftarrow{e}$

$\xrightarrow{z}$

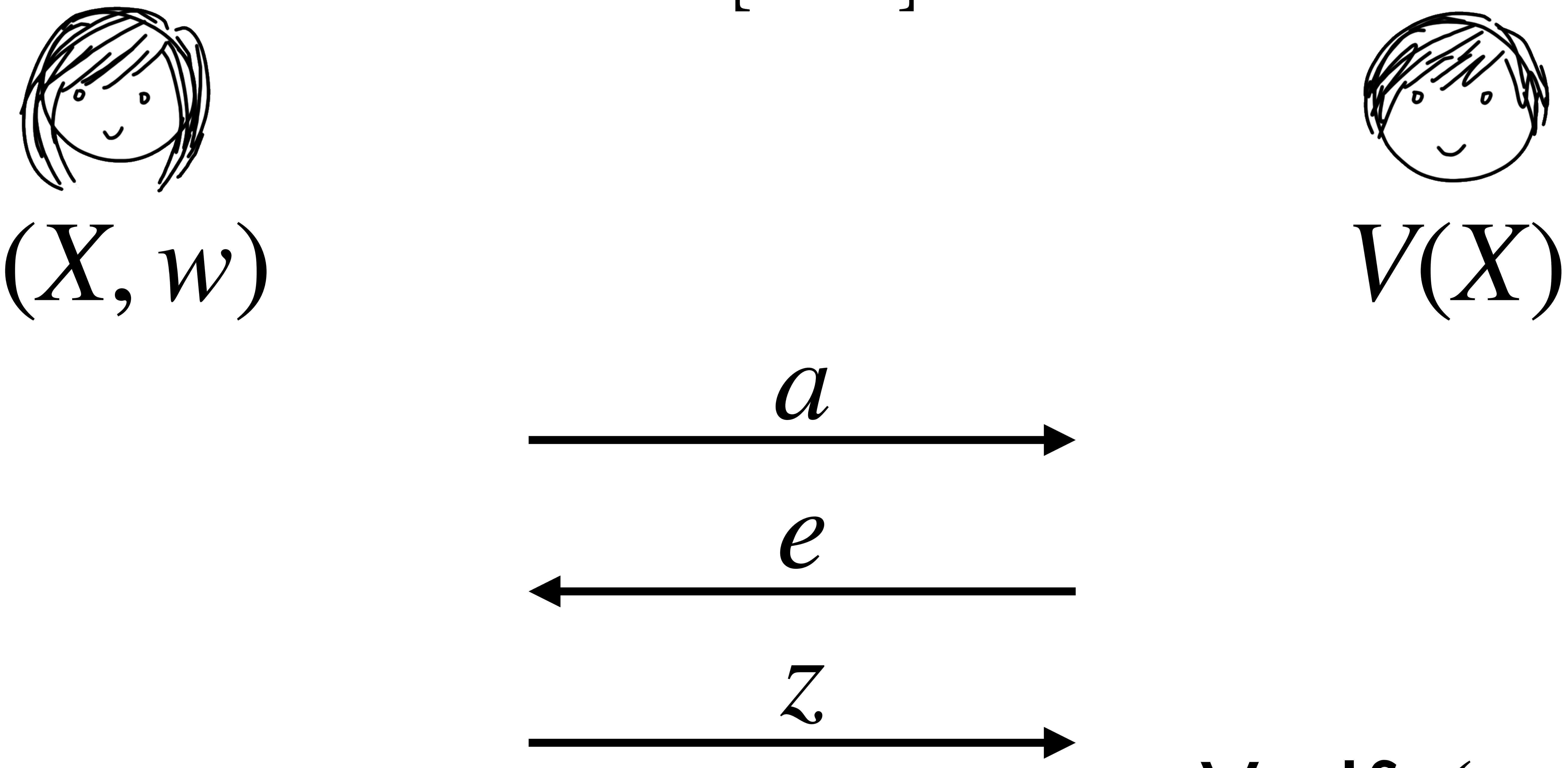
$\mathsf{Verify}(a, e, z)$

Strong 2-special soundness:

$$w \leftarrow \mathsf{Ext}(X, a, (e_1, z_1), (e_2, z_2)) \text{ such that } R(X, w) = 1$$

Tightening Conditions for Extraction

[Ks 22]



$P(X, w)$

$V(X)$

a

e

z

$\mathsf{Verify}(a, e, z)$

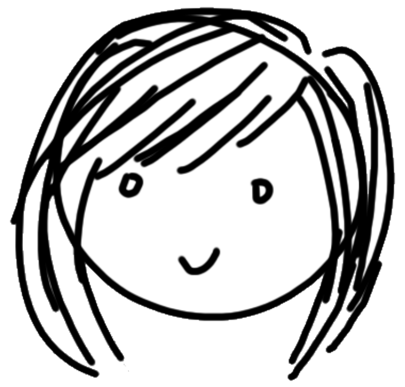
Strong 2-special soundness:

$$w \leftarrow \mathsf{Ext}(X, a, (e_1, z_1), (e_2, z_2)) \text{ such that } R(X, w) = 1$$

$$e_1 \neq e_2 \;\text{ OR }\; z_1 \neq z_2$$

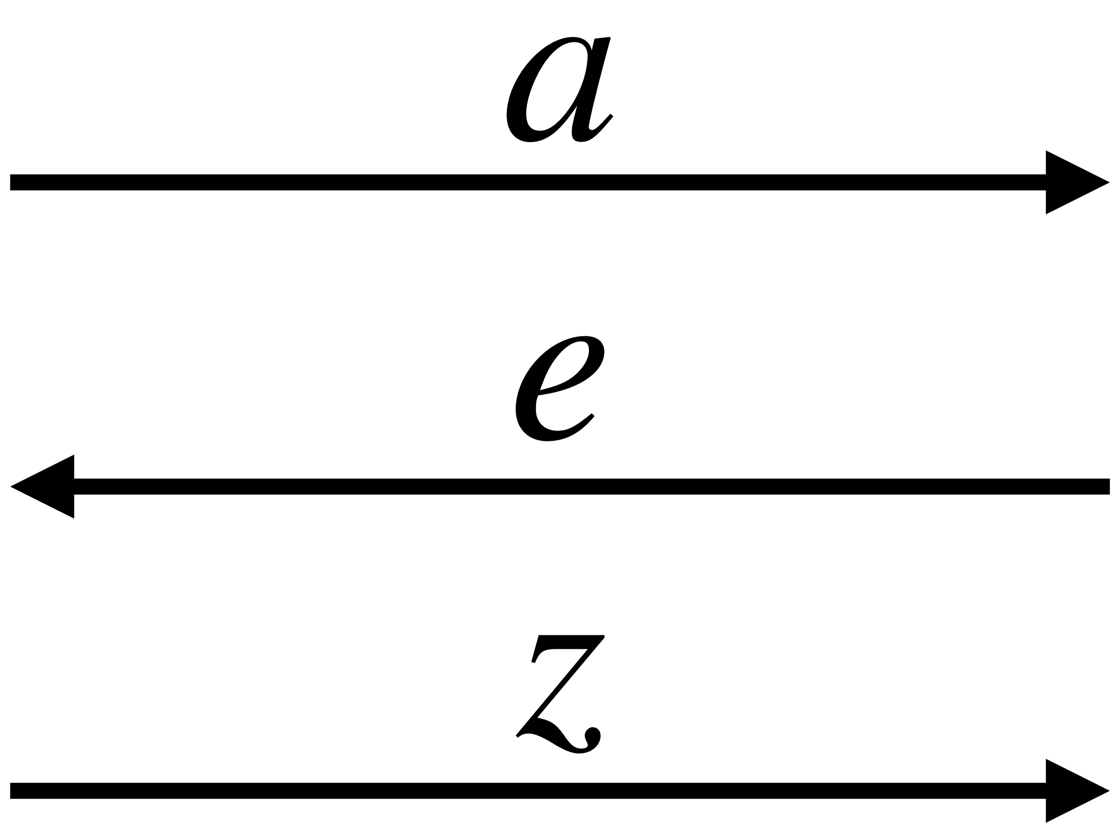
Tightening Conditions for Extraction

[Ks 22]



$P(X, w)$

$V(X)$



a

e

z

$\mathsf{Verify}(a, e, z)$

Strong 2-special soundness:

$$w \leftarrow \mathsf{Ext}(X, a, (e_1, z_1), (e_2, z_2)) \text{ such that } R(X, w) = 1$$

$$e_1 \neq e_2 \ \text{ OR } \ z_1 \neq z_2$$

...are we done?