Sometimes You Can't Distribute Random-Oracle-Based Proofs

¯_(ツ)_/¯

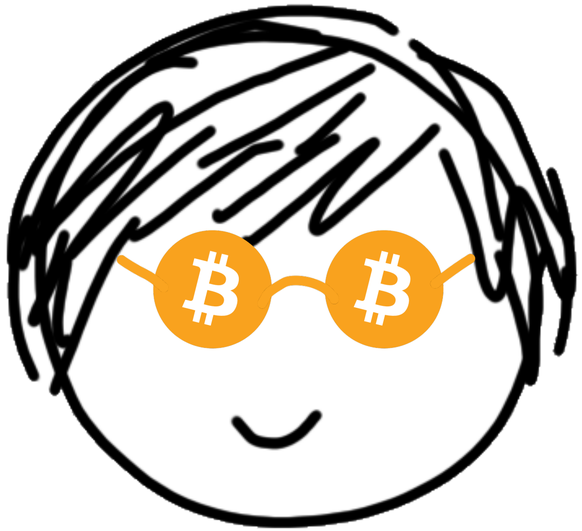
Jack Doerner Yashvanth Kondi Leah Namisa Rosenbloom

TECHNION Israel Institute of Technology

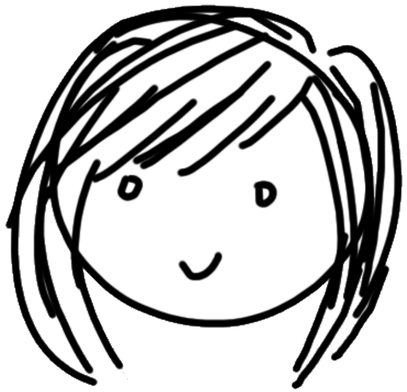
AARHUS UNIVERSITY

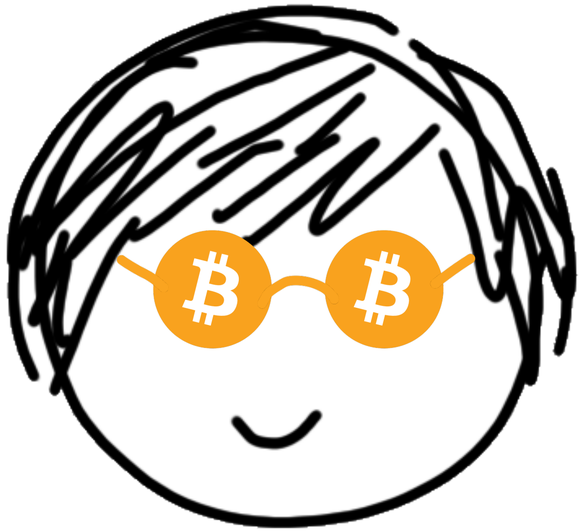
BROWN

Ballad of ₿itcoin Bob

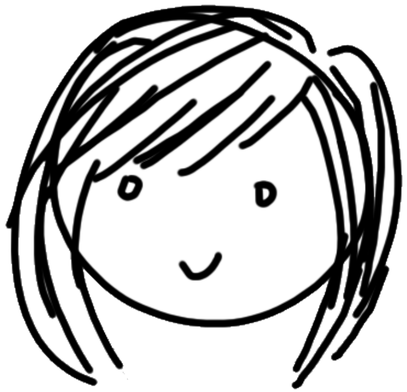


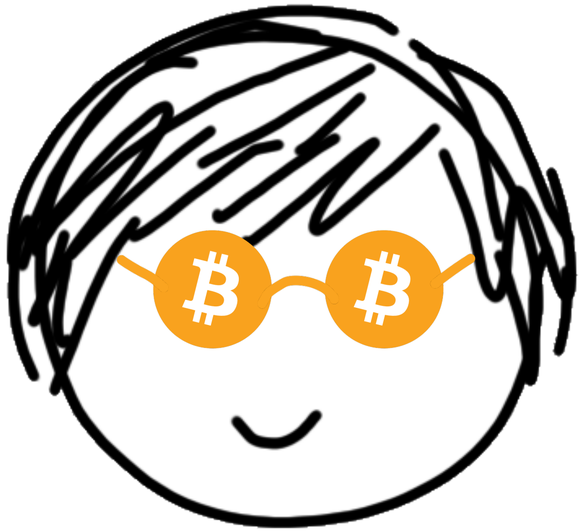
Ballad of ₿itcoin Bob



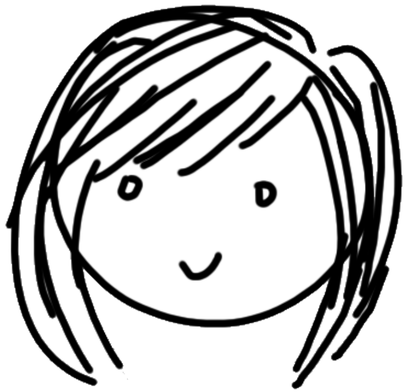


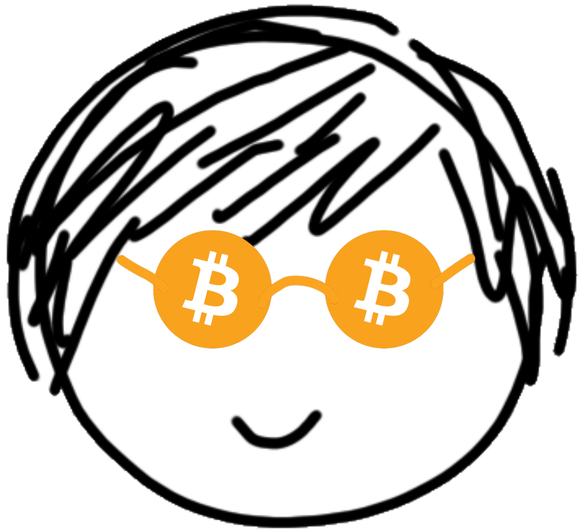
Ballad of ₿itcoin Bob

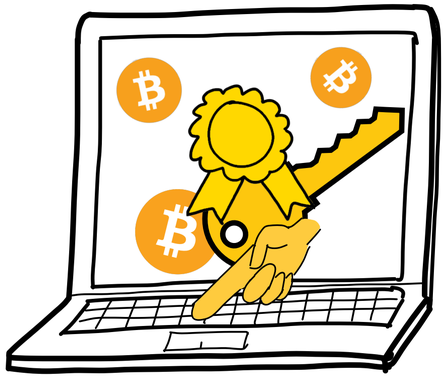




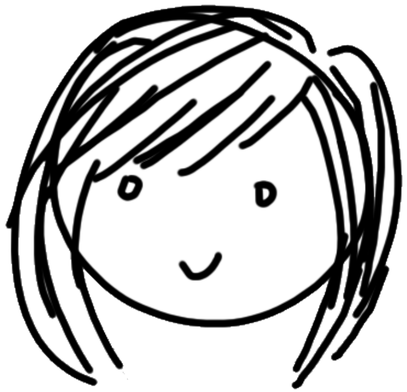
Ballad of ₿itcoin Bob

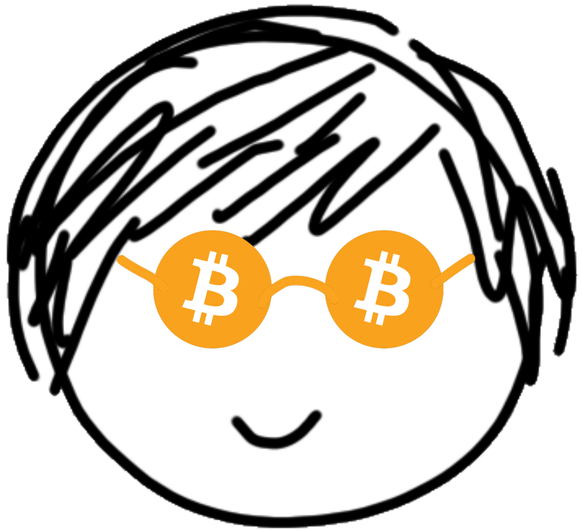




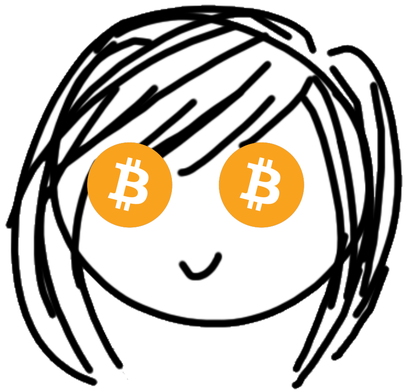


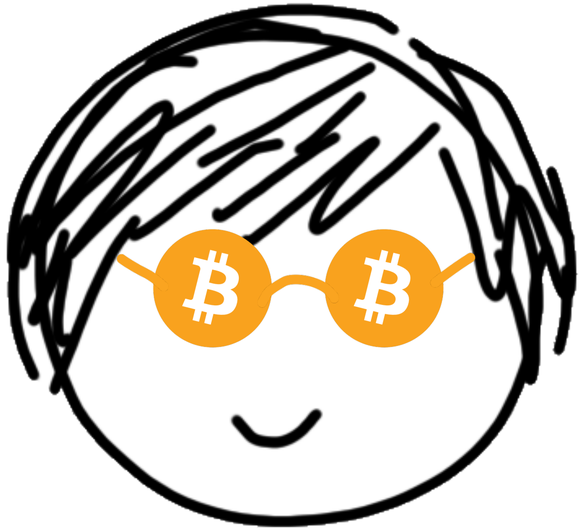
Ballad of ₿itcoin Bob



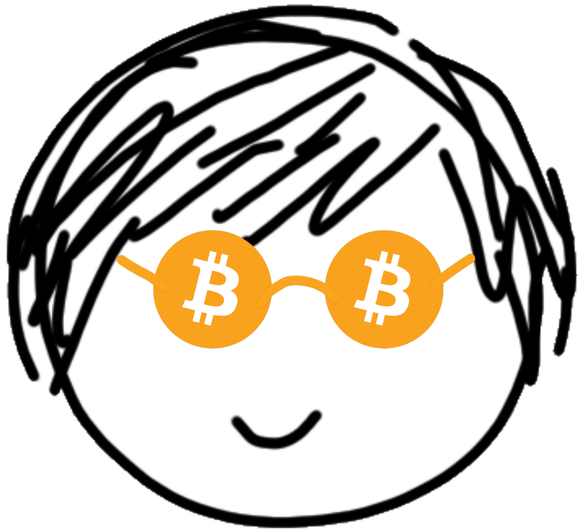


Ballad of ₿itcoin Bob

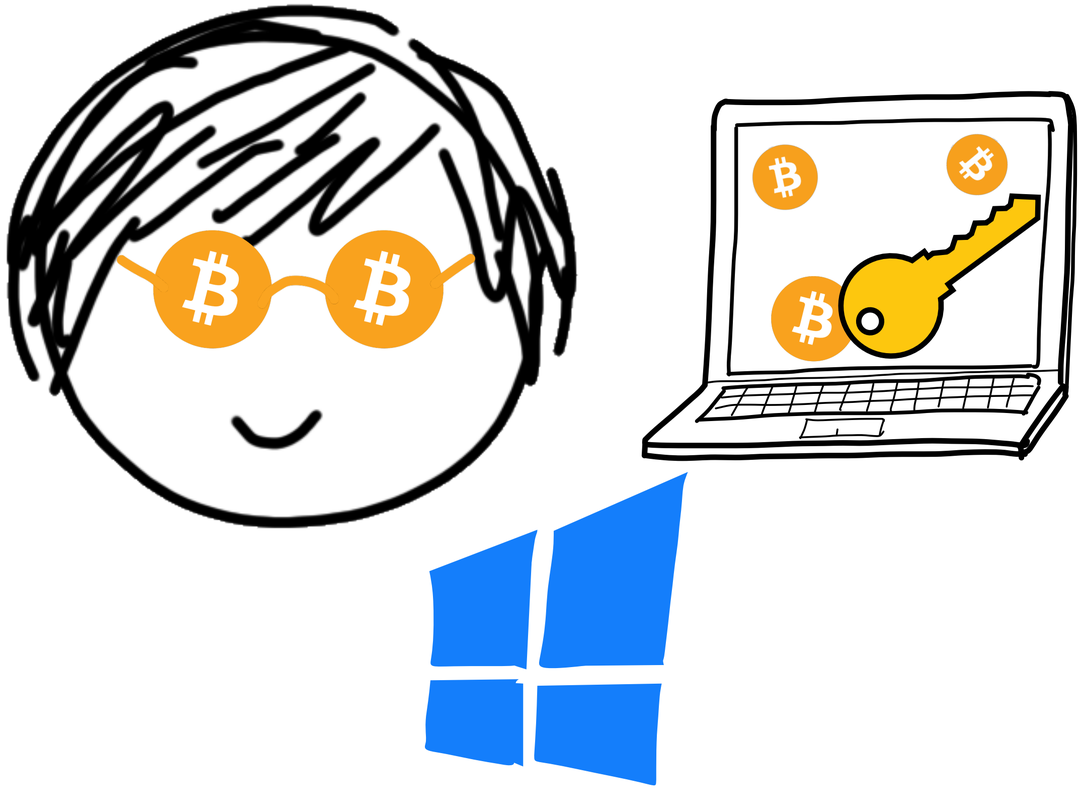




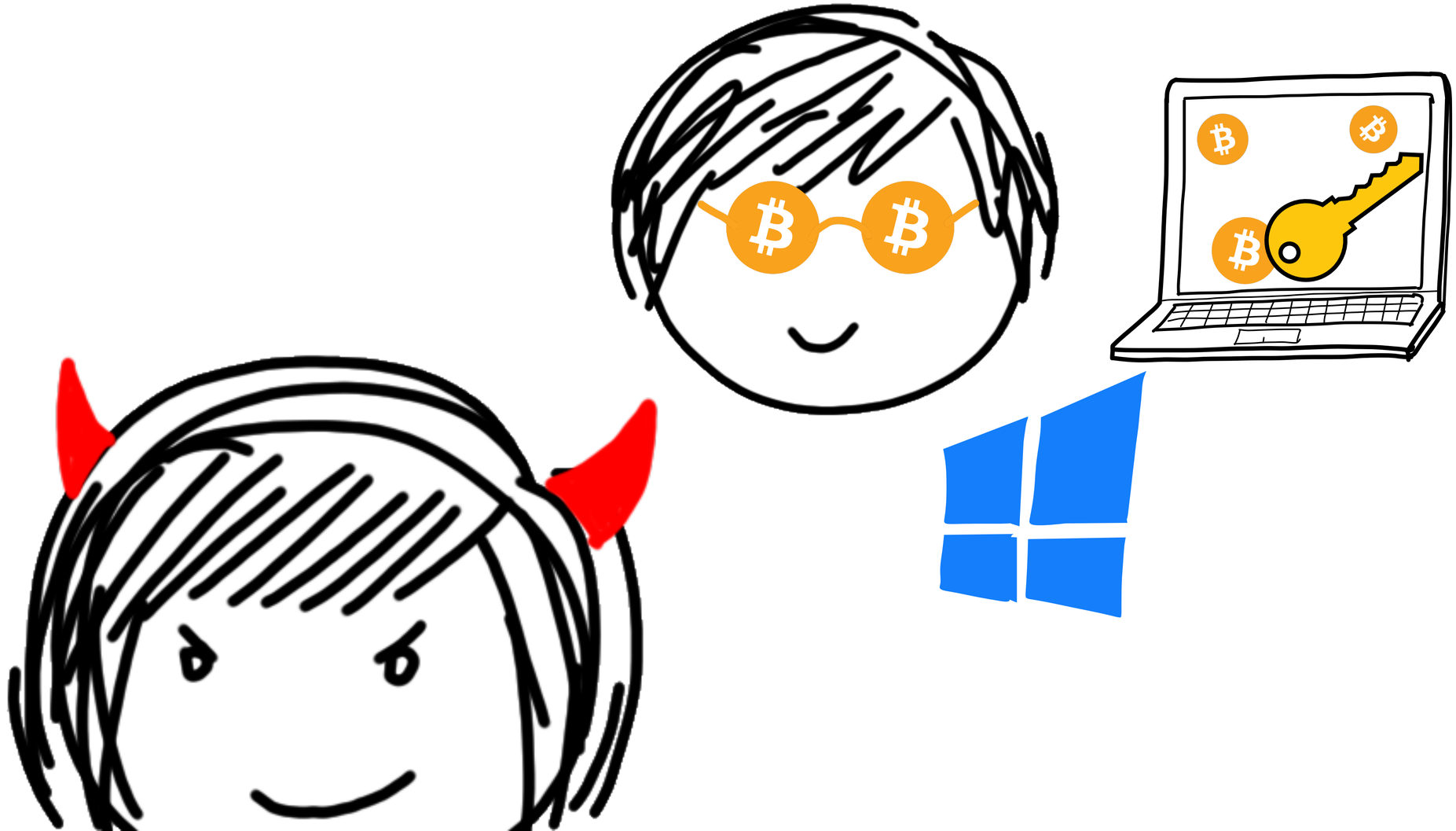
Ballad of ₿itcoin Bob



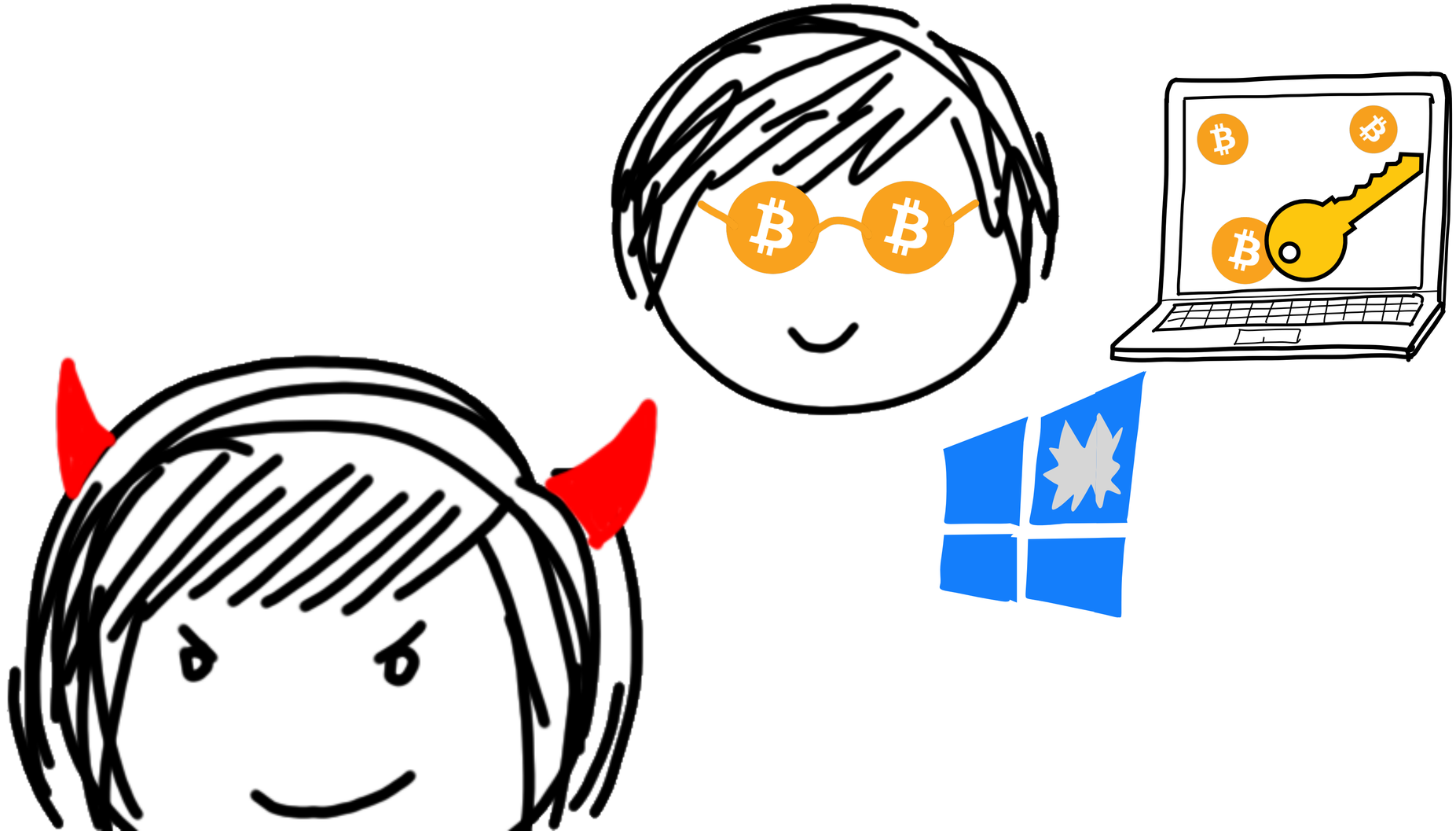
Ballad of ₿itcoin Bob



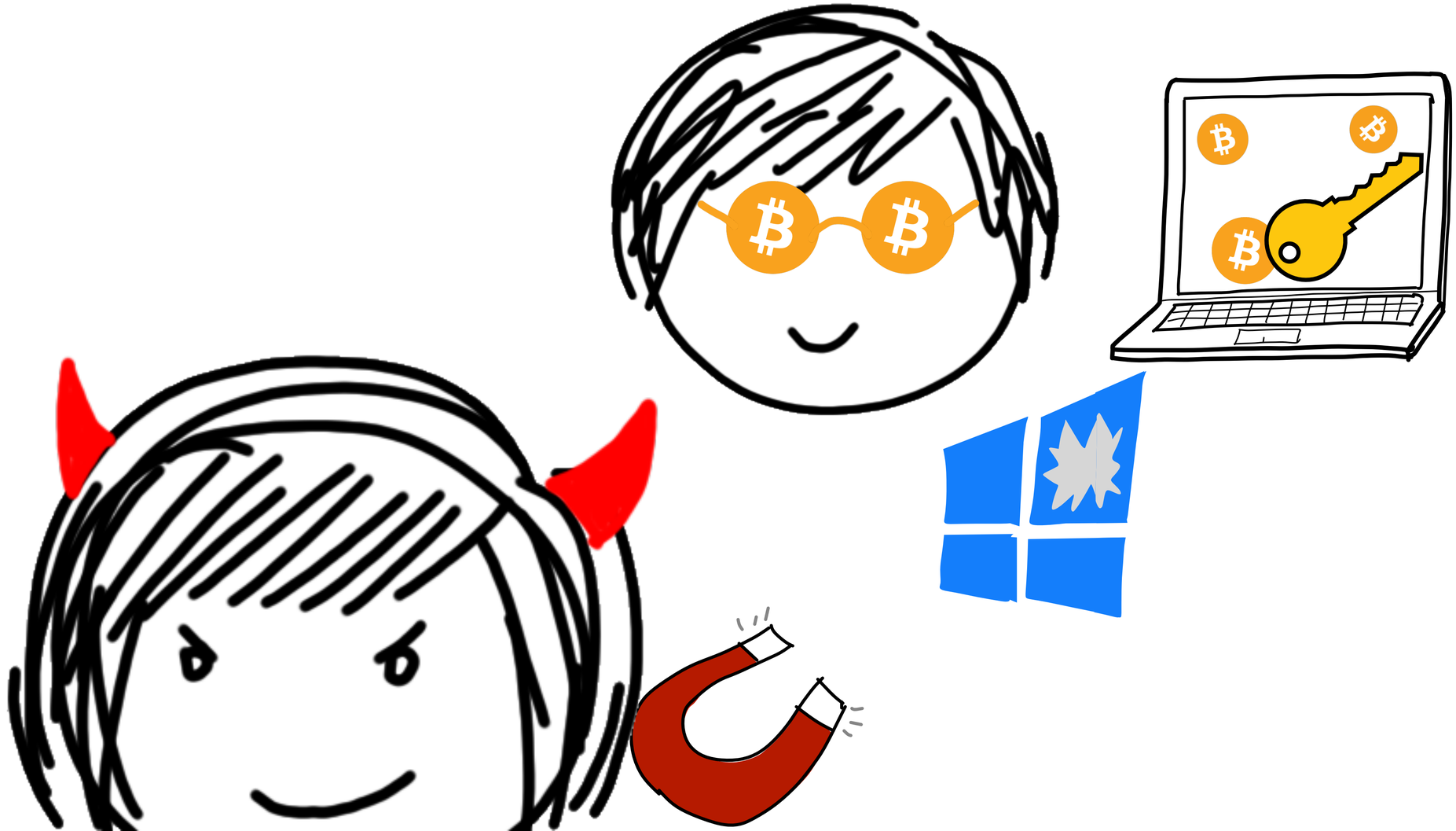
Ballad of ₿itcoin Bob



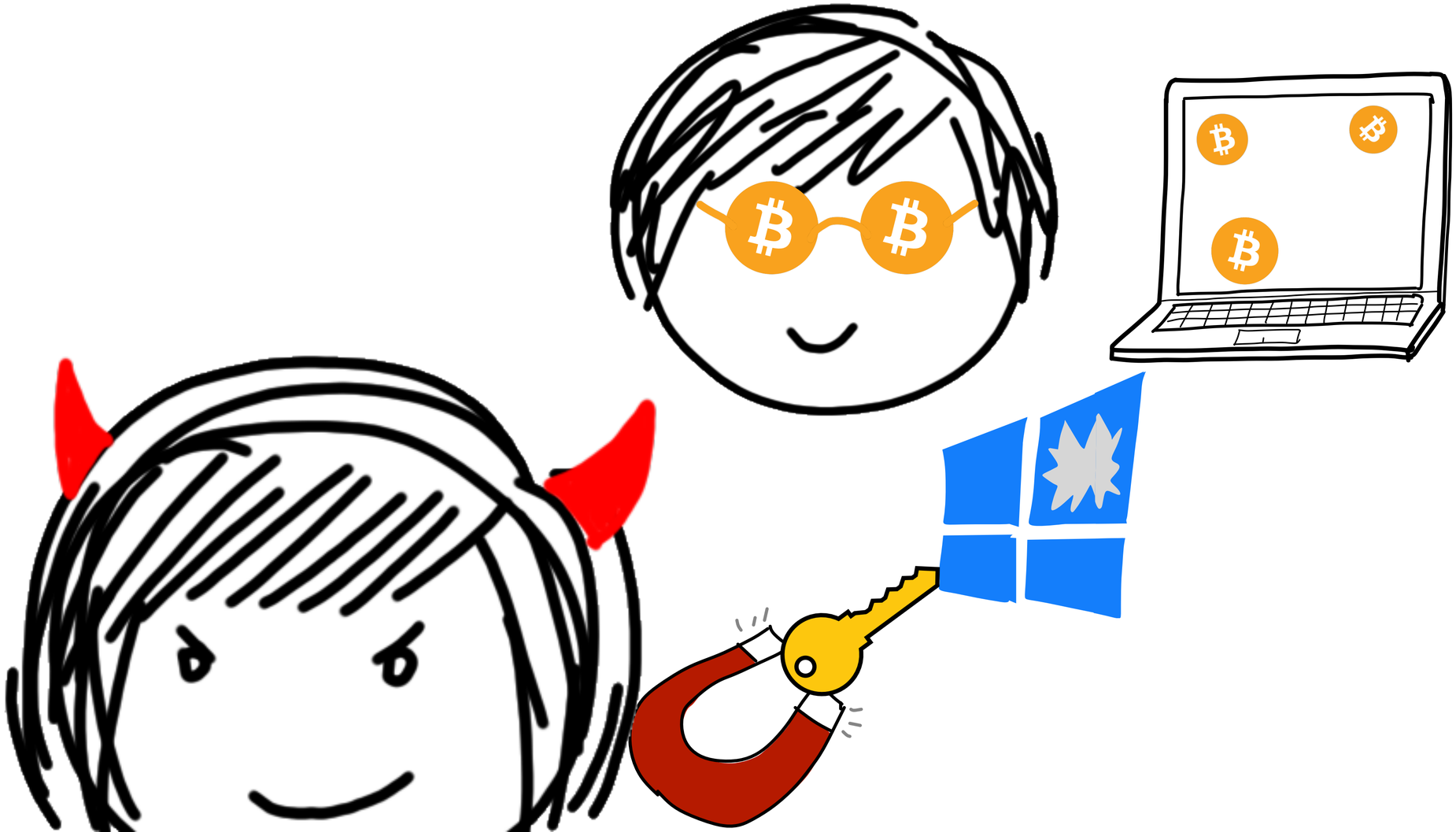
Ballad of ₿itcoin Bob



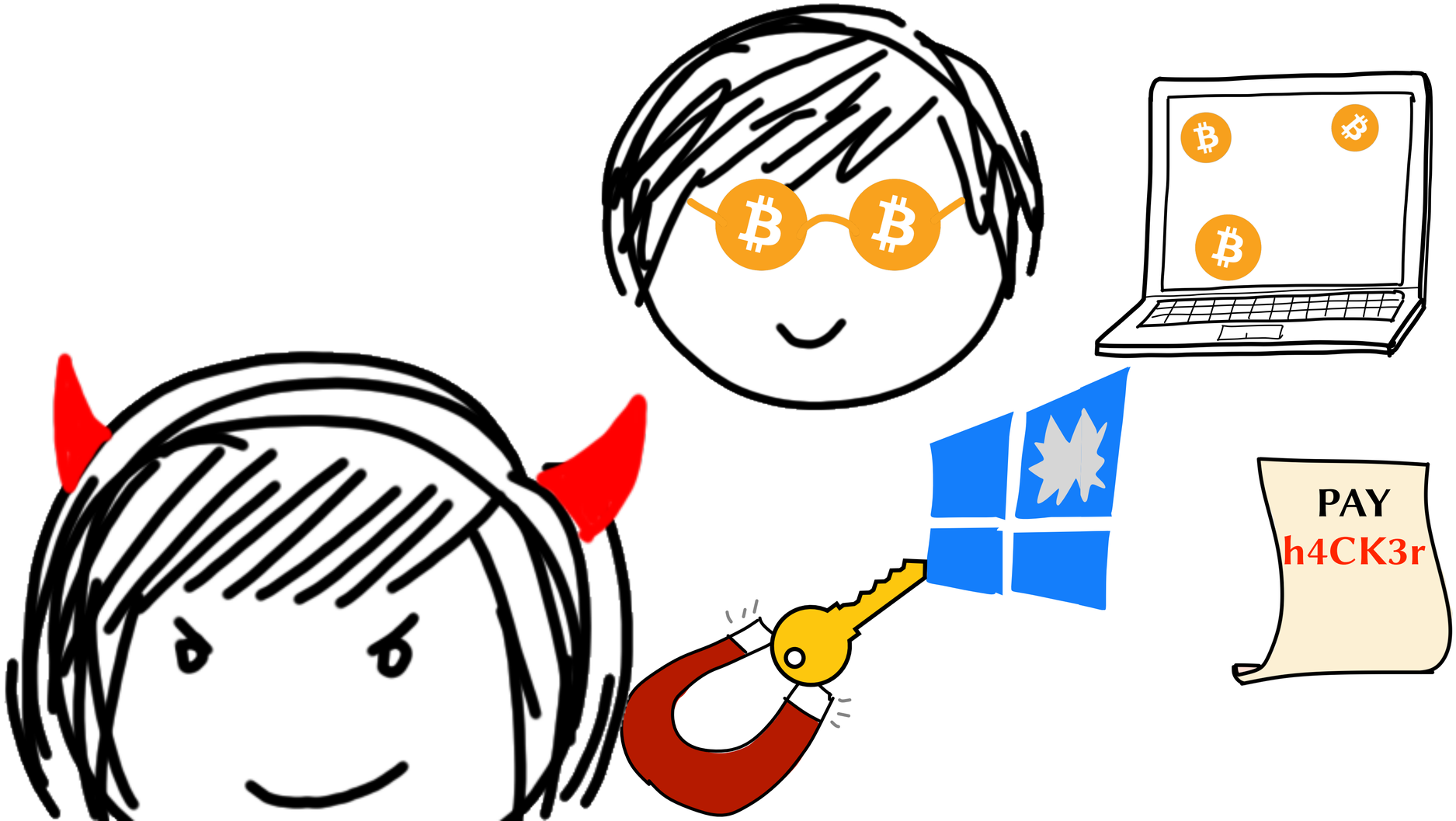
Ballad of ₿itcoin Bob



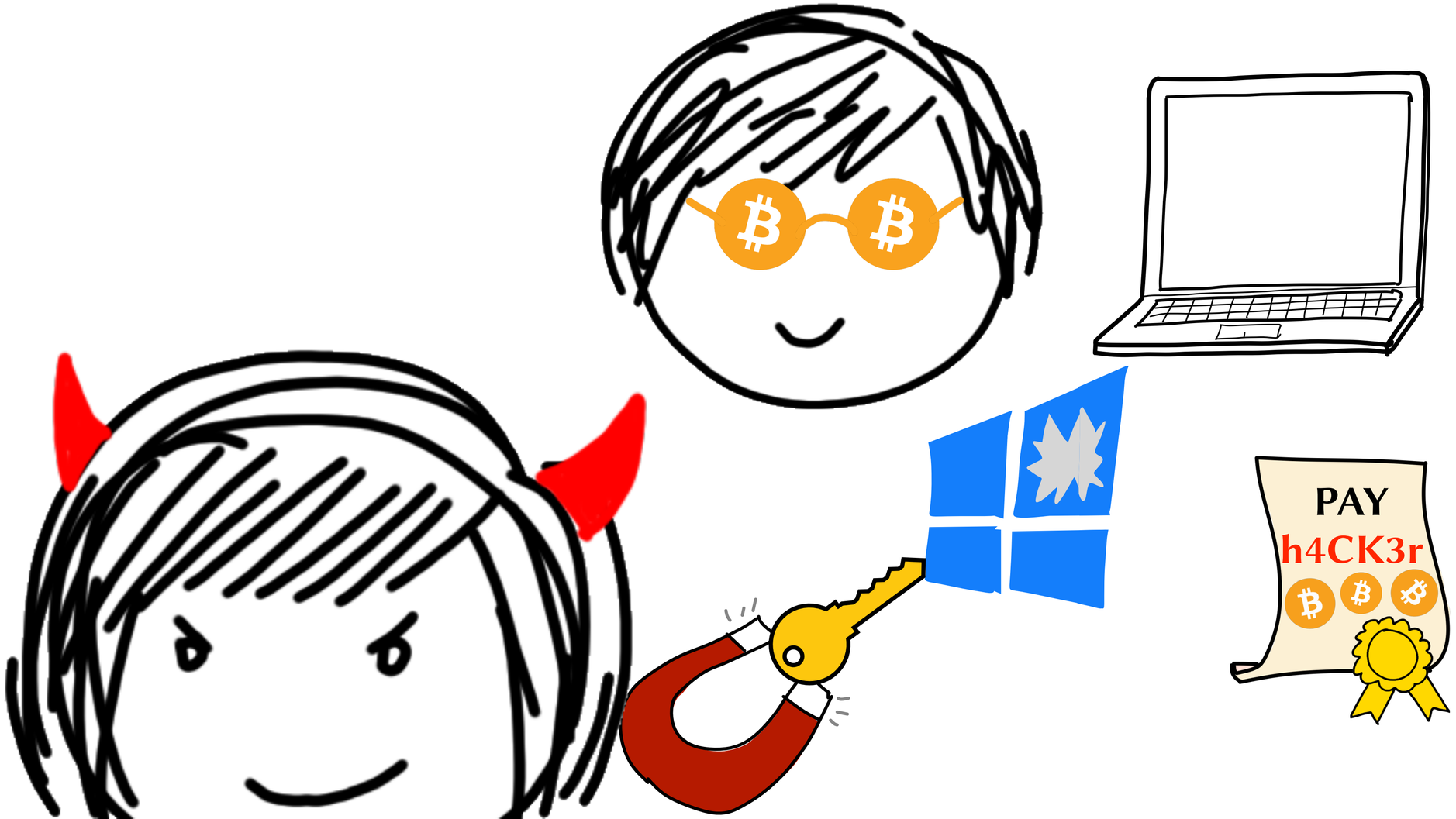
Ballad of ₿itcoin Bob



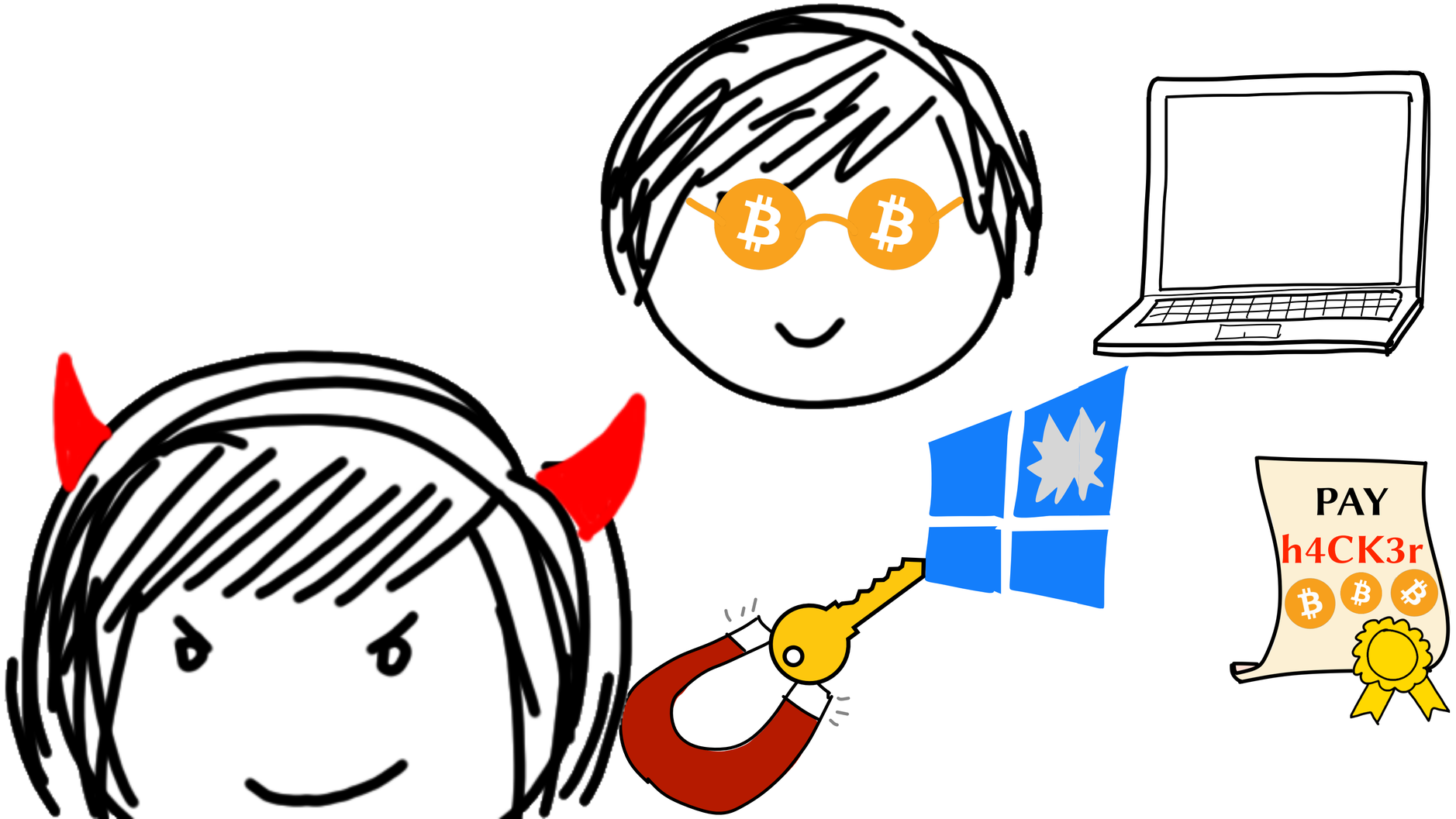
Ballad of ₿itcoin Bob



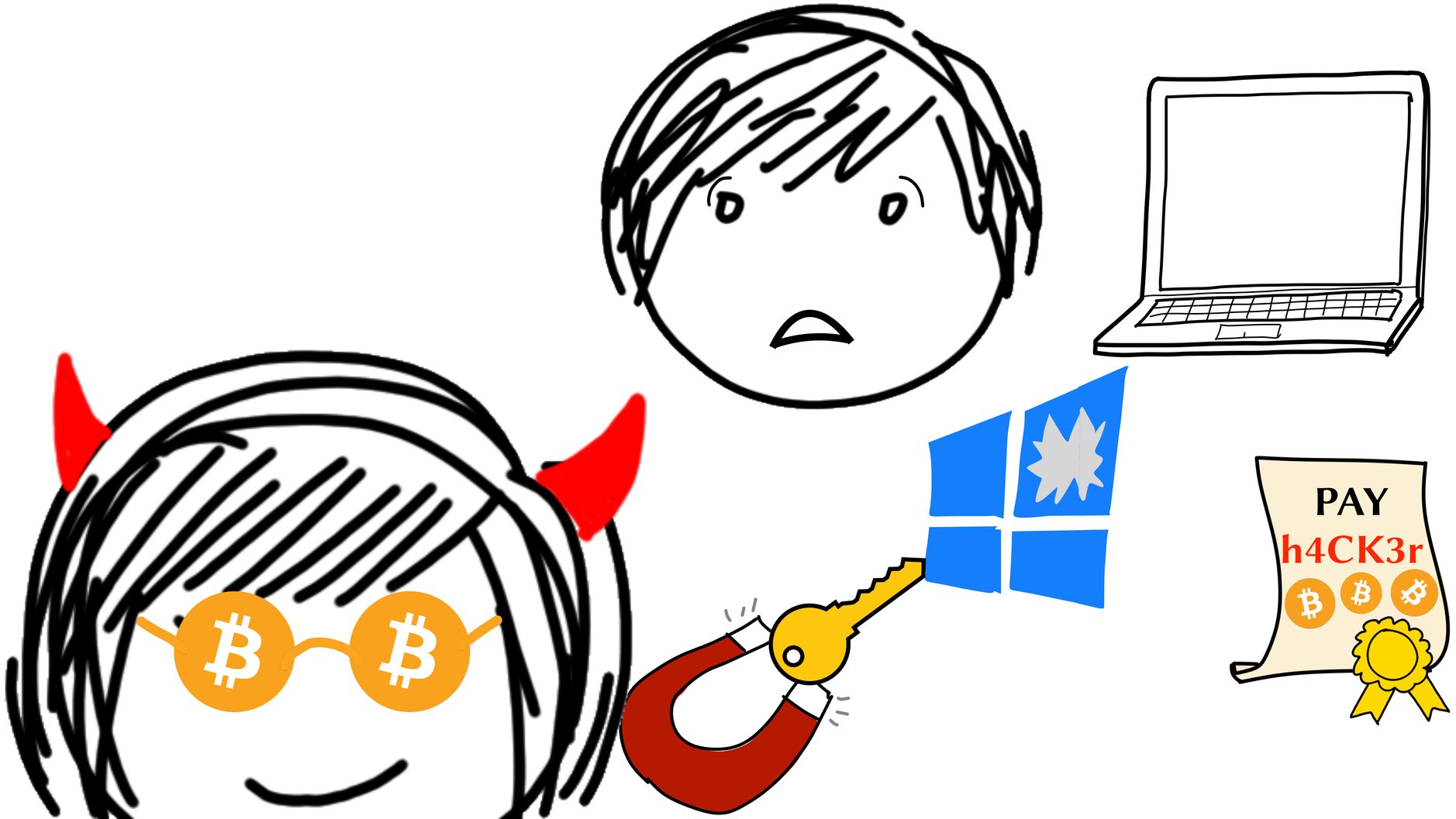
Ballad of ₿itcoin Bob



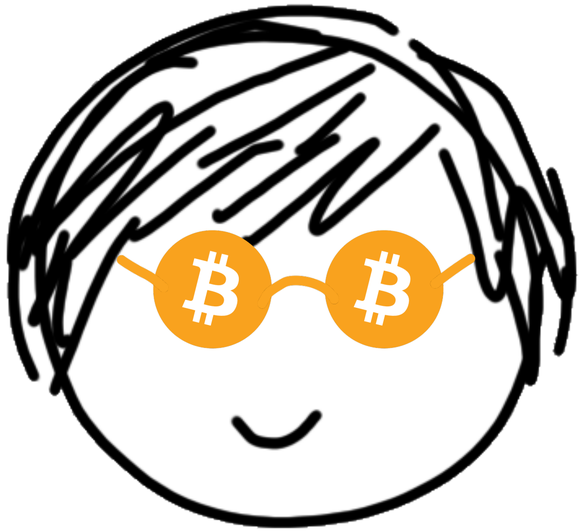
Ballad of ₿itcoin Bob



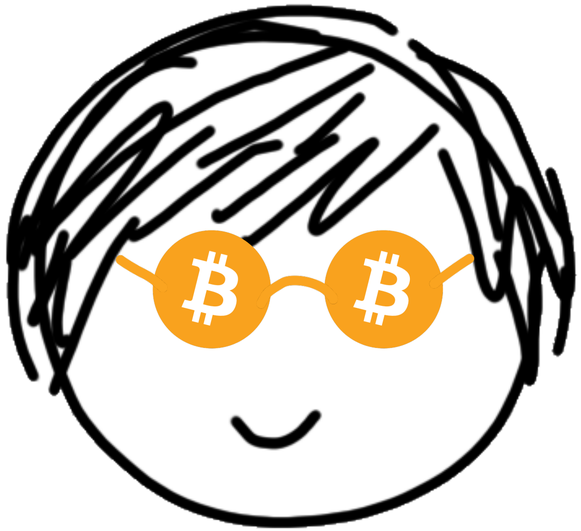
Ballad of ₿itcoin Bob



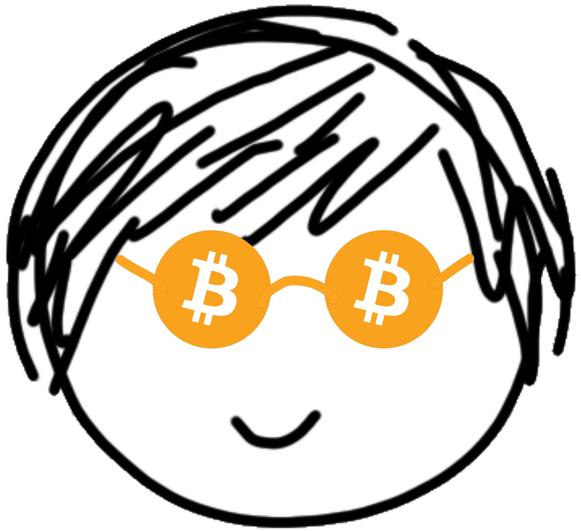
Ballad of ₿itcoin Bob



Ballad of ₿itcoin Bob

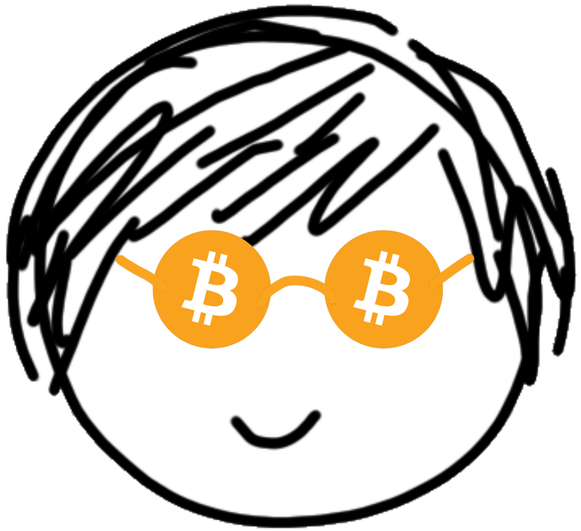


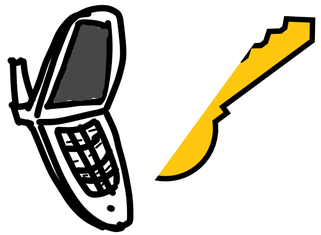
Ballad of ₿itcoin Bob

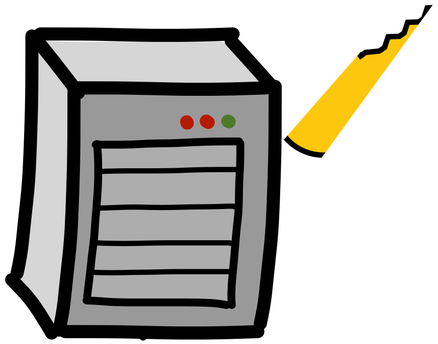


Ballad of ₿itcoin Bob

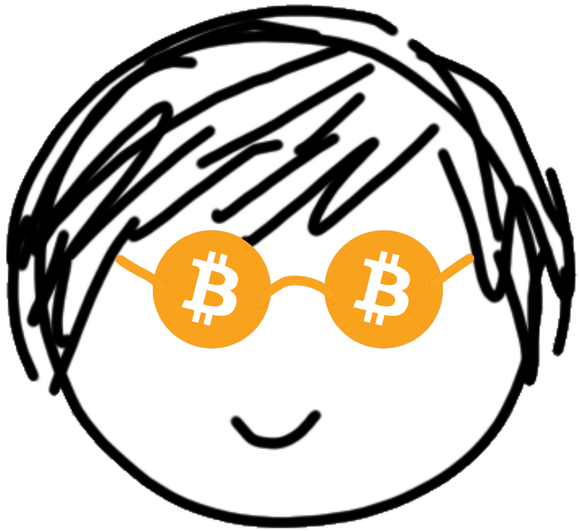
Ballad of ₿itcoin Bob

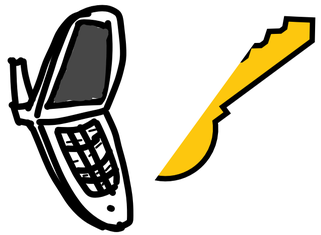


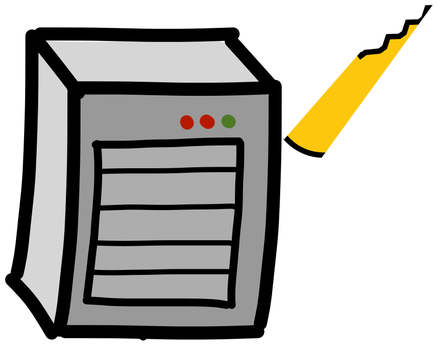




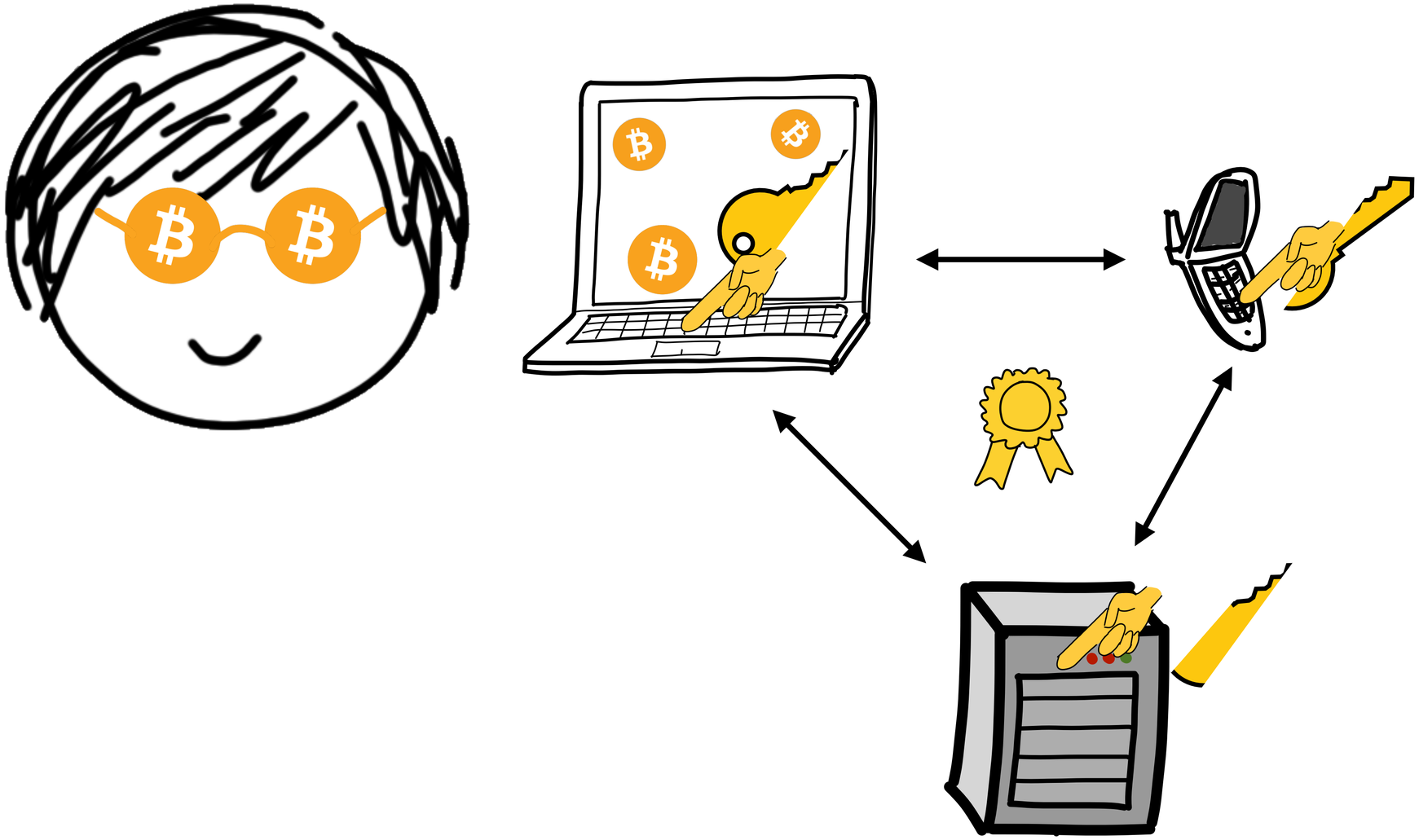
Ballad of ₿itcoin Bob



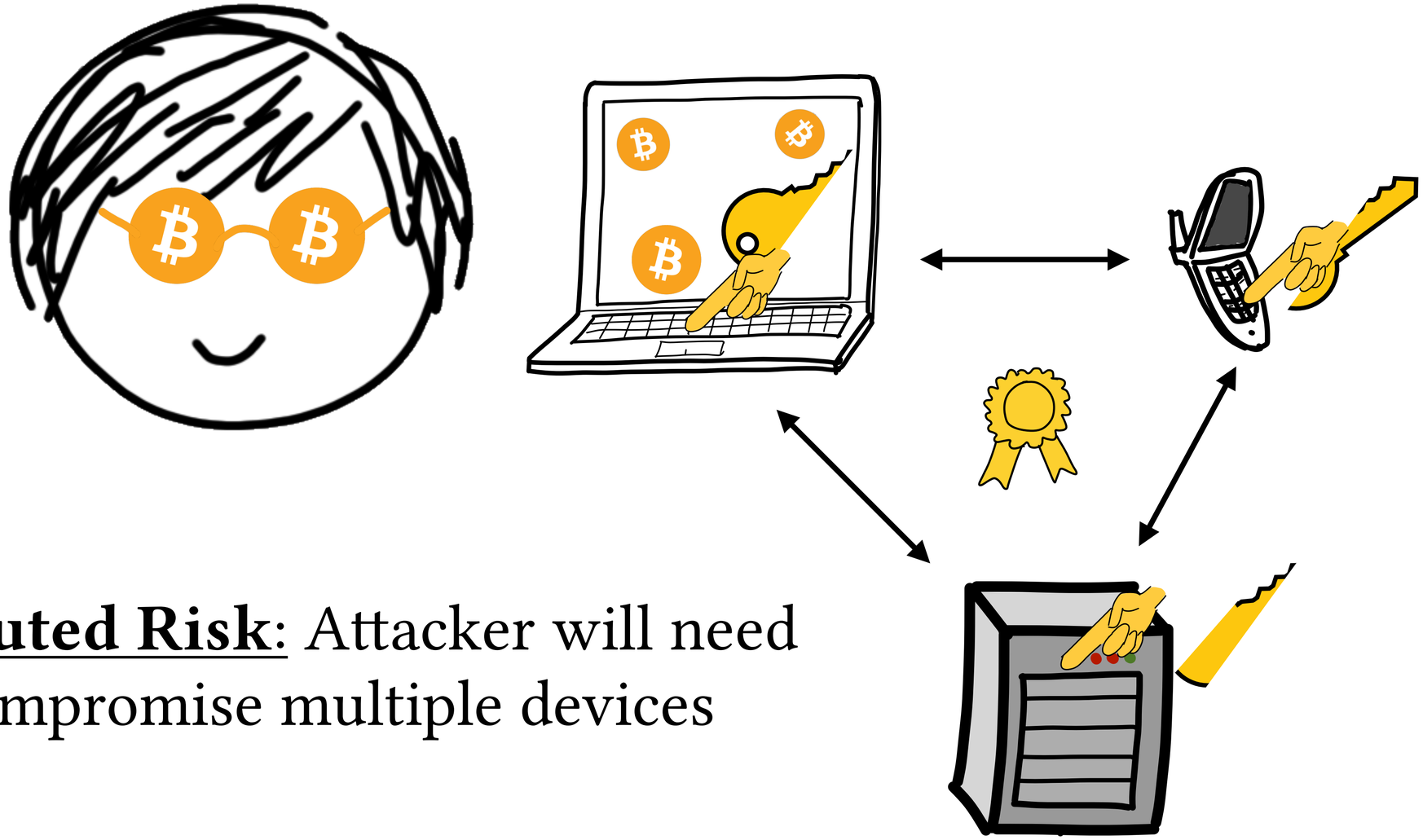




Ballad of ₿itcoin Bob



Ballad of ₿itcoin Bob



<u>Distributed Risk</u>: Attacker will need to compromise multiple devices

- **<u>Compatibility</u>**:
  Verifies w.r.t. original algorithm

- **<u>Corruption Resilience</u>**:
  Compromising some devices does not leak the signing key

- This talk: Signatures ⇔ Non-interactive Zero-knowledge

 $\Leftrightarrow \pi$

( , ) $\Leftrightarrow (x, w)$

Distributed Signing ⇔ Distributed Proving

How to Distribute Signing

- Any signing scheme can be distributed via general MPC
- "Practical" efficiency usually requires more fine-grained notions than just feasibility
- As one proxy, practical distributed signing protocols make **blackbox use** of non-linear components of the signing algorithm:
  - Integer arithmetic in $\mathbb{Z}_q$ or $\mathbb{Z}_N^*$
  - Elliptic curve group operations
  - Hash functions

How to Distribute Signing

- Any signing scheme can be distributed via general MPC
- "Practical" efficiency usually requires more fine-grained notions than just feasibility
- As one proxy, practical distributed signing protocols make **blackbox use** of non-linear components of the signing algorithm:
  - Integer arithmetic in $\mathbb{Z}_q$ or $\mathbb{Z}_N^*$
  - Elliptic curve group operations
  - Hash functions

RSA, Schnorr/EdDSA, ECDSA, BLS, BBS+, custom constructions using lattices, isogenies, etc.

What about Purely Hash Based?

- [Ozdemir Boneh 22]: distributed version of Fractal
  [Cui Zhang Chen Liu Yu 21]: distributed MPC-in-the-head
  Proof size, verifier time linear in #provers

- [Khaburzaniya Chalkias Lewi Malvai 21]: aggregate Lamport signatures with STARKs
  Prove statements about circuit representation of hash function

- [Dziembowski Faust Lizurej 23]: "individual cryptography"
  Hash-based proofs that are designed to be hard to distribute

- [Nielsen Hall-Andersen 23]: Incrementally Verifiable Computation must make non-blackbox use of hash function

This Work: Limitations

- For some hash based NIZKs[1], there is an inherent barrier[2] to designing practical protocols[3] to distribute their computation.

This Work: Limitations

- For some hash based NIZKs[1], there is an inherent barrier[2] to designing practical protocols[3] to distribute their computation.

1. NIZKs that have straight-line extractors in the Random-Oracle Model

This Work: Limitations

- For some hash based NIZKs[1], there is an inherent barrier[2] to designing practical protocols[3] to distribute their computation.

1. NIZKs that have straight-line extractors in the Random-Oracle Model

2. Attack that completely recovers the witness by corrupting all-but-one distributed provers

This Work: Limitations

- For some hash based NIZKs[1], there is an inherent barrier[2] to designing practical protocols[3] to distribute their computation.

1. NIZKs that have straight-line extractors in the Random-Oracle Model
2. Attack that completely recovers the witness by corrupting all-but-one distributed provers
3. Protocol that is blackbox in the same hash function (i.e. Random Oracle) as the NIZK

Implications for distributing…

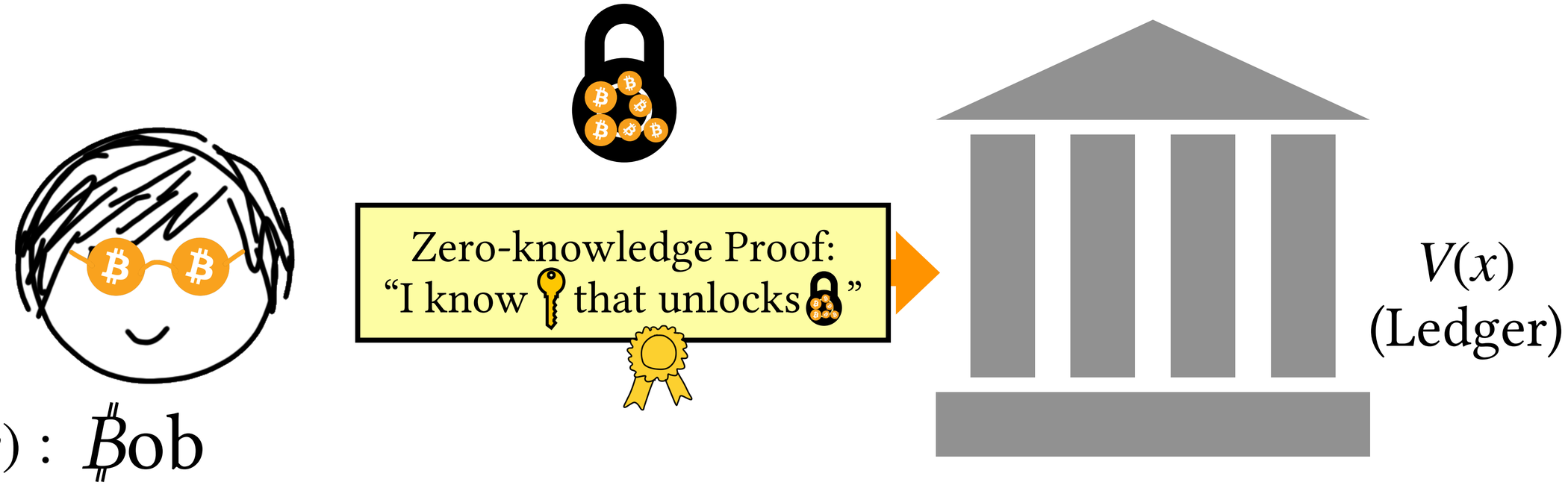
- Signing for standard schemes based on MPC-in-the-head
- NIZKs/signatures obtained by compiling Sigma protocols via:
  - Pass' or Fischlin's transformations (tight/concurrent security)
  - Unruh's transformation (post-quantum)
- PCPs/IOPs compiled via hash functions

Zero-knowledge Proofs

- Very powerful cryptographic primitive, introduced by [Goldwasser Micali Rackoff 85]
- Intuition: Prover convinces a Verifier of a statement, without revealing its secret trapdoor.
- In this talk, we only look at:
  - Non-interactive proofs (NIZK)
  - Proofs of Knowledge (PoK)

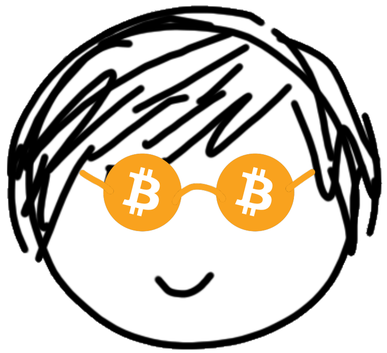
Zero-knowledge Proofs

- 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 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

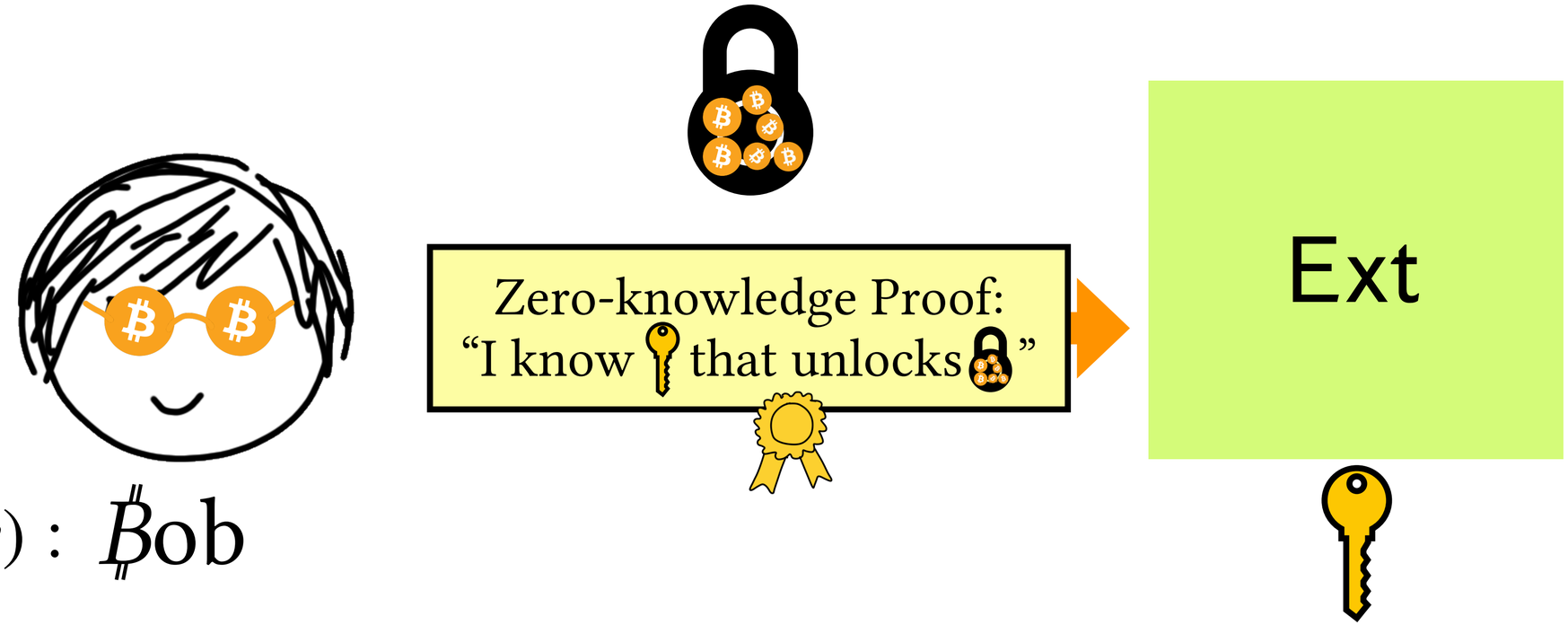


$P(x, w)$: Bob

Ext

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



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
- “Straight-line” Extraction (SLE): no rewinding. Instead, use other trapdoor like CRS, RO, etc.

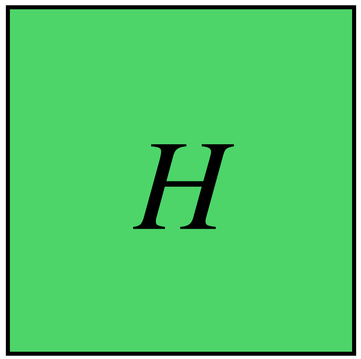
Why is Ext special?

- 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
- "Straight-line" Extraction (SLE): no rewinding. Instead, use other trapdoor like CRS, RO, etc.

Bad for:

- Quantum
- Concurrency
- Tightness

Random Oracle Model

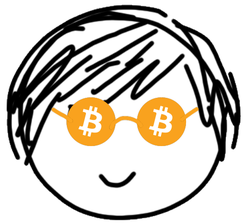


$$H : \{0,1\}^* \mapsto \{0,1\}^{\ell}$$

Random Oracles as Ext Privilege

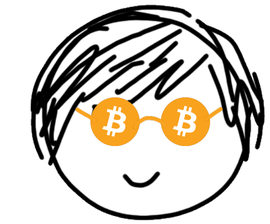
H

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



$P(x, w)$

Random Oracles as Ext Privilege



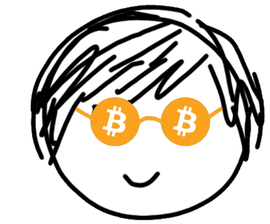
$P(x, w)$

H

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

Q_1	Q_2	•	•	Q_i	•	•	•	•	•	Q_n

Random Oracles as Ext Privilege



$P(x, w)$

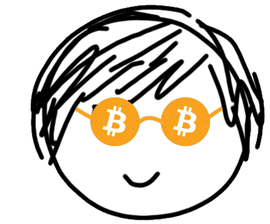
H

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

Q_1	Q_2	•	•	Q_i	•	•	•	•	•	Q_n

$V(x)$

Random Oracles as Ext Privilege



$P(x, w)$

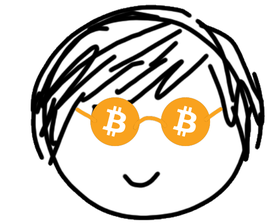
H

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

Q_1	Q_2	•	•	Q_i	•	•	•	•	•	Q_n

$V(x)$

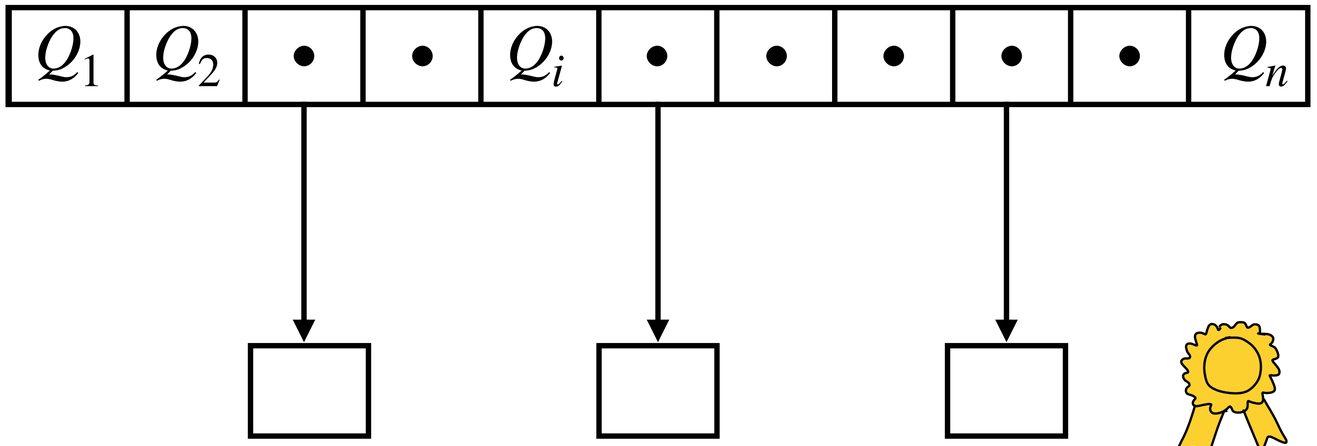
Random Oracles as Ext Privilege



$P(x, w)$

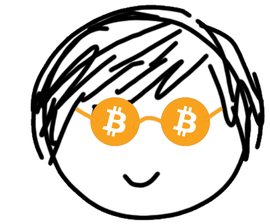
H

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



$V(x)$

Random Oracles as Ext Privilege



$P(x, w)$

H

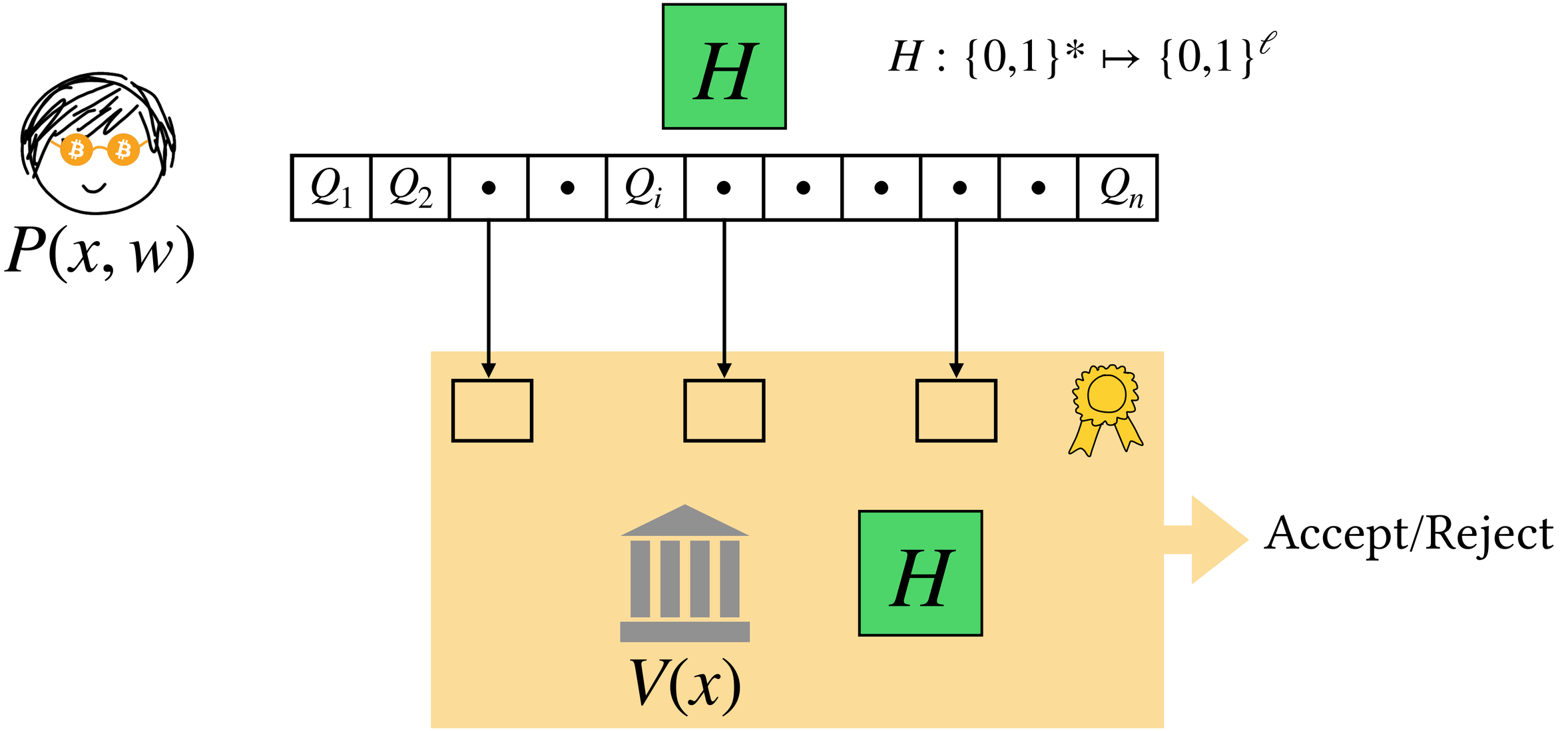
$H : \{0,1\}^* \mapsto \{0,1\}^\ell$

Q_1 Q_2 • • Q_i • • • • • Q_n

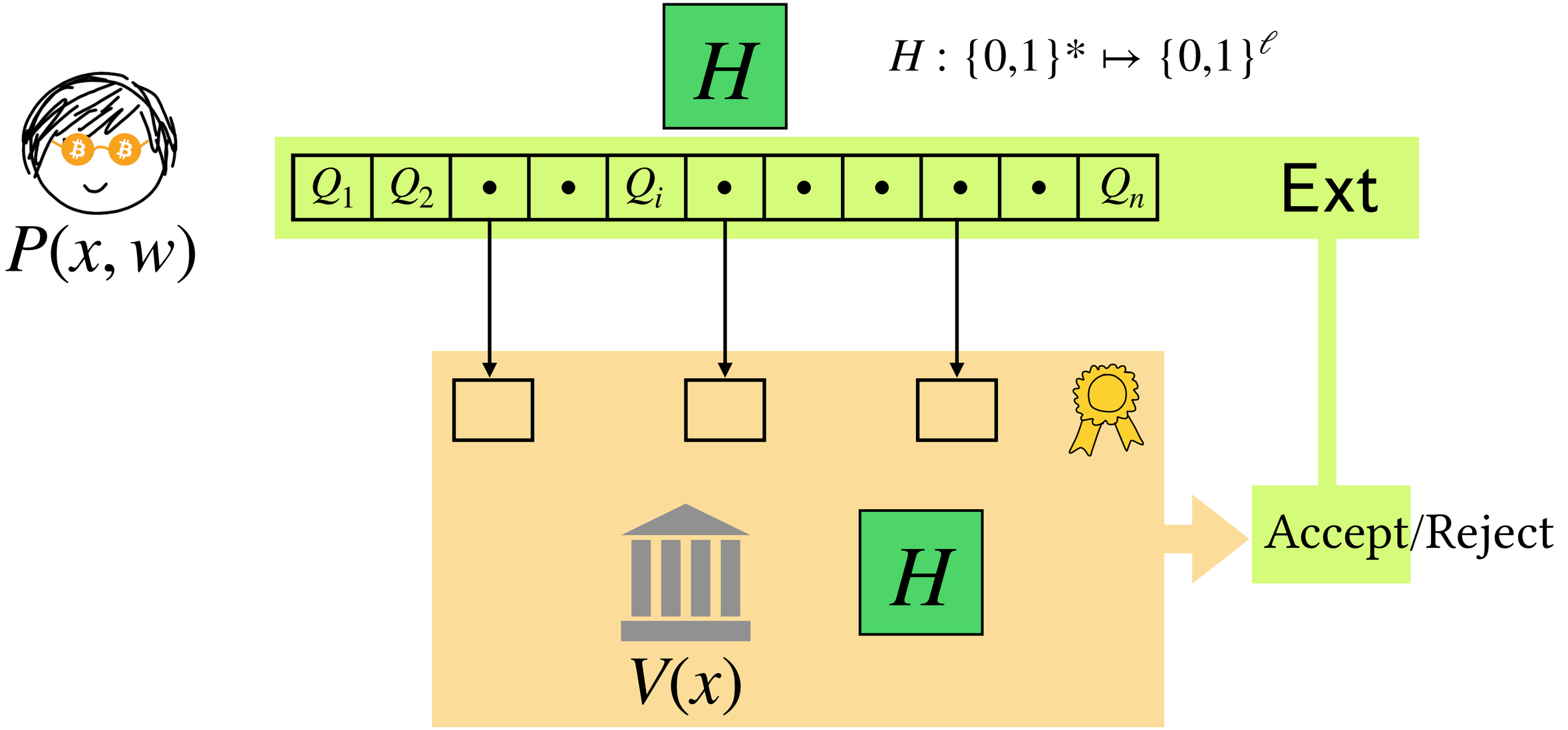
$V(x)$

H

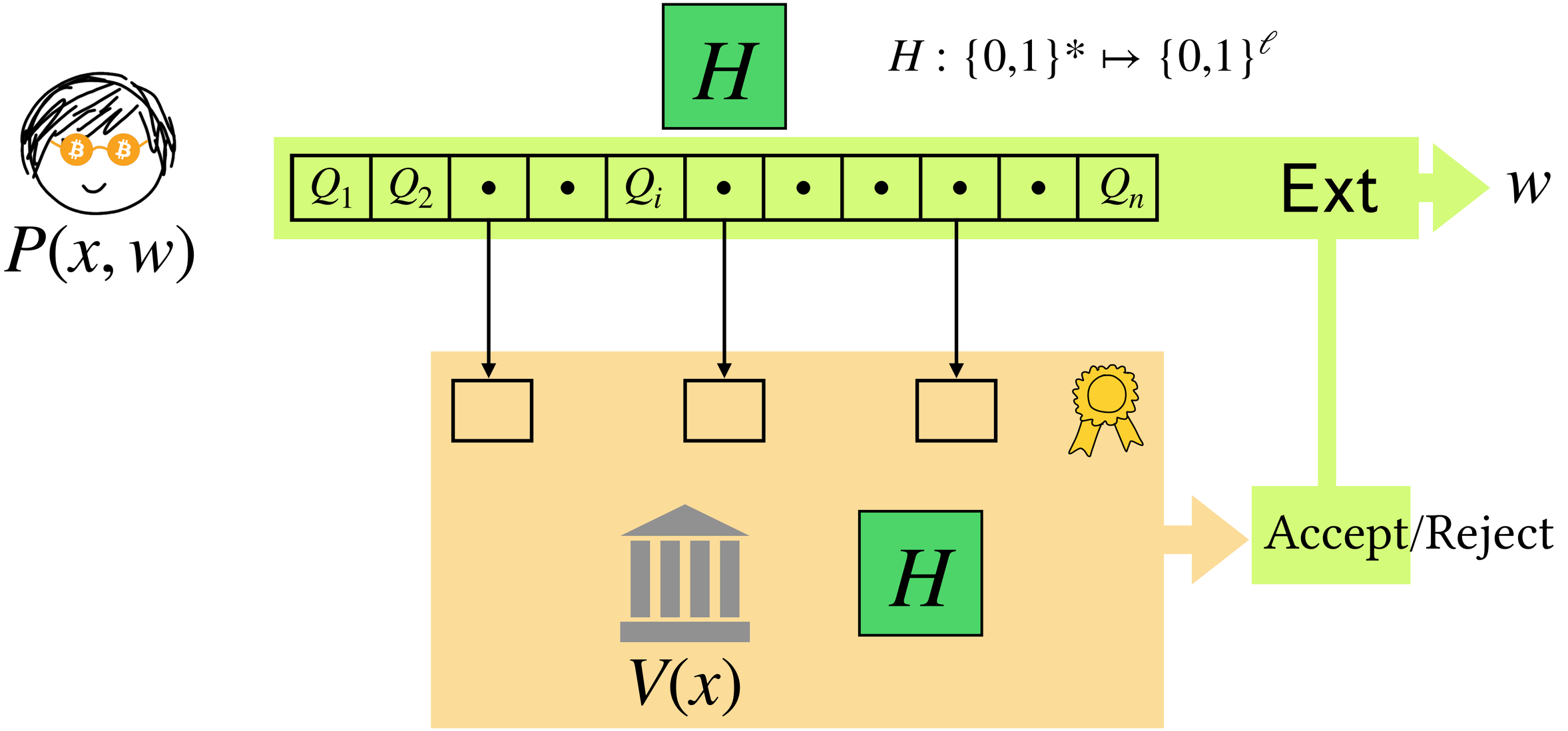
Random Oracles as Ext Privilege



Random Oracles as Ext Privilege



Random Oracles as Ext Privilege



Random Oracles as Ext Privilege

- Why is it a meaningful trapdoor?
  - Hash functions are complex and highly unstructured
  - Bob must "query" each Q_i to H to obtain $H(Q_i)$
  - Ext gets $\{Q_i\}$ without rewinding
- Practical usage:
  - No "trusted setup", each query is very cheap
  - Many NIZKs **happen to achieve** SLE in the ROM

Distributing NIZKs in the ROM

- Multiparty protocols to securely compute RO-based NIZKs should ideally make blackbox use of H
  - Conceptually: H should not have a circuit description
  - Practically: hash functions have large circuits
- We call them "Oracle Respecting Distributed" (ORD) protocols

Trivial Oracle Respecting Distribution

$$\pi \leftarrow P(x, w) \qquad V(x, \pi) = 1$$

Consider languages where (x, w) can be "secret shared":

$x_0 + x_1 + x_2 = x \qquad w_0 + w_1 + w_2 = w$ (think DLog)

$(x_0, w_0), (x_1, w_1), (x_2, w_2) \in L \Leftrightarrow (x, w) \in L$

Trivial Oracle Respecting Distribution

$$\pi \leftarrow P(x, w) \qquad V(x, \pi) = 1$$

Consider languages where (x, w) can be "secret shared":

$x_0 + x_1 + x_2 = x \qquad w_0 + w_1 + w_2 = w$ (think DLog)

$(x_0, w_0), (x_1, w_1), (x_2, w_2) \in L \Leftrightarrow (x, w) \in L$

$P^3(x, w)$:

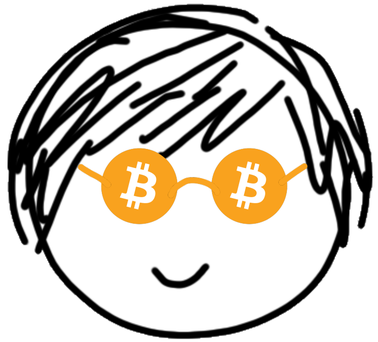
$w_0, w_1, w_2 \leftarrow \mathsf{Share}(w)$

Output $\{\pi_i = P(x_i, w_i)\}_{i \in [3]}$

$V^3(x, \pi_0, \pi_1, \pi_2)$:

$V(x_0, \pi_0) \wedge V(x_1, \pi_1)$
$\wedge V(x_2, \pi_2)$

Trivial Oracle Respecting Distribution

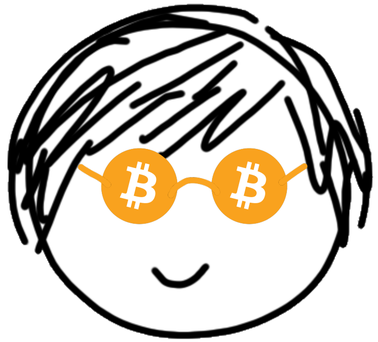


$P^3(x, w)$:

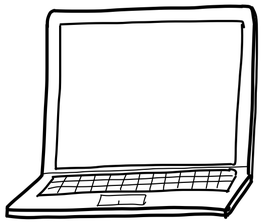
$w_0, w_1, w_2 \leftarrow \mathsf{Share}(w)$

Output $\{\pi_i = P(x_i, w_i)\}_{i \in [3]}$

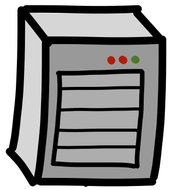
Trivial Oracle Respecting Distribution



$P^3(x, w)$:



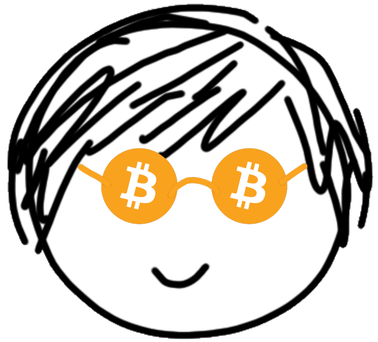
w_0



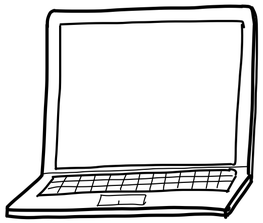
w_1

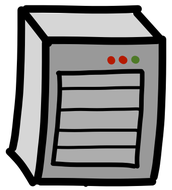
w_2

Trivial Oracle Respecting Distribution



$P^3(x, w)$:

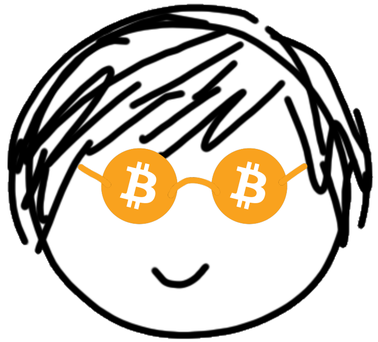




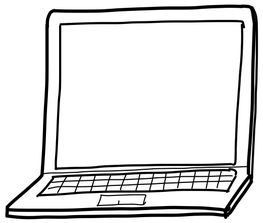
w_0 w_1 w_2

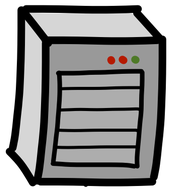
$\pi_0 \leftarrow P(w_0)$ $\pi_1 \leftarrow P(w_1)$ $\pi_2 \leftarrow P(w_2)$

Trivial Oracle Respecting Distribution



$P^3(x, w)$:



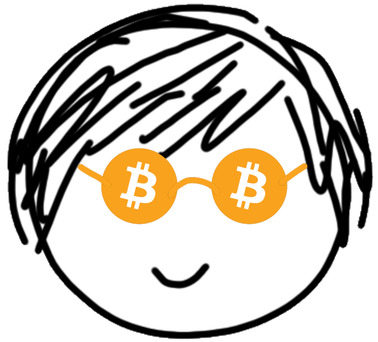


w_0 w_1 w_2

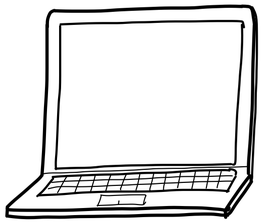
$\pi_0 \leftarrow P(w_0)$ $\pi_1 \leftarrow P(w_1)$ $\pi_2 \leftarrow P(w_2)$

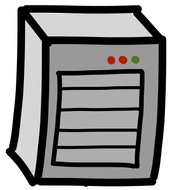
Output (π_0, π_1, π_2)

Trivial Oracle Respecting Distribution



$P^3(x, w)$:



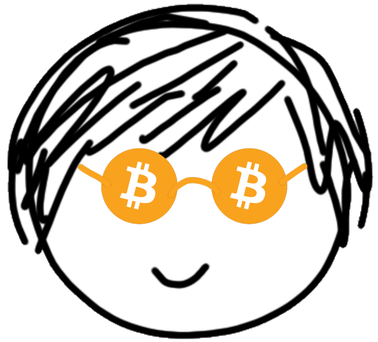


H w_0 $\quad$ H w_1 $\quad$ H w_2

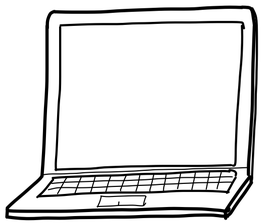
$\pi_0 \leftarrow P(w_0)$ $\quad$ $\pi_1 \leftarrow P(w_1)$ $\quad$ $\pi_2 \leftarrow P(w_2)$

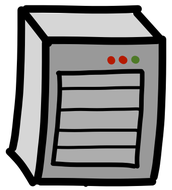
Output (π_0, π_1, π_2)

Trivial Oracle Respecting Distribution



$P^3(x, w)$:





H w_0 $\quad$ H w_1 $\quad$ H w_2

$\pi_0 \leftarrow P(w_0)$ $\quad$ $\pi_1 \leftarrow P(w_1)$ $\quad$ $\pi_2 \leftarrow P(w_2)$

Additive secret sharing:
Resilience to two corruptions

Output (π_0, π_1, π_2)

Oracle Respecting Distribution: Notes

- Imagine if P^3 had to be distributed among *four* parties instead of three
- In general: P^* that outputs $n \times \pi$ can be distributed amongst n parties, **as long as V^* is aware of n**
- We show that for any NIZK that is SLE in the ROM, this is inherent in the $n - 1$ corruption setting

V "unaware" of n

- Consider a proof system (P^H, V^H) for some language
- Assumption: $n \in \mathsf{poly}(\kappa)$ is a strict upper bound on queries made by V to the random oracle H
  - Holds for most 'natural' schemes
- We will show: any $n + 1$-party protocol that ORD-computes P^H will leak the witness to n parties

Trimming Resilience

$P(x, w)$

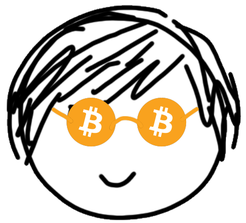
H

Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

π

Trimming Resilience

H



$P(x, w)$

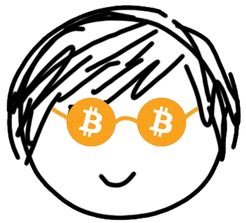
Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

π

$V(x)$

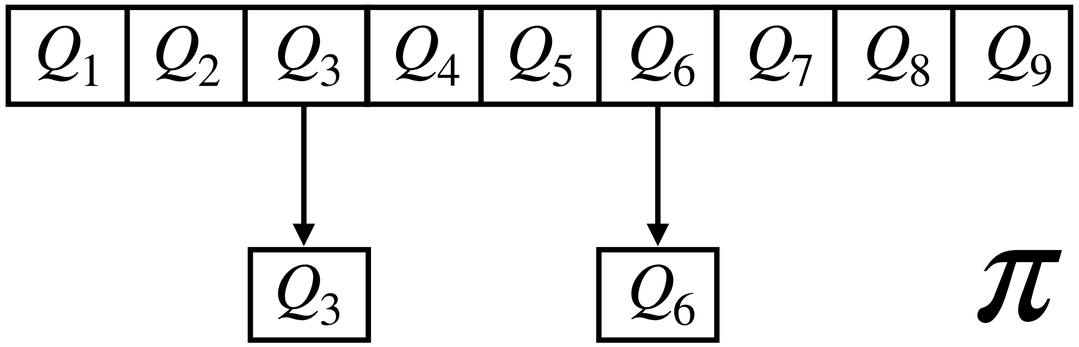
V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$

H

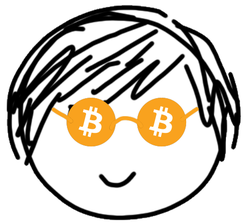


π

$V(x)$

V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$

H

Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

Q_3 Q_6

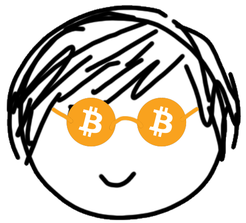
π

$V(x)$

H

V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$

H

Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

$V(x)$

H

V checks at most $n = 2$ queries

Trimming Resilience

$P(x, w)$

H

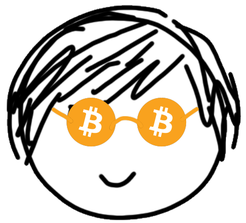
Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

$V(x)$

H

V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$

H

At most two partitions will be touched by V

Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

$V(x)$

H

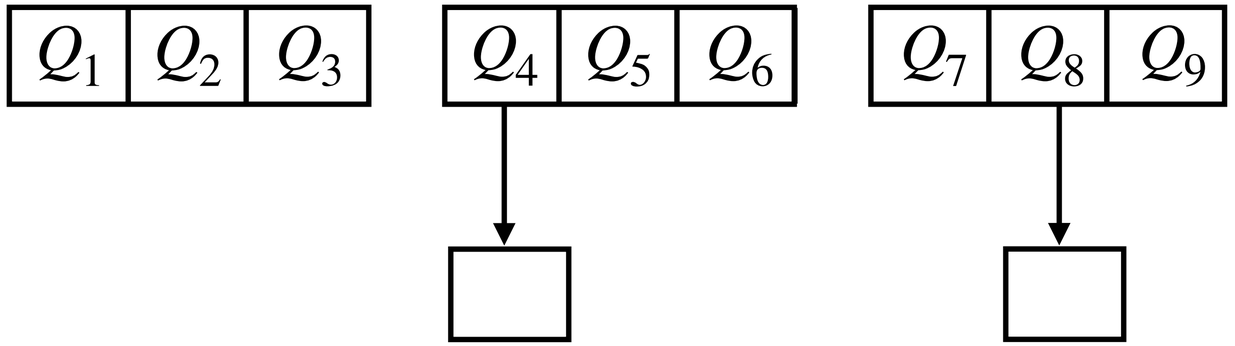
V checks at most $n = 2$ queries

Trimming Resilience

$P(x, w)$

H

At most two partitions will be touched by V

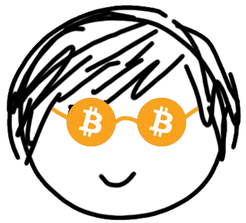


$V(x)$

H

V checks at most $n = 2$ queries

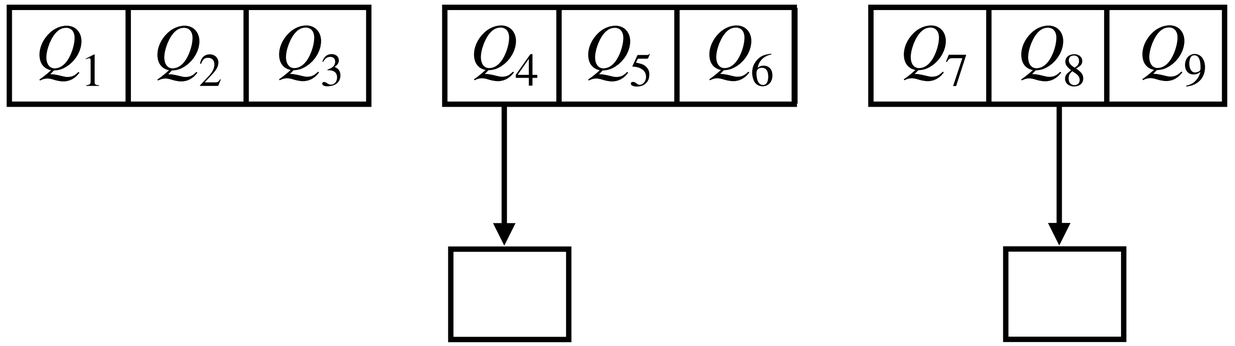
Trimming Resilience



$P(x, w)$

H

At most two partitions will be touched by V



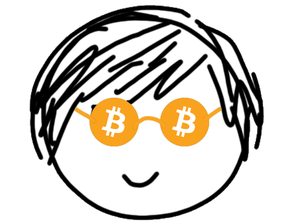
Randomly selected partition:
Pr[untouched by V] $\geq 1/3$

$V(x)$

H

V checks at most $n = 2$ queries

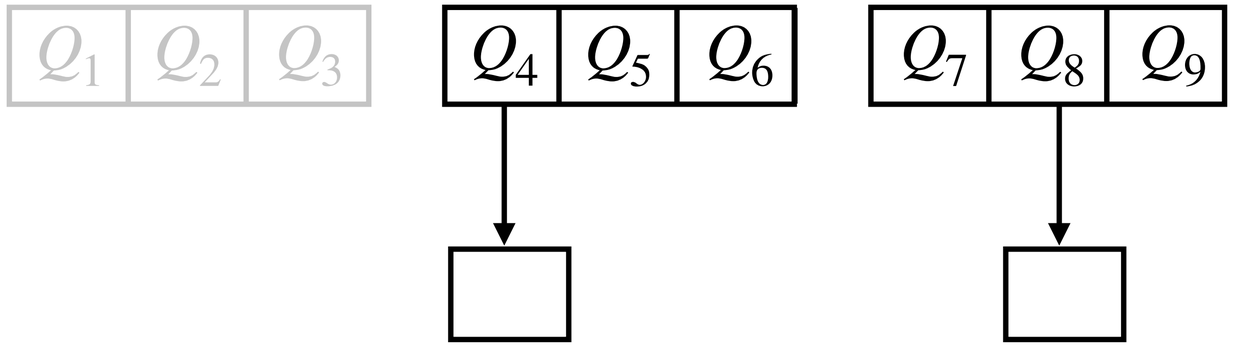
Trimming Resilience



$P(x, w)$

H

At most two partitions will be touched by V



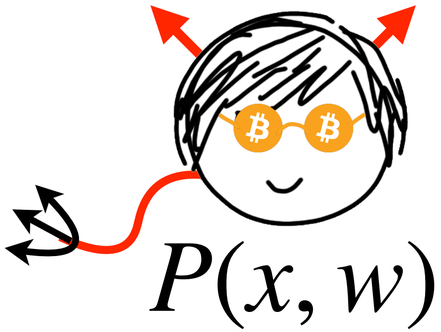
Randomly selected partition:
Pr[untouched by V] $\geq 1/3$

$V(x)$

H

V checks at most $n = 2$ queries

Trimming Resilience

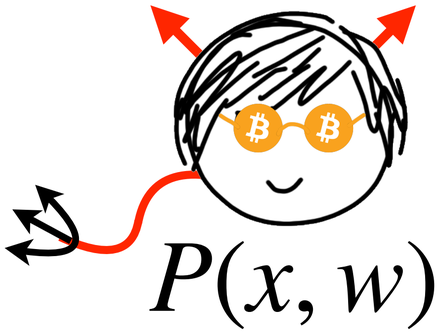


H

Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

V checks at most $n = 2$ queries

Trimming Resilience



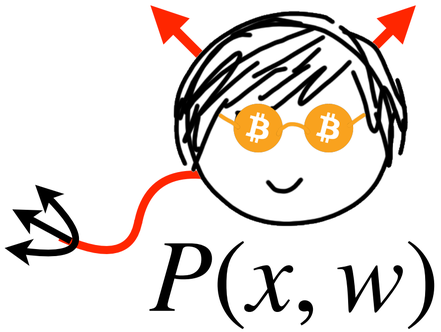
H^*

H

Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

V checks at most $n = 2$ queries

Trimming Resilience



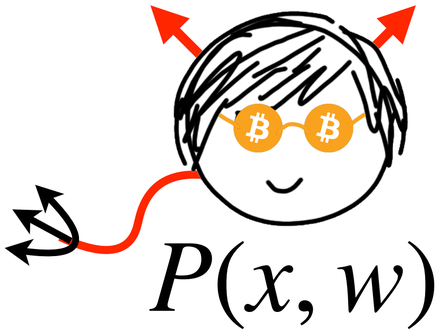
H

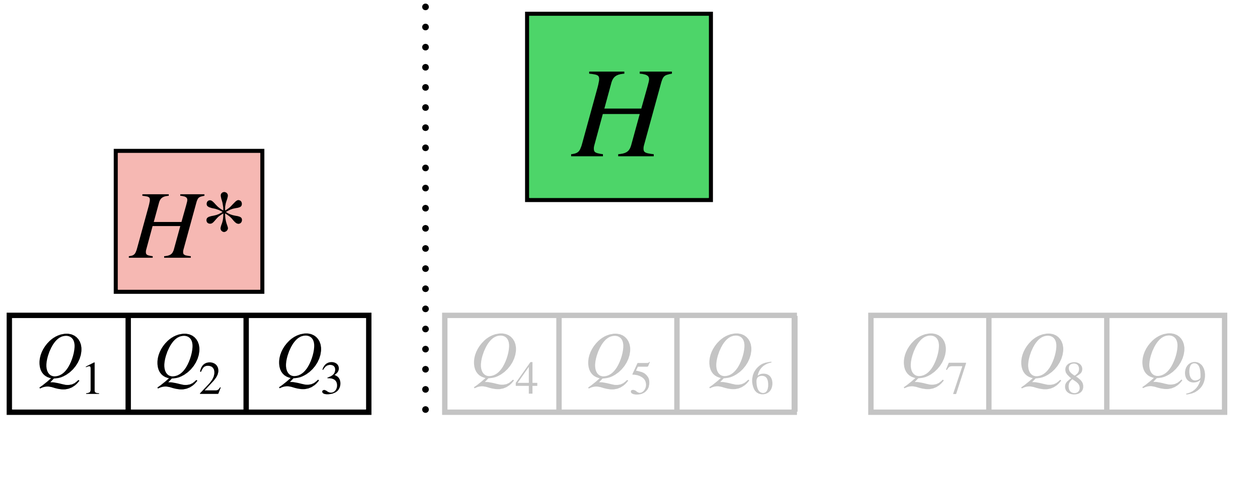
H^*

Q_1 Q_2 Q_3 Q_4 Q_5 Q_6 Q_7 Q_8 Q_9

V checks at most $n = 2$ queries

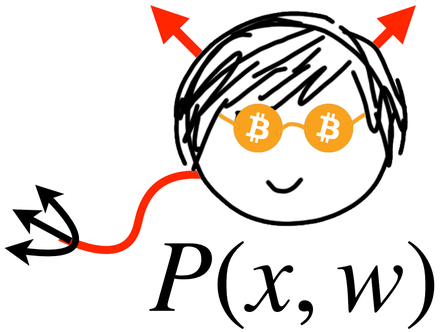
Trimming Resilience

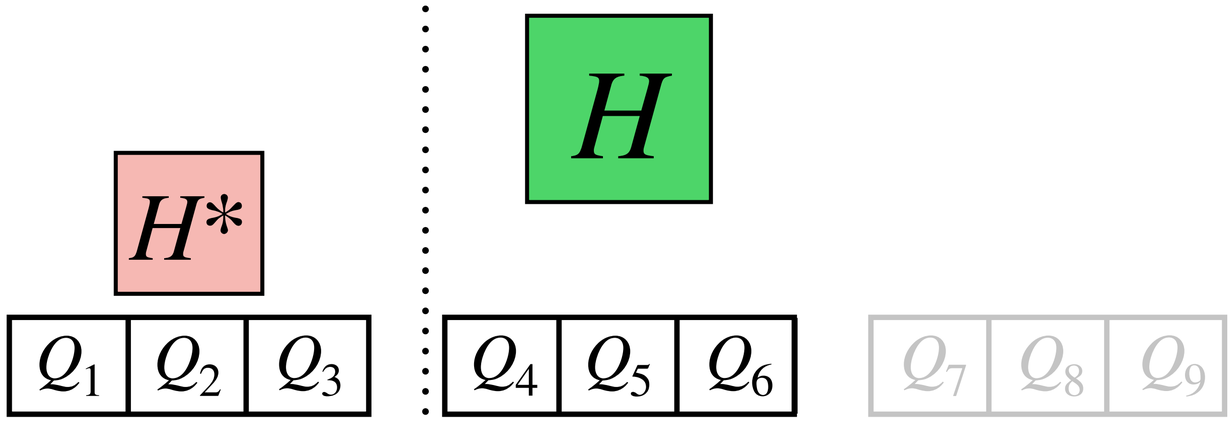




V checks at most $n = 2$ queries

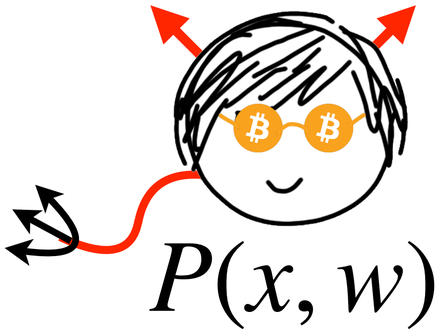
Trimming Resilience





V checks at most
$n = 2$ queries

Trimming Resilience



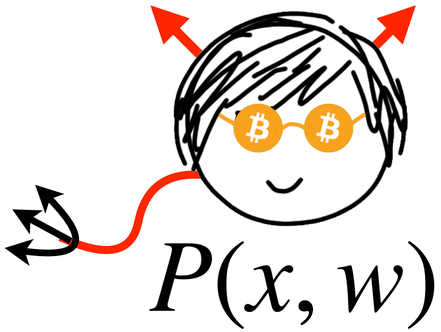
H^*

H

Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$

H^*

Q_1 Q_2 Q_3

H

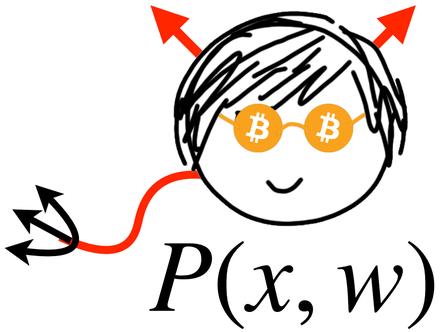
Q_4 Q_5 Q_6 Q_7 Q_8 Q_9

$V(x)$

H

V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$

H^*

Q_1 Q_2 Q_3

H

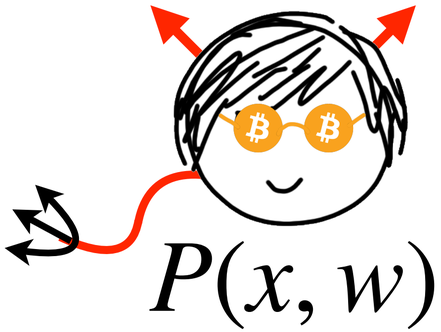
Q_4 Q_5 Q_6 Q_7 Q_8 Q_9

$V(x)$

H

V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$

H^*

Q_1 Q_2 Q_3

H

Q_4 Q_5 Q_6 Q_7 Q_8 Q_9

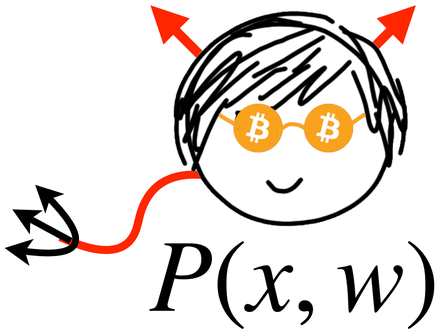
Reject

$V(x)$

H

V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$

H^*

Q_1 Q_2 Q_3

H

Q_4 Q_5 Q_6

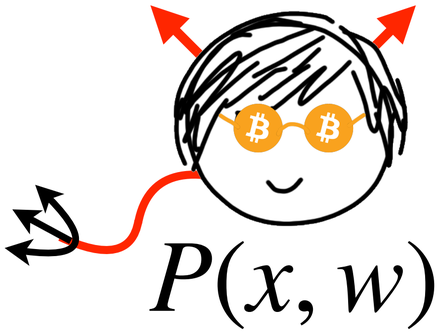
Q_7 Q_8 Q_9

$V(x)$

H

V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$

H^*

Q_1 Q_2 Q_3

H

Q_4 Q_5 Q_6

Q_7 Q_8 Q_9

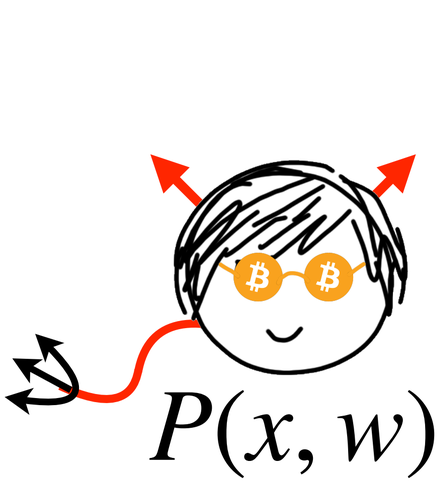
$V(x)$

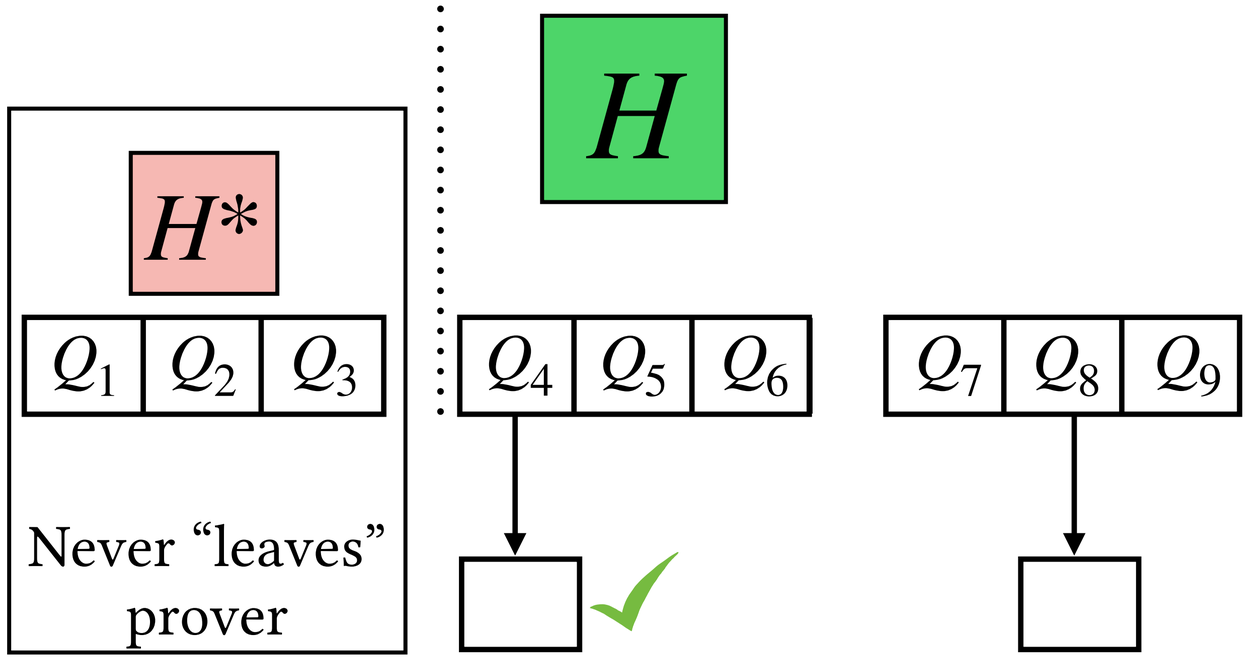
H

$\Pr[V \text{ accepts}] \geq 1/3$

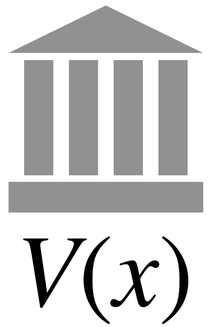
V checks at most $n = 2$ queries

Trimming Resilience





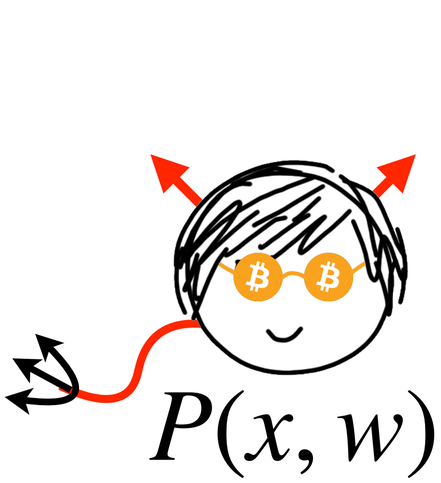
$\Pr[V \text{ accepts}] \geq 1/3$



H

V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$

H^*

Q_1 Q_2 Q_3

Never "leaves" prover

H

Q_4 Q_5 Q_6 Q_7 Q_8 Q_9 Ext w

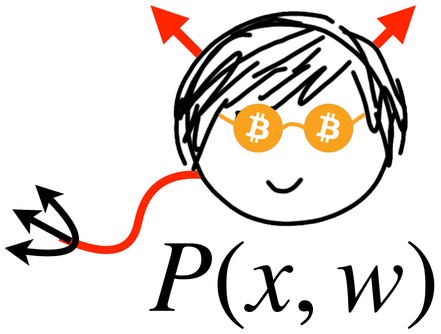
$\Pr[V \text{ accepts}] \geq 1/3$

$V(x)$

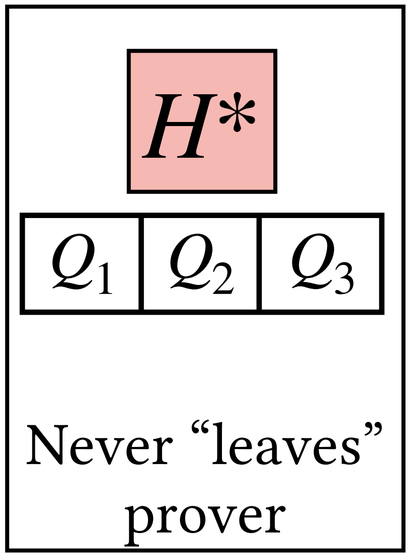
H

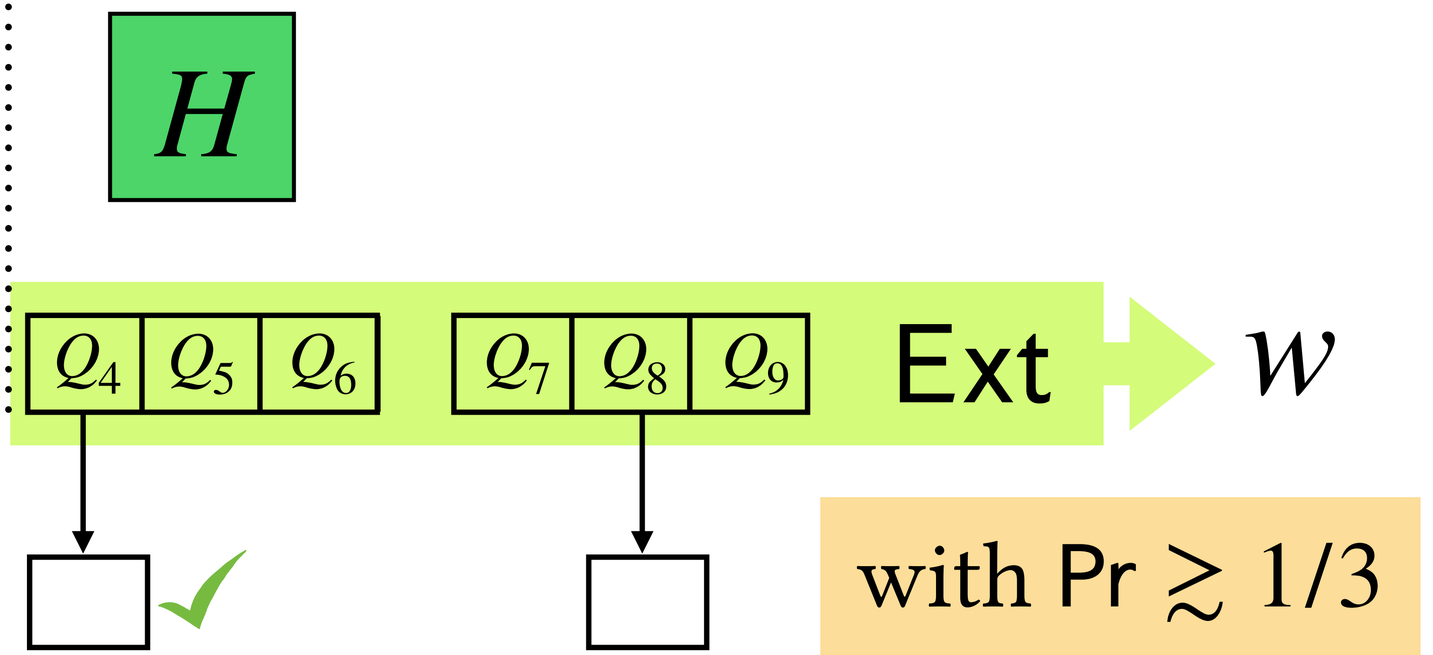
V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$





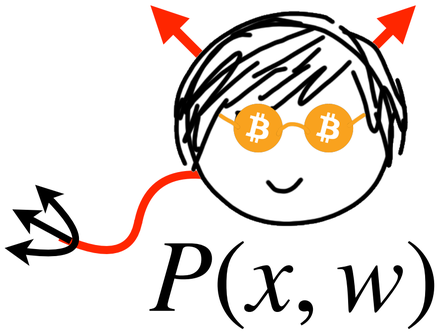
with Pr $\gtrsim 1/3$

Pr[V accepts] $\geq 1/3$

$V(x)$

V checks at most $n = 2$ queries

Trimming Resilience



$P(x, w)$

H^*

Q_7 Q_5 Q_3

Never "leaves" prover

H

(for any 3-partitioning)

Q_8 Q_1 Q_9 Q_2 Q_4 Q_6 Ext w

with Pr $\gtrsim 1/3$

Pr[V accepts] $\geq 1/3$

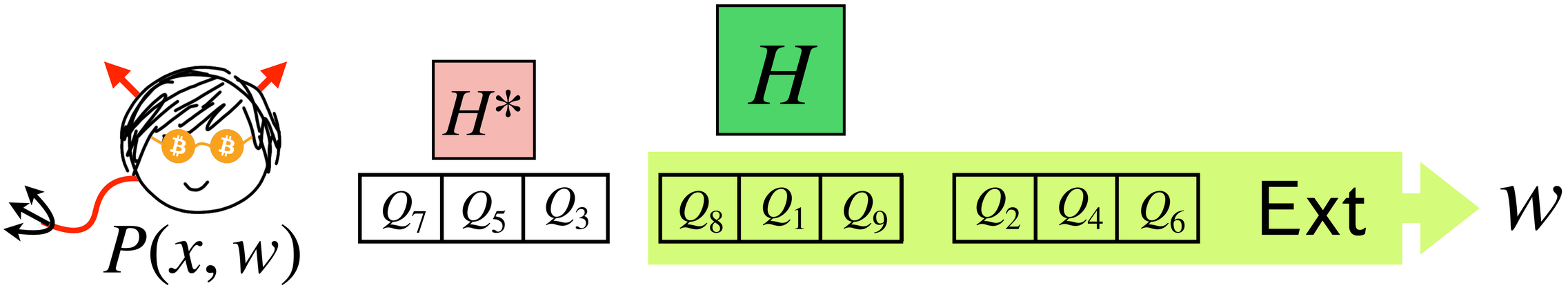
$V(x)$

H

V checks at most $n = 2$ queries

Trimming Resilience

Lemma: For any $n + 1$-partitioning of RO queries, omitting *one* partition still allows extraction

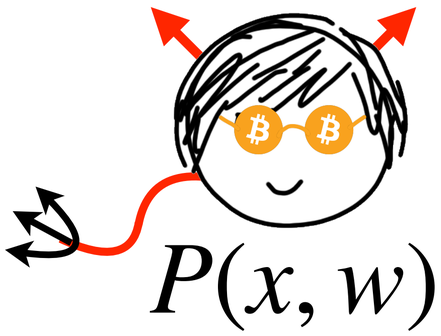


Trimming Resilience

Lemma: For any $n + 1$-partitioning of RO queries, omitting *one* partition still allows extraction

(random)

(w. noticeable probability)



$P(x, w)$

H^*

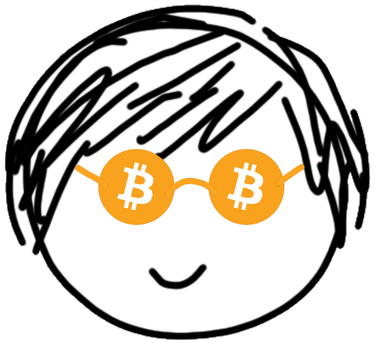
Q_7	Q_5	Q_3

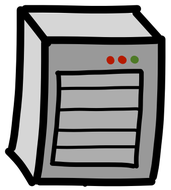
H

Q_8	Q_1	Q_9	Q_2	Q_4	Q_6

Ext → w

Oracle Respecting Distribution

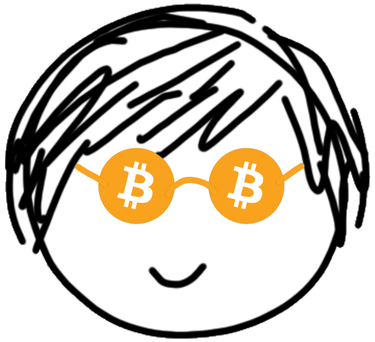




(x, w)

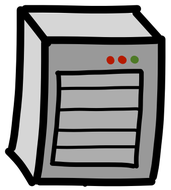
$w_0, w_1, w_2 \leftarrow \mathsf{Share}(w)$

Oracle Respecting Distribution



(x, w)

H



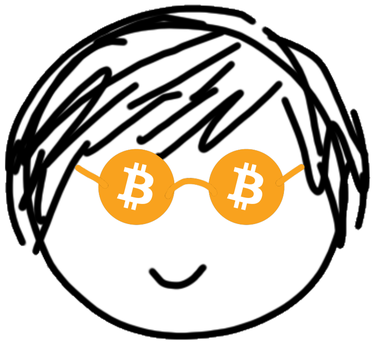
w_1

H

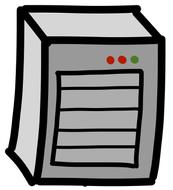
w_2

H

Oracle Respecting Distribution

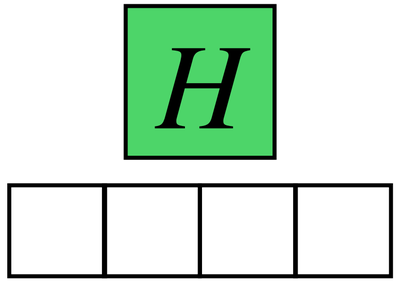


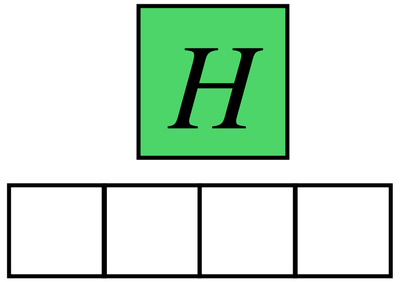
(x, w)

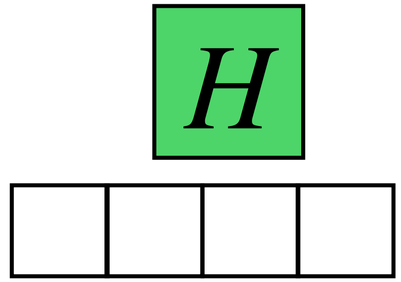


w_1

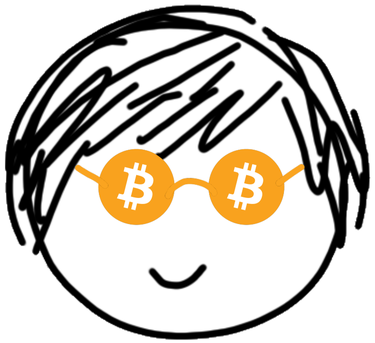
w_2



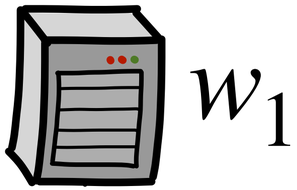


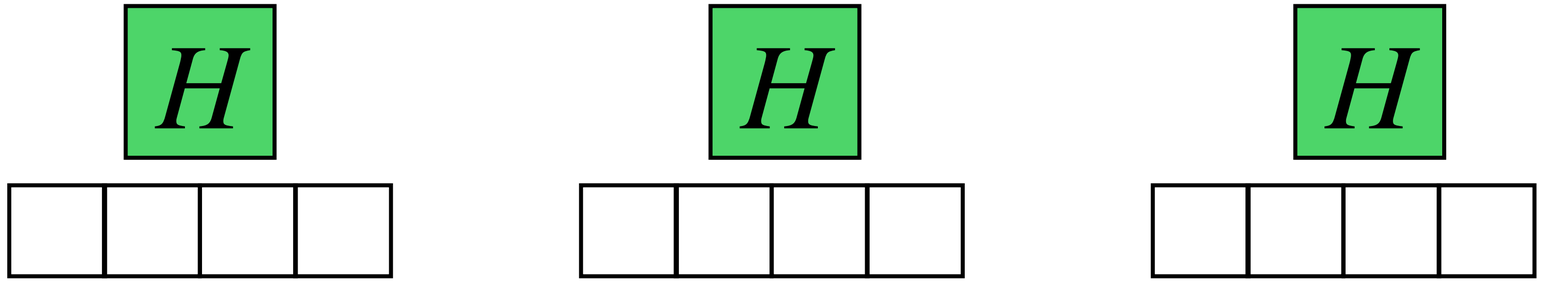


Oracle Respecting Distribution



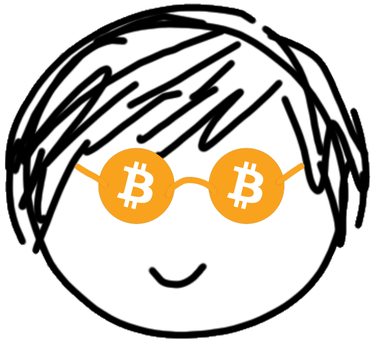
(x, w)



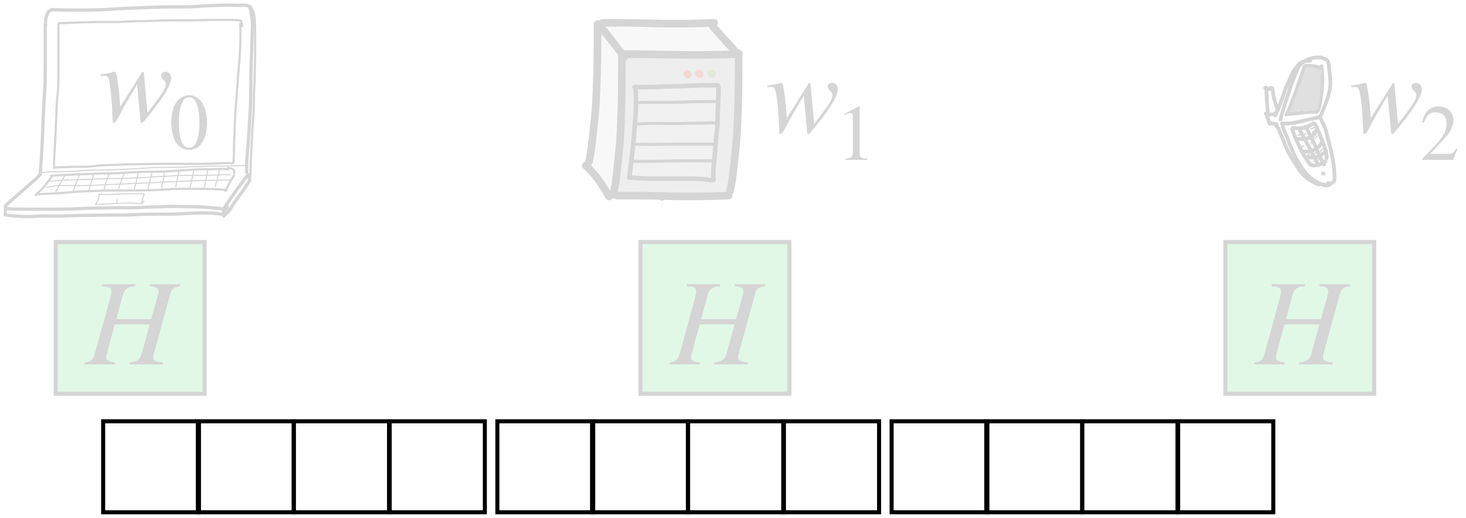


π

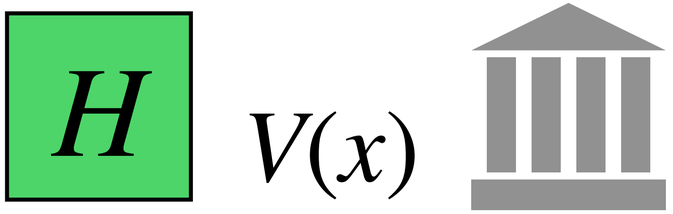
Oracle Respecting Distribution



(x, w)

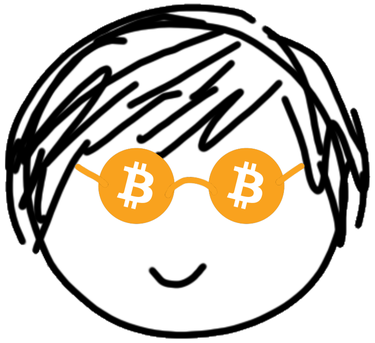


π

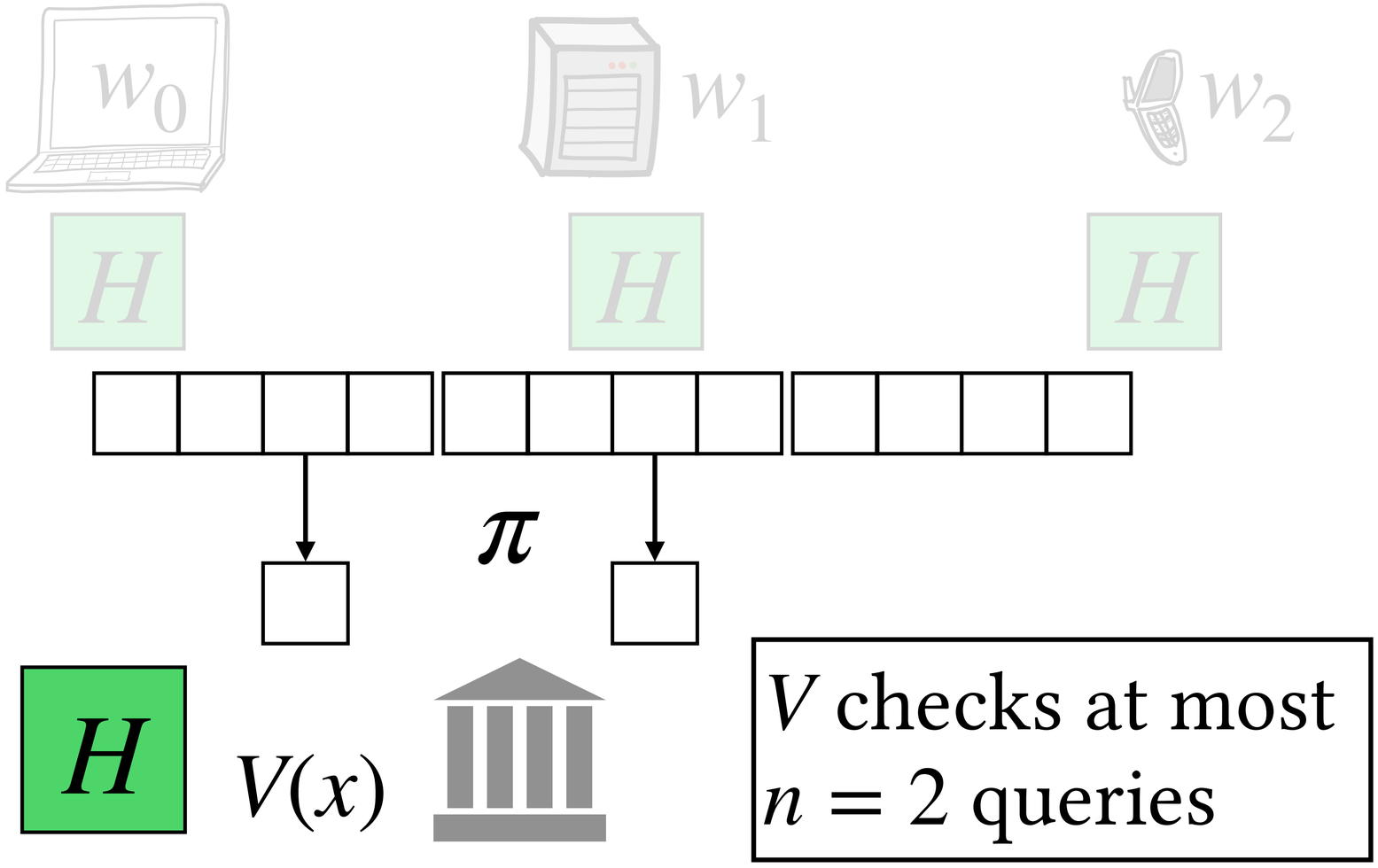


V checks at most $n = 2$ queries

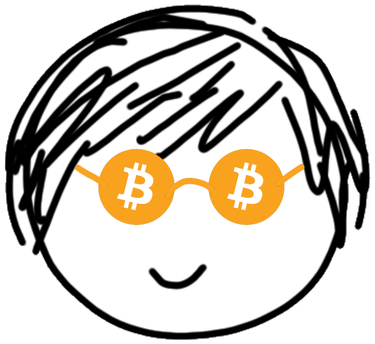
Oracle Respecting Distribution



(x, w)



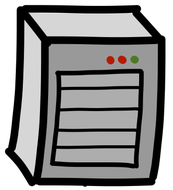
Oracle Respecting Distribution



(x, w)

Natural partitioning

w_0



w_1

w_2

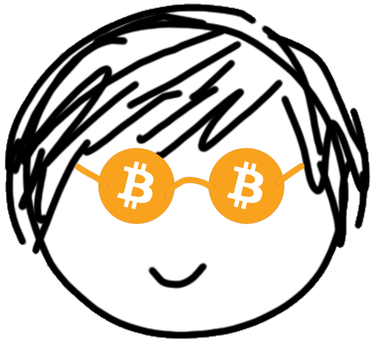
H H H

π

H $V(x)$

V checks at most $n = 2$ queries

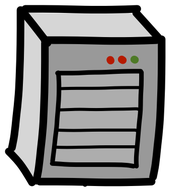
Oracle Respecting Distribution



(x, w)

Natural partitioning

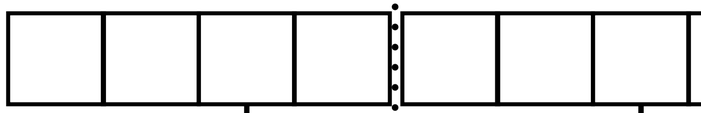
w_0



w_1

w_2

H H H^*

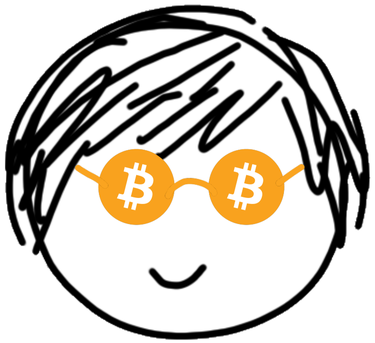


π

H $V(x)$

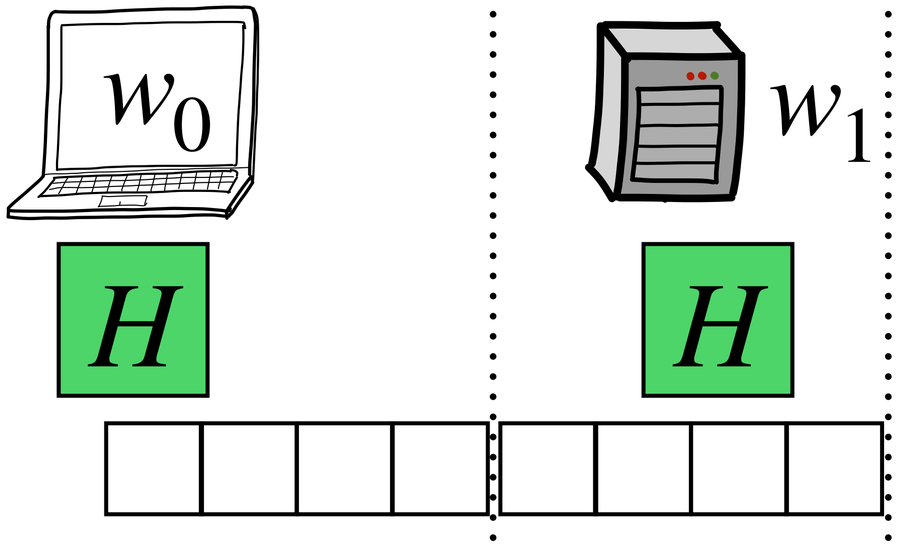
V checks at most $n = 2$ queries

Oracle Respecting Distribution

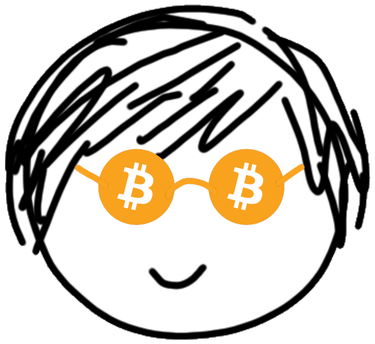


(x, w)

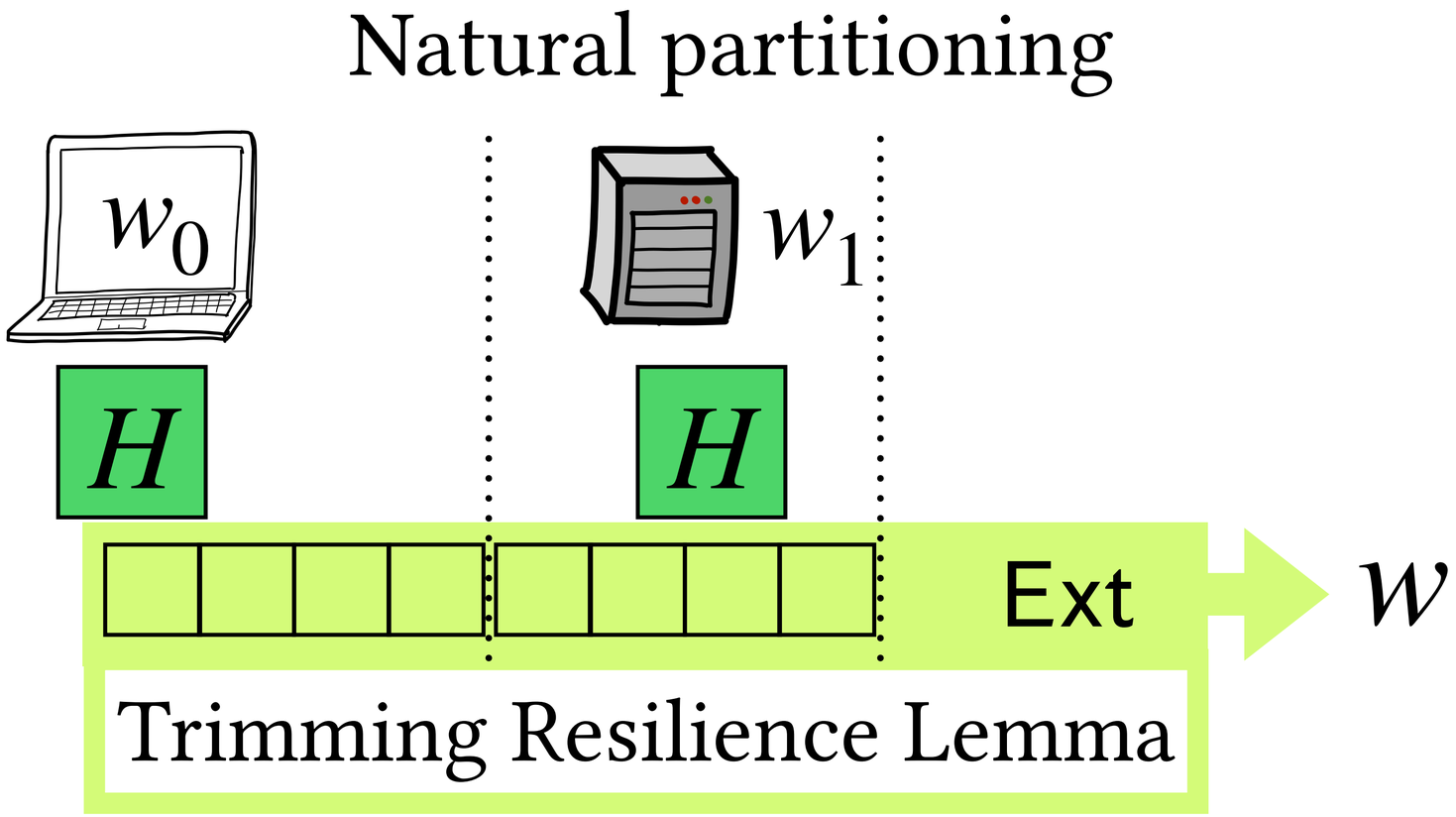
Natural partitioning



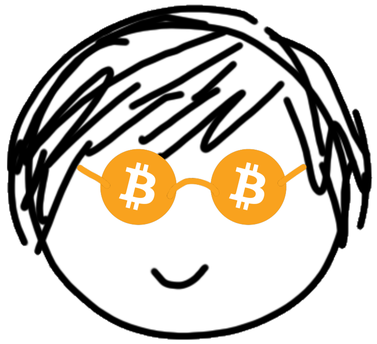
Oracle Respecting Distribution



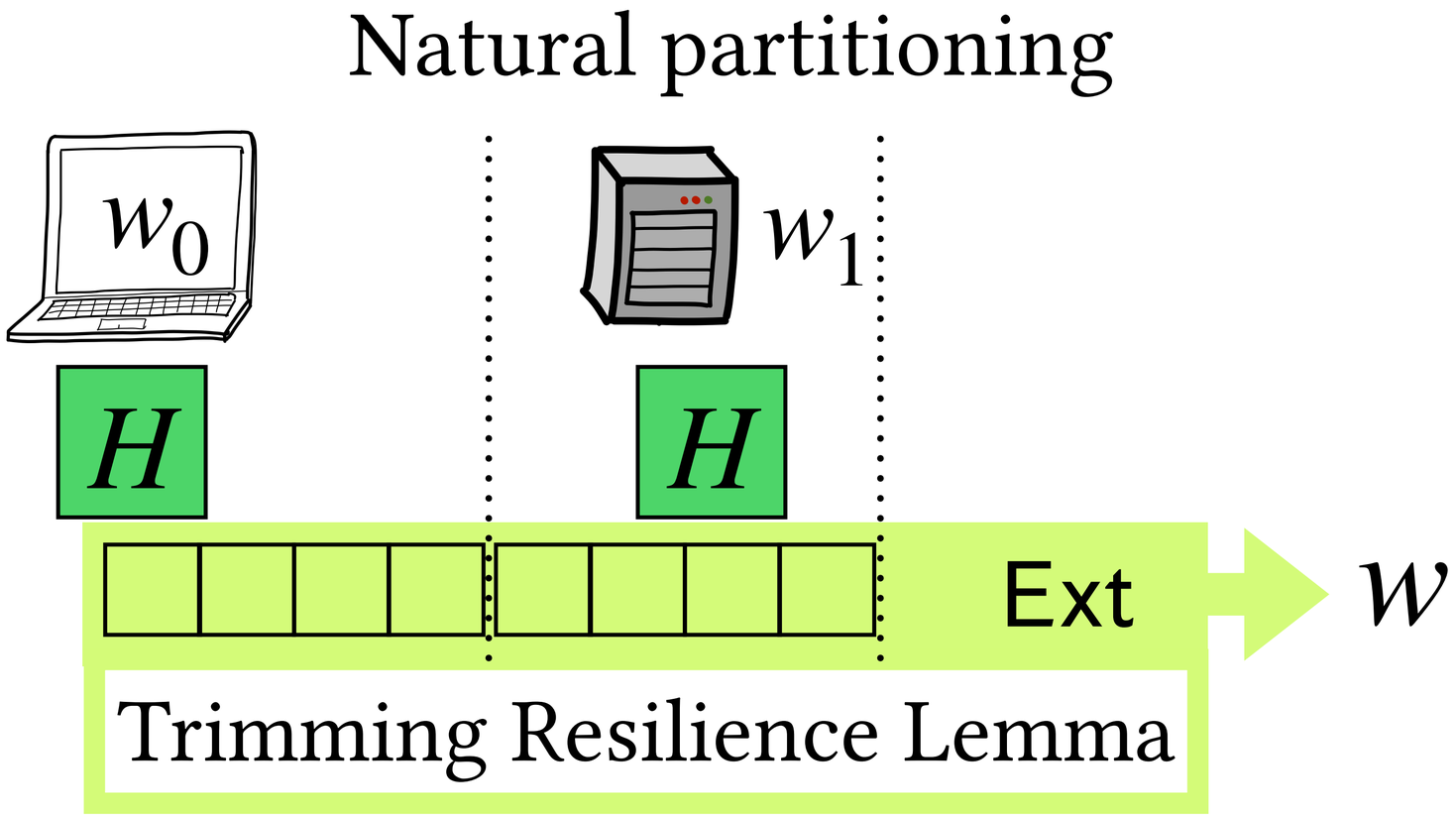
(x, w)



Oracle Respecting Distribution

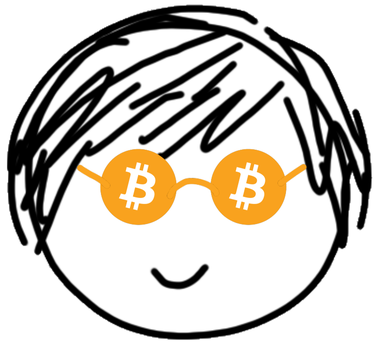


(x, w)

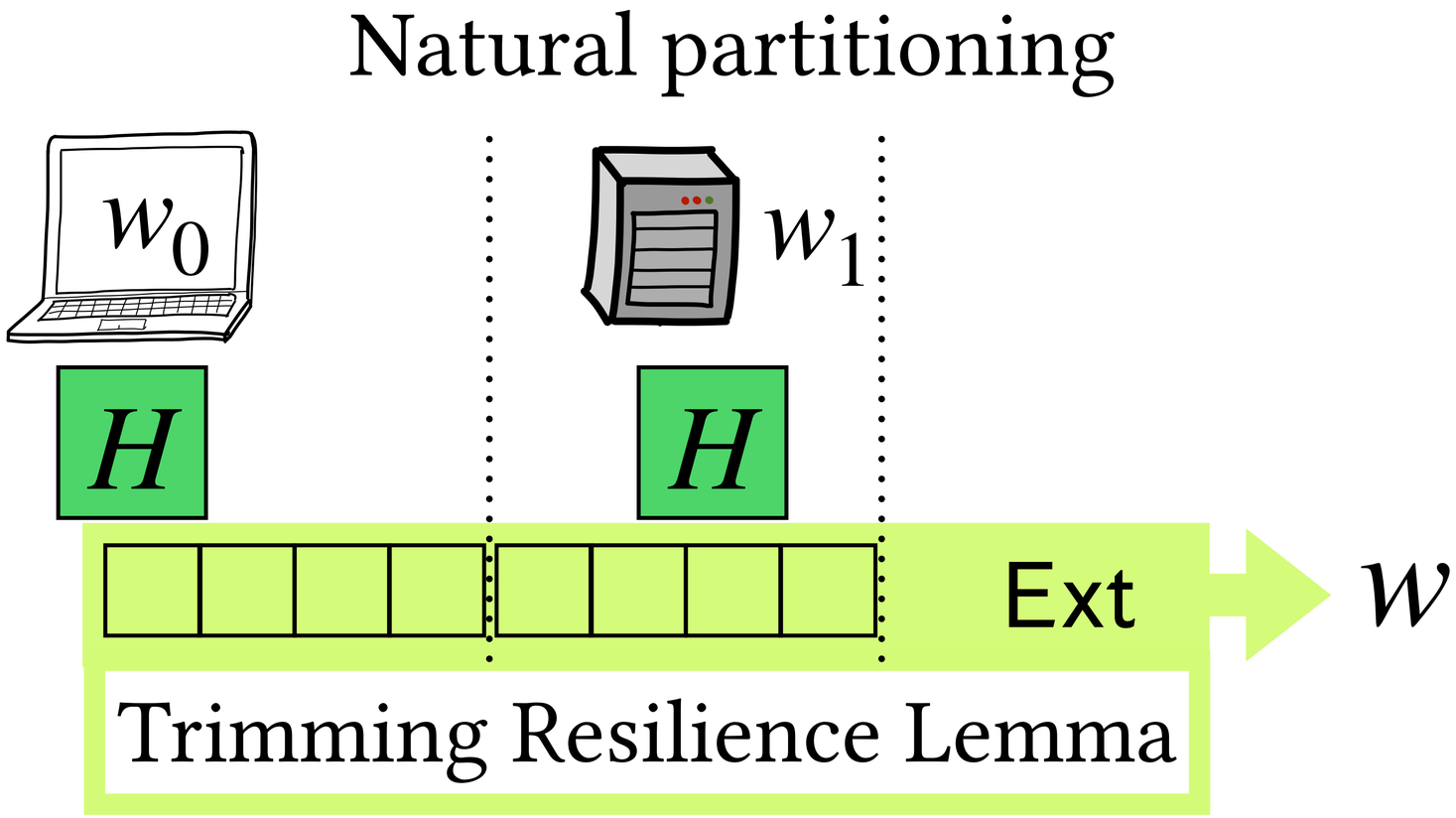


Two views are sufficient to reconstruct the witness

Oracle Respecting Distribution



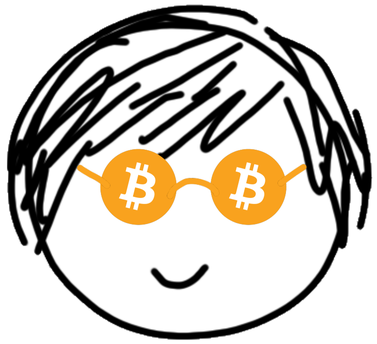
(x, w)



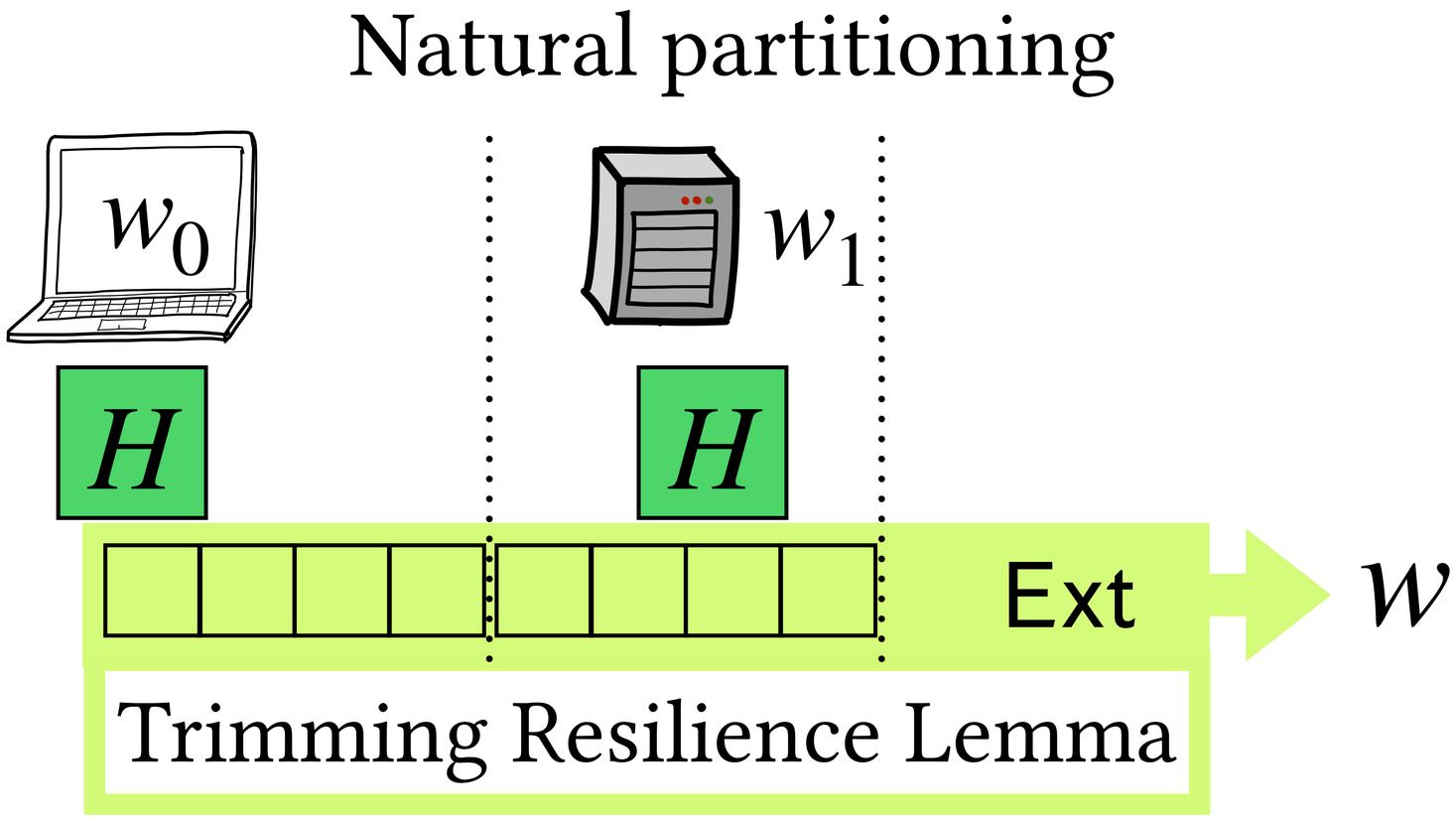
Two views are sufficient to reconstruct the witness

3 party ORD protocol can not withstand 2 passive corruptions

Oracle Respecting Distribution



(x, w)



Two views are sufficient to reconstruct the witness

n party ORD protocol can not withstand n-1 passive corruptions

Caveats

- The n-party protocol must be mapped to a single party algorithm to apply the trimming lemma
- This mapping induces one of two artefacts:
  - Protocol property: Each RO query in the protocol must "traceable" to the party that first made it

    OR
  - NIZK property: $\mathsf{Ext}(\overrightarrow{Q}, \pi)$ does not actually need $H(\overrightarrow{Q})$

Honest Majority?

- Previous technique can not be directly extended for fewer than $n - O(1)$ corruptions

  $\exists$ NIZKPoK of DLog π s.t. for any constant c, $\exists$ n-party ORD protocol to securely compute π with tolerance to $c \cdot n$ malicious corruptions
  Caveat: only beats trivial solution when $n > \kappa$

- However, ORD protocols for NIZKs where Ext needs a *single private query* of P seem unlikely **for even one corruption**

Conclusion

- We showed that n-party protocols to securely compute certain hash-based signatures/NIZKs can not make blackbox use of the same hash function
  - Includes MPC-in-the-head, Fischlin/Unruh/Pass/Ks22 transform, PCPs/IOPs
- Dist. NIZK Verifier must depend on #parties—could indicate that thresh. signature must grow with #signers?

Thanks!

Thanks **Eysa Lee** for