Threshold ECDSA in Three Rounds

Jack Doerner — TECHNION Israel Institute of Technology

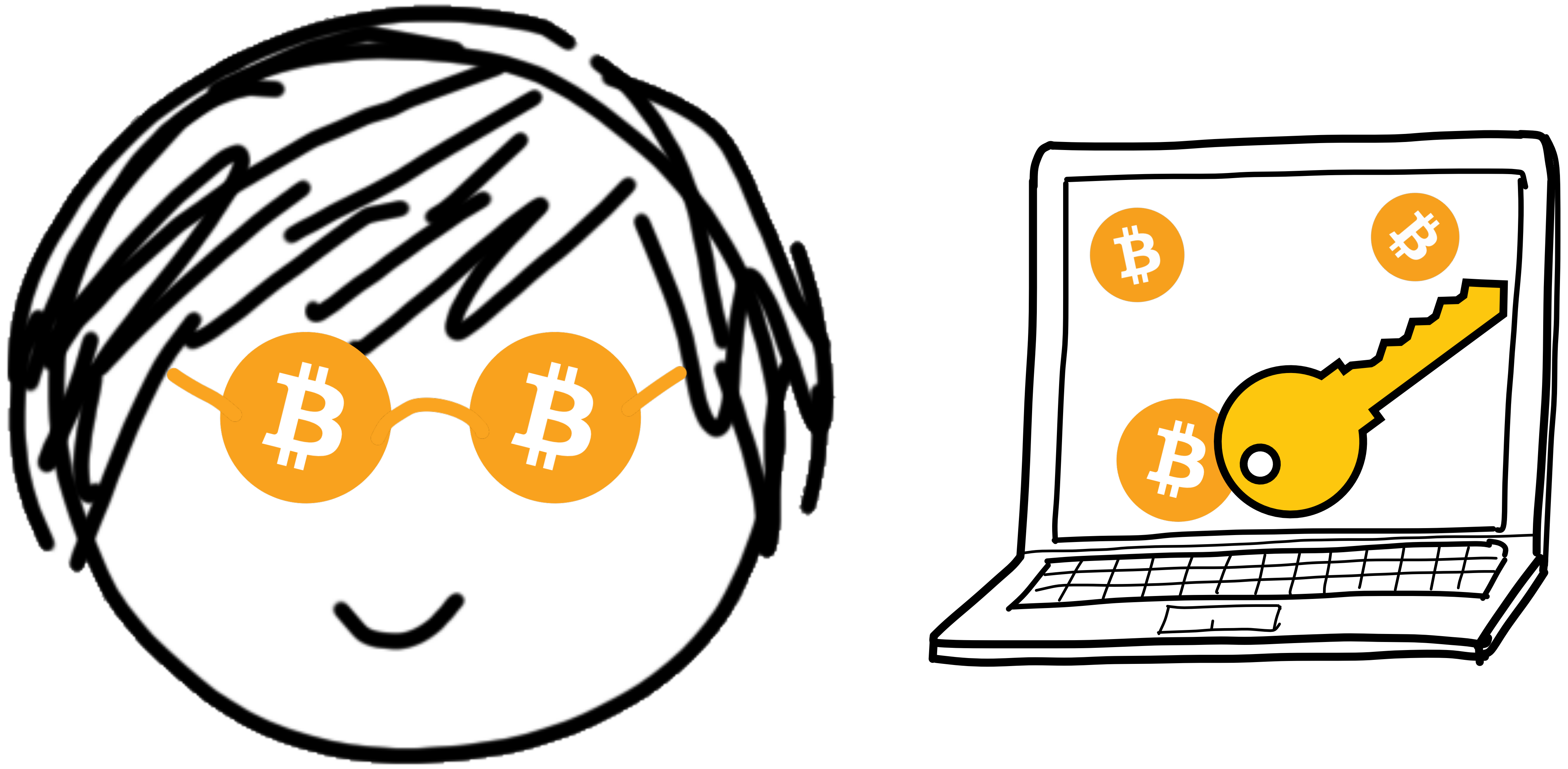
Yashvanth Kondi — SILENCE LABORATORIES

Eysa Lee — BROWN

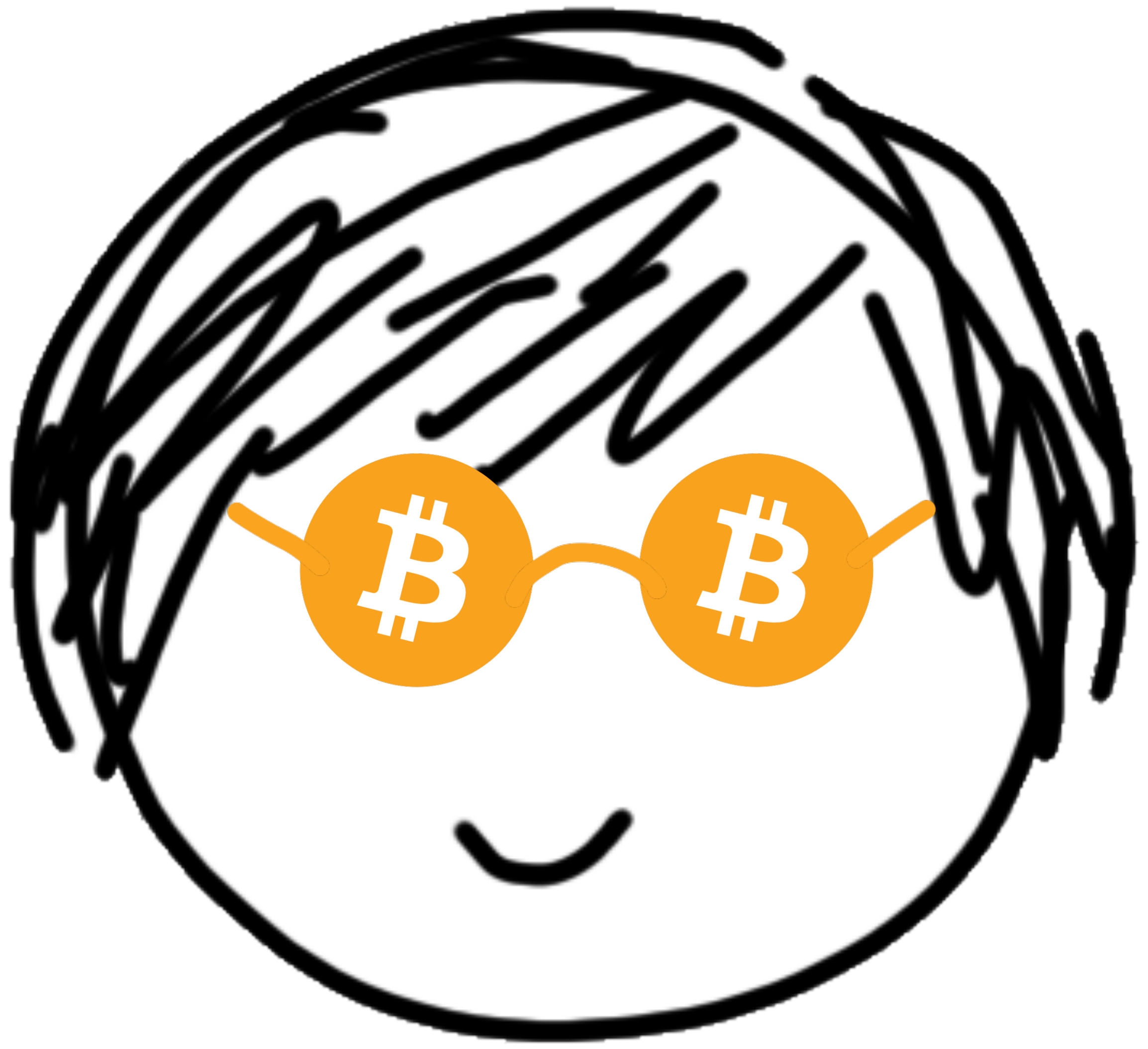
abhi shelat — Northeastern University

dkls.info

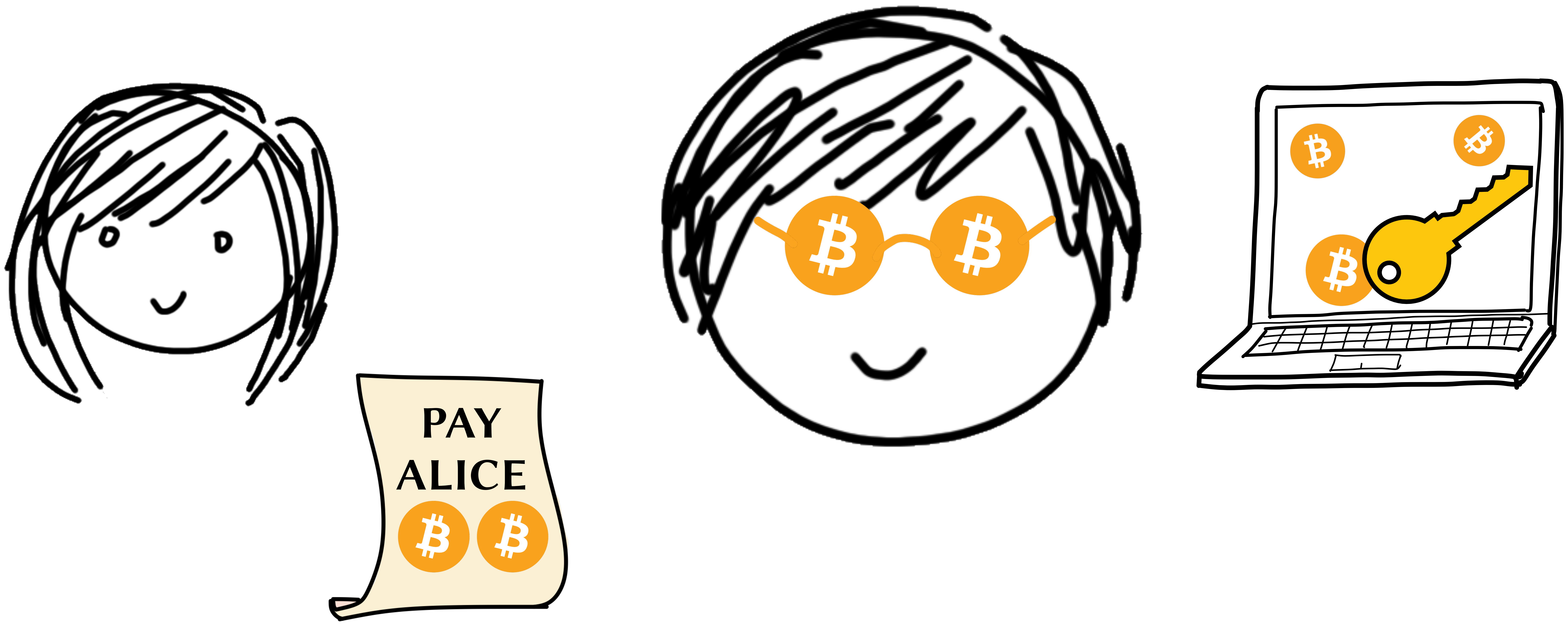
Ballad of Ɓitcoin Bob



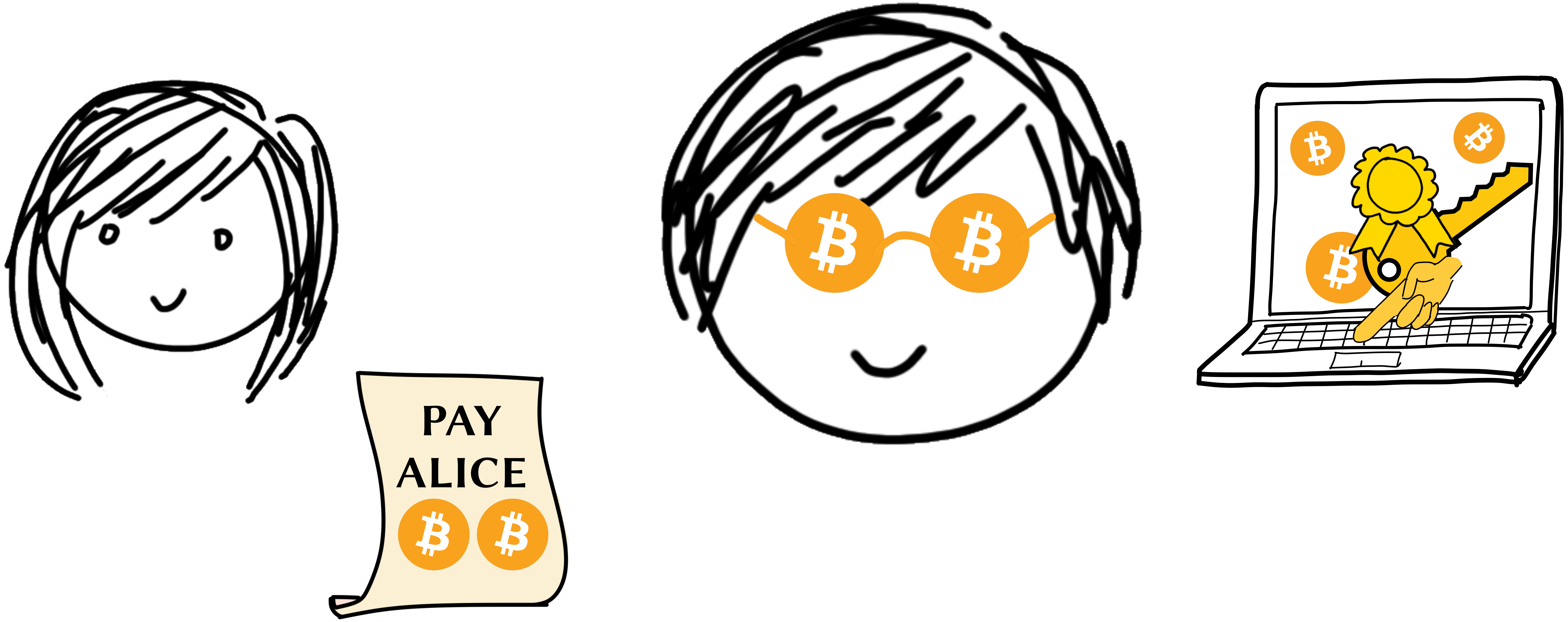
Ballad of ₿itcoin Bob



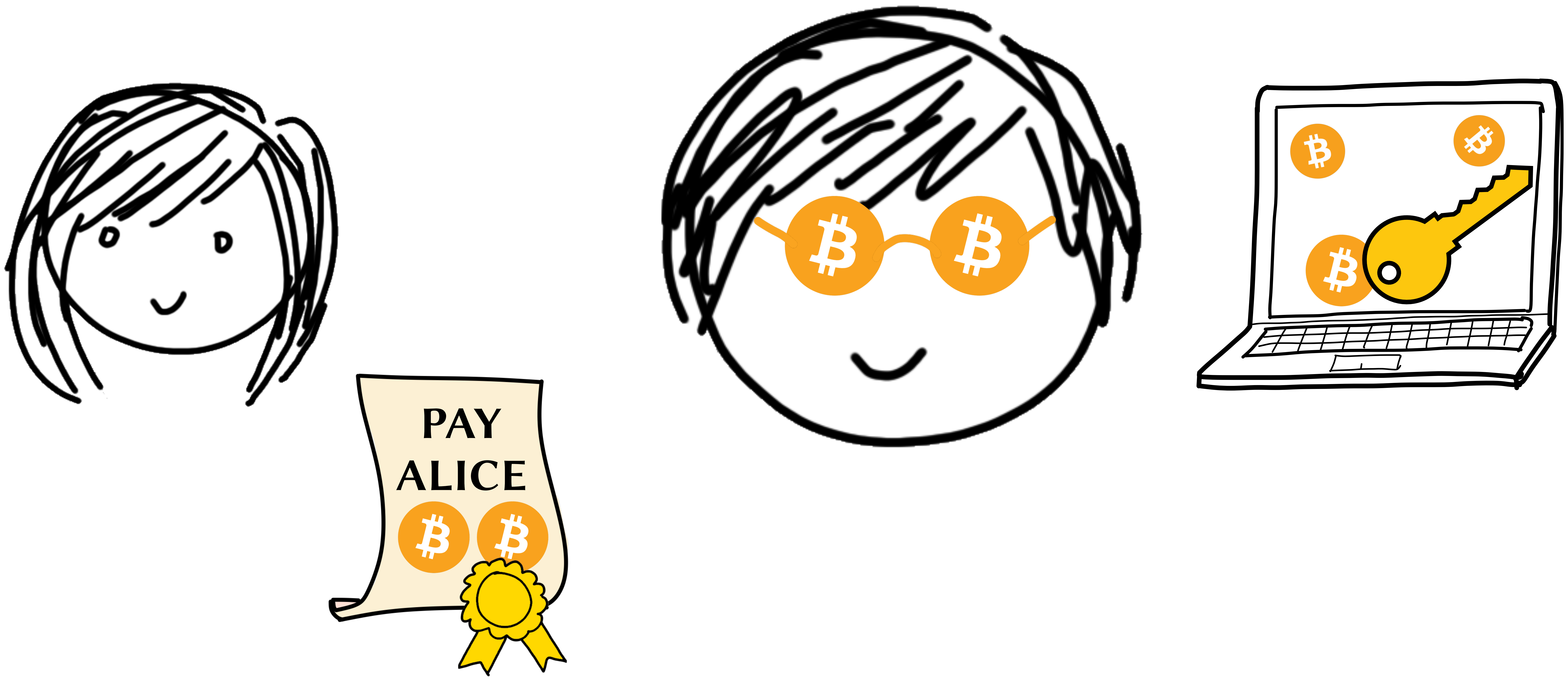
Ballad of ฿itcoin Bob



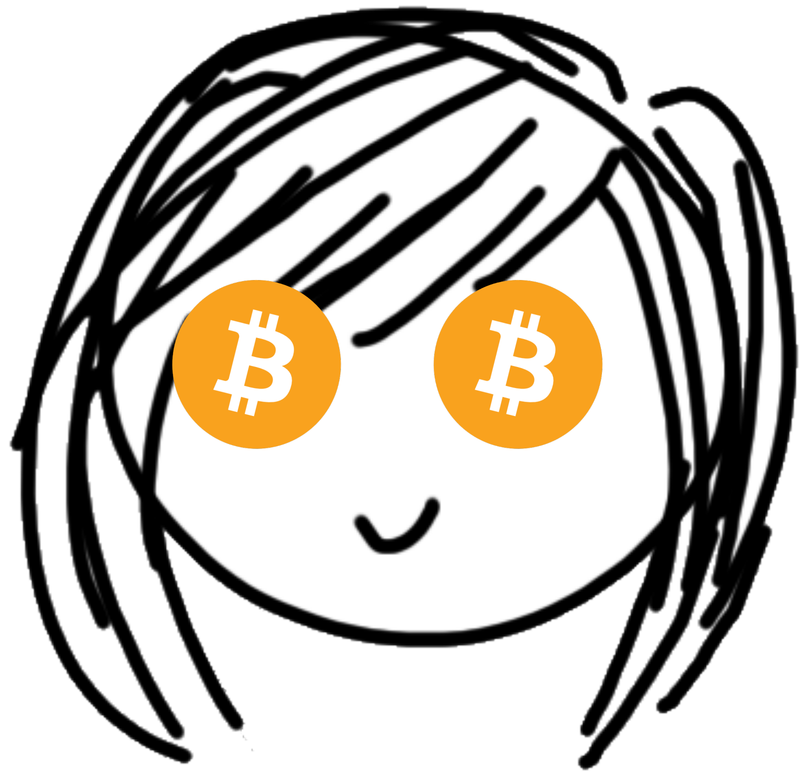
Ballad of Ƀitcoin Bob

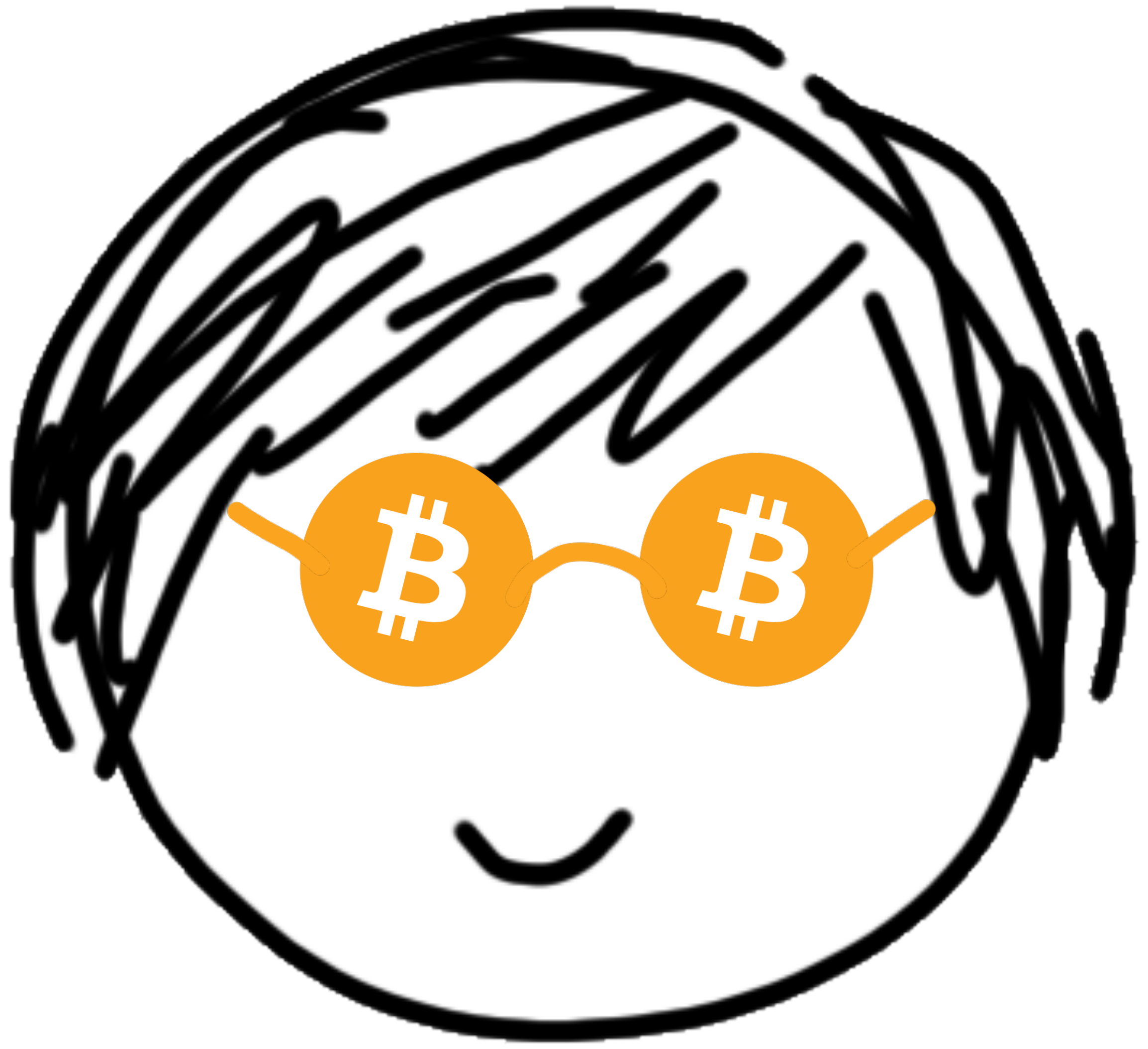


Ballad of ₿itcoin Bob

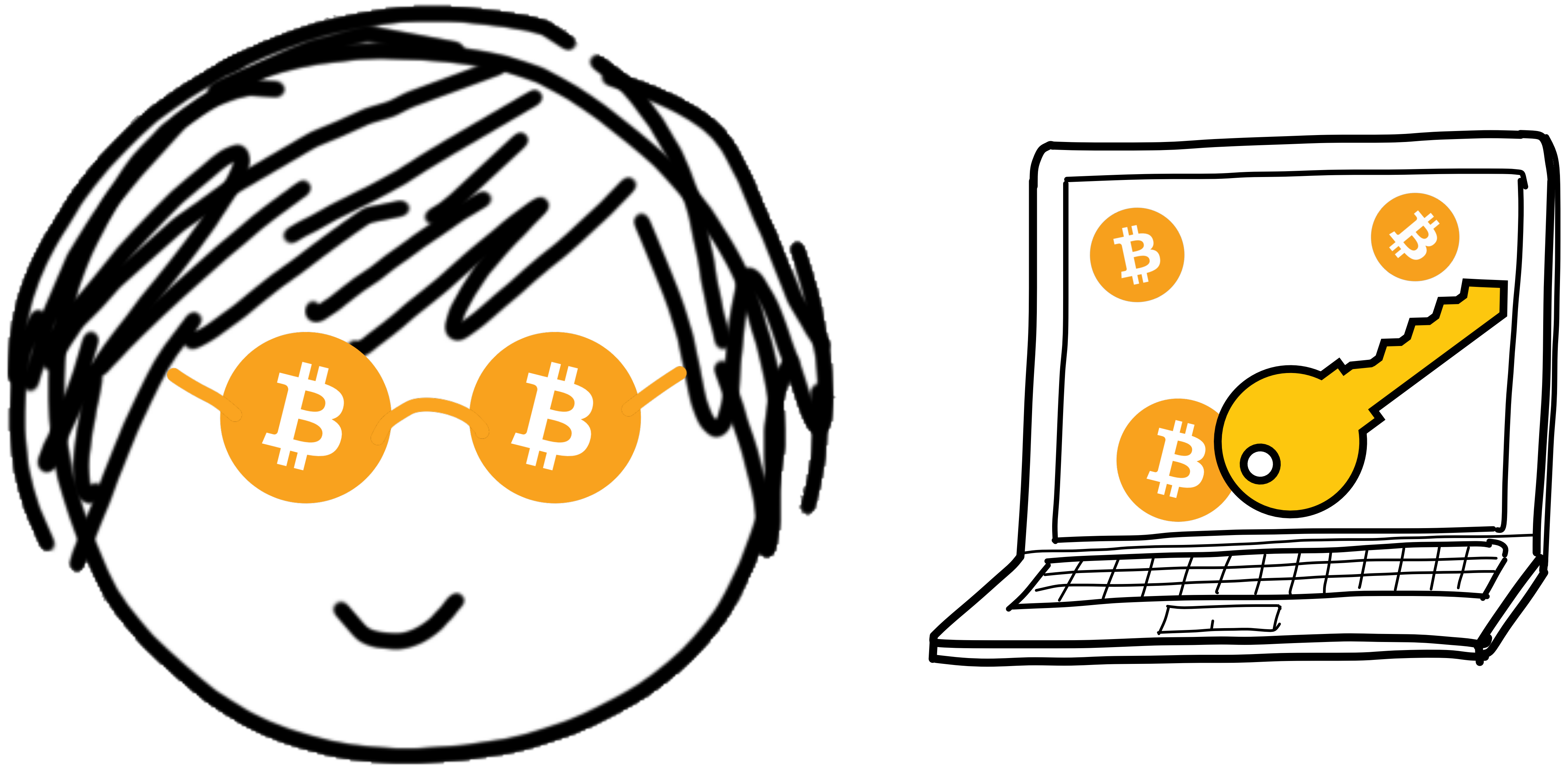


Ballad of ₿itcoin Bob

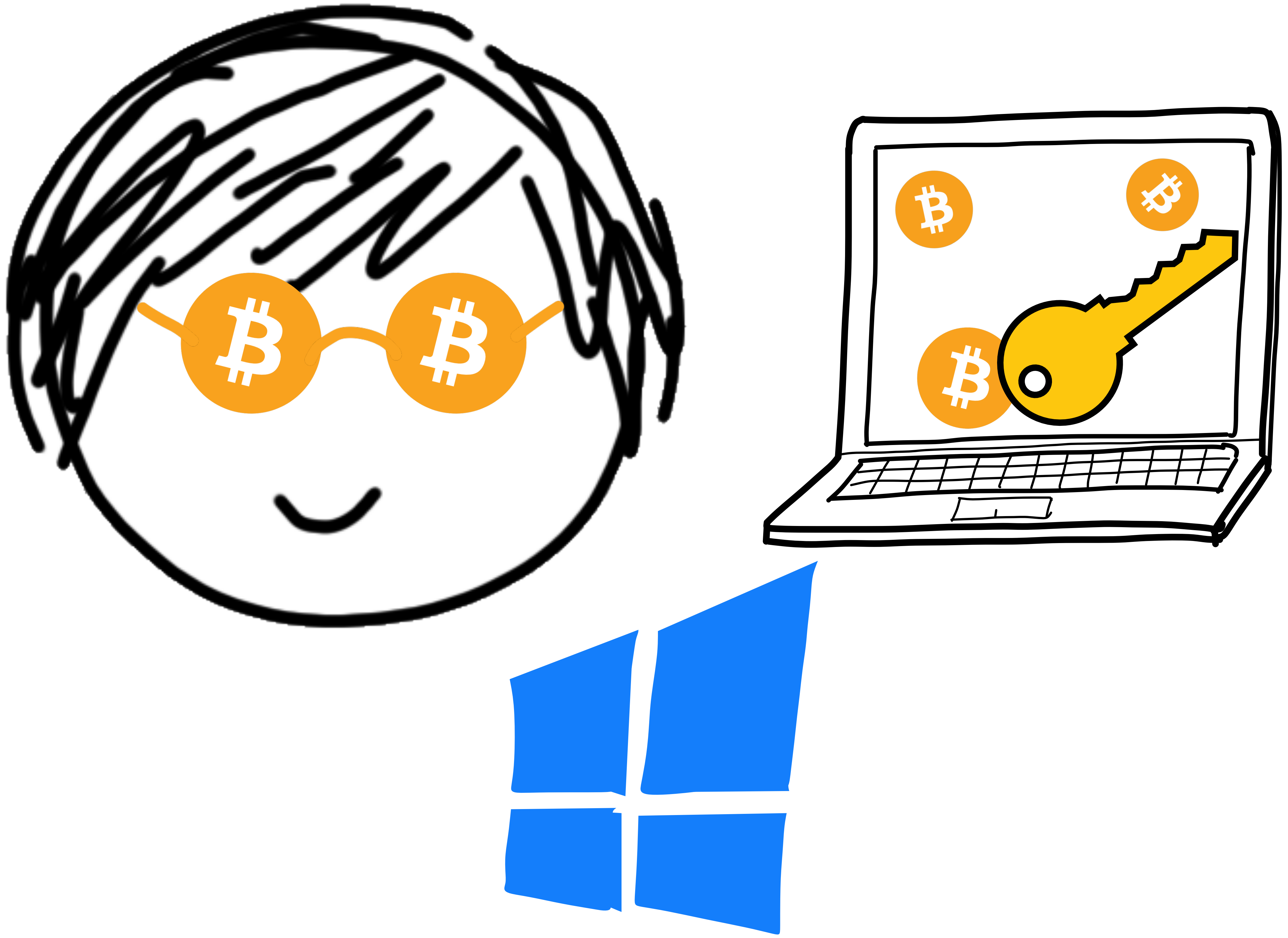




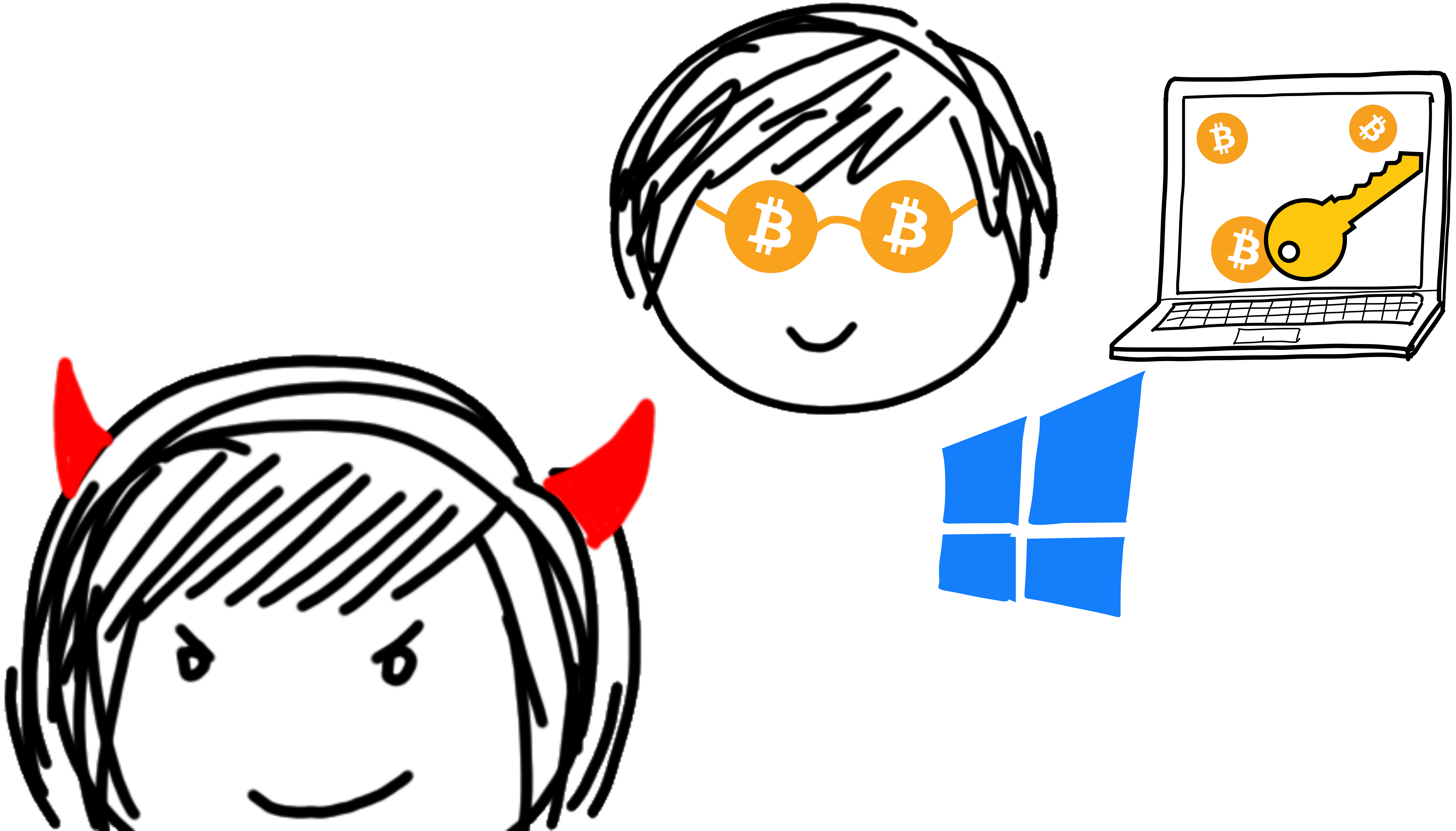
Ballad of Ƀitcoin Bob



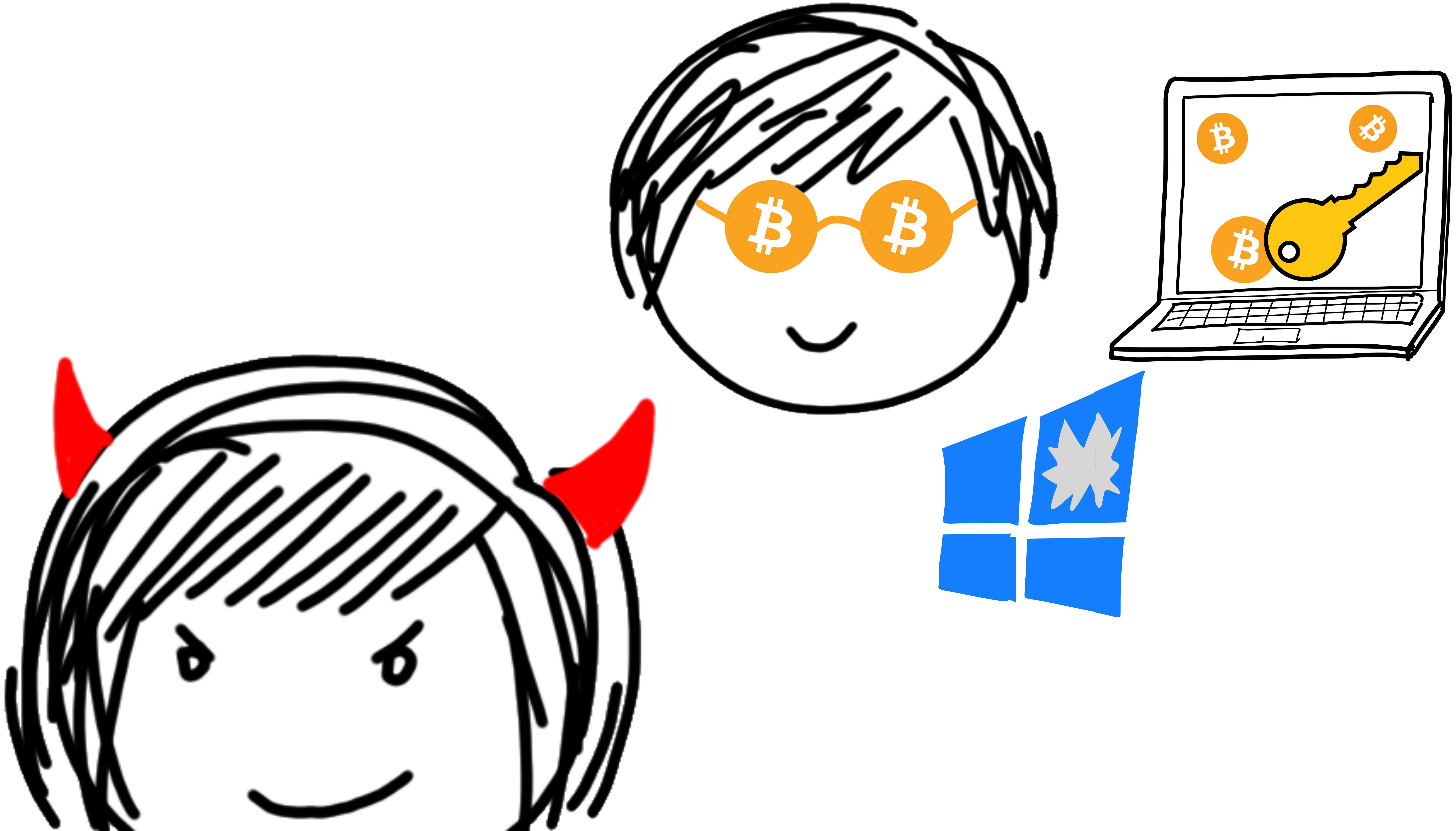
Ballad of ₿itcoin Bob



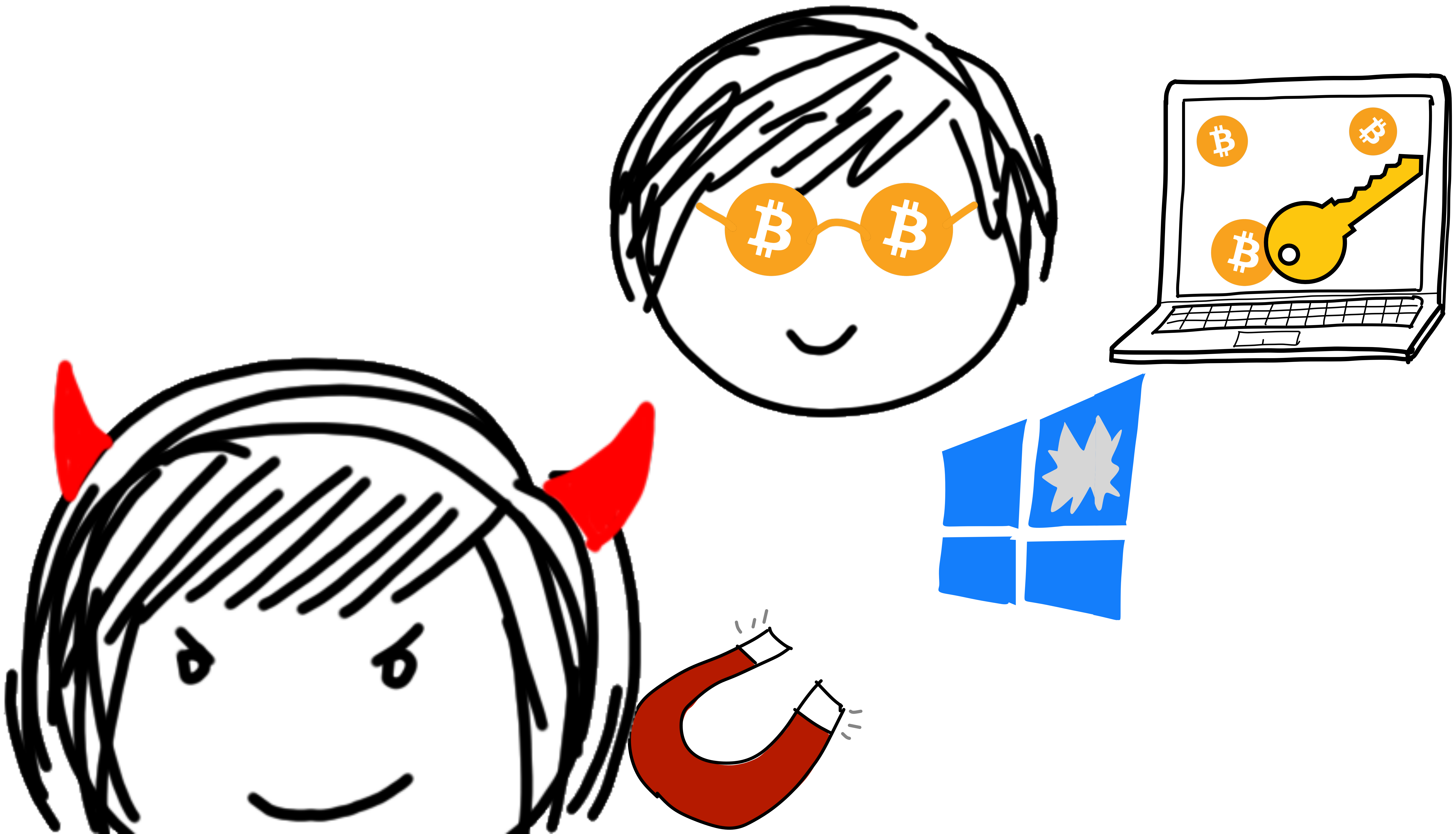
Ballad of ₿itcoin Bob



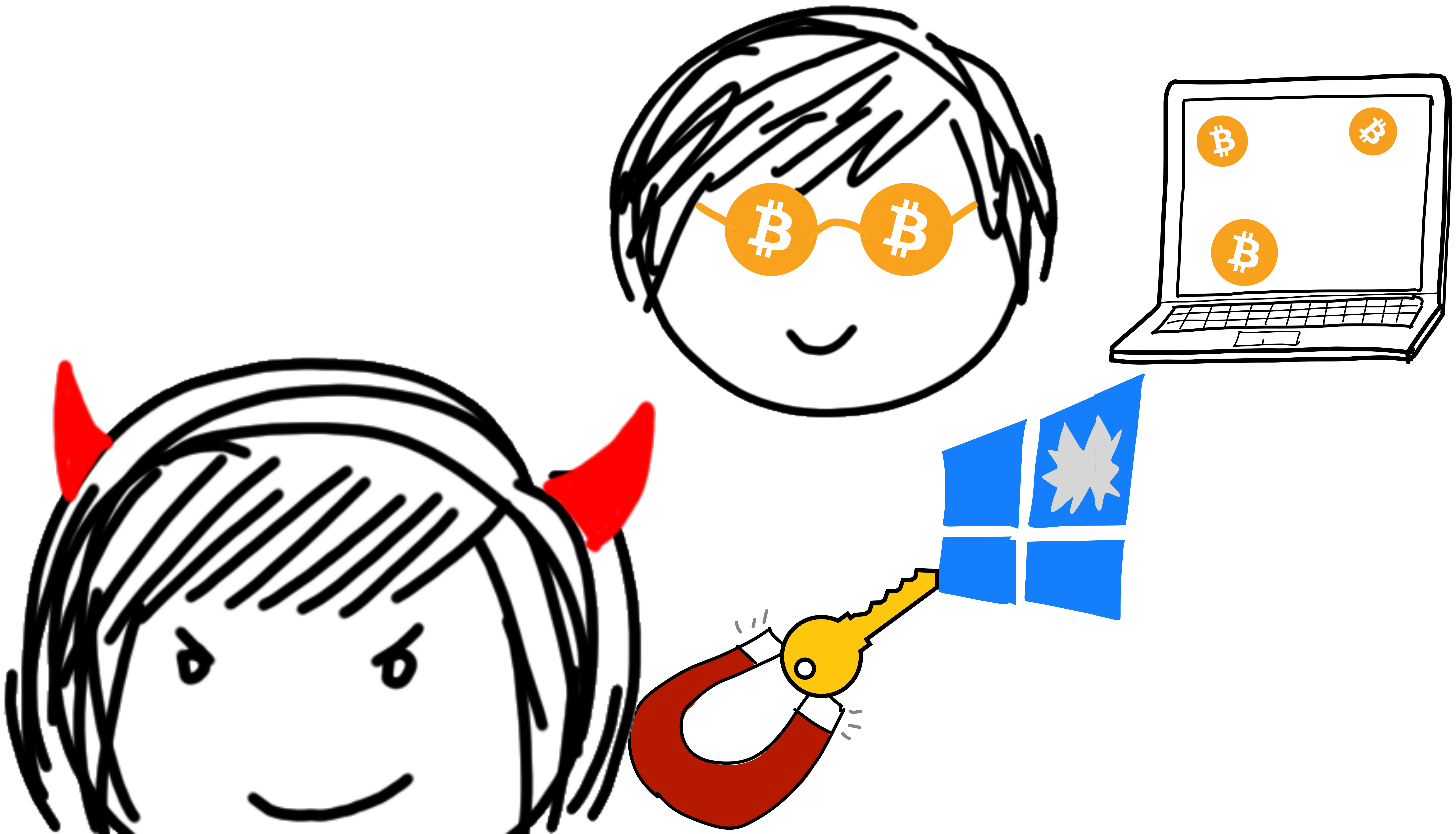
Ballad of ₿itcoin Bob



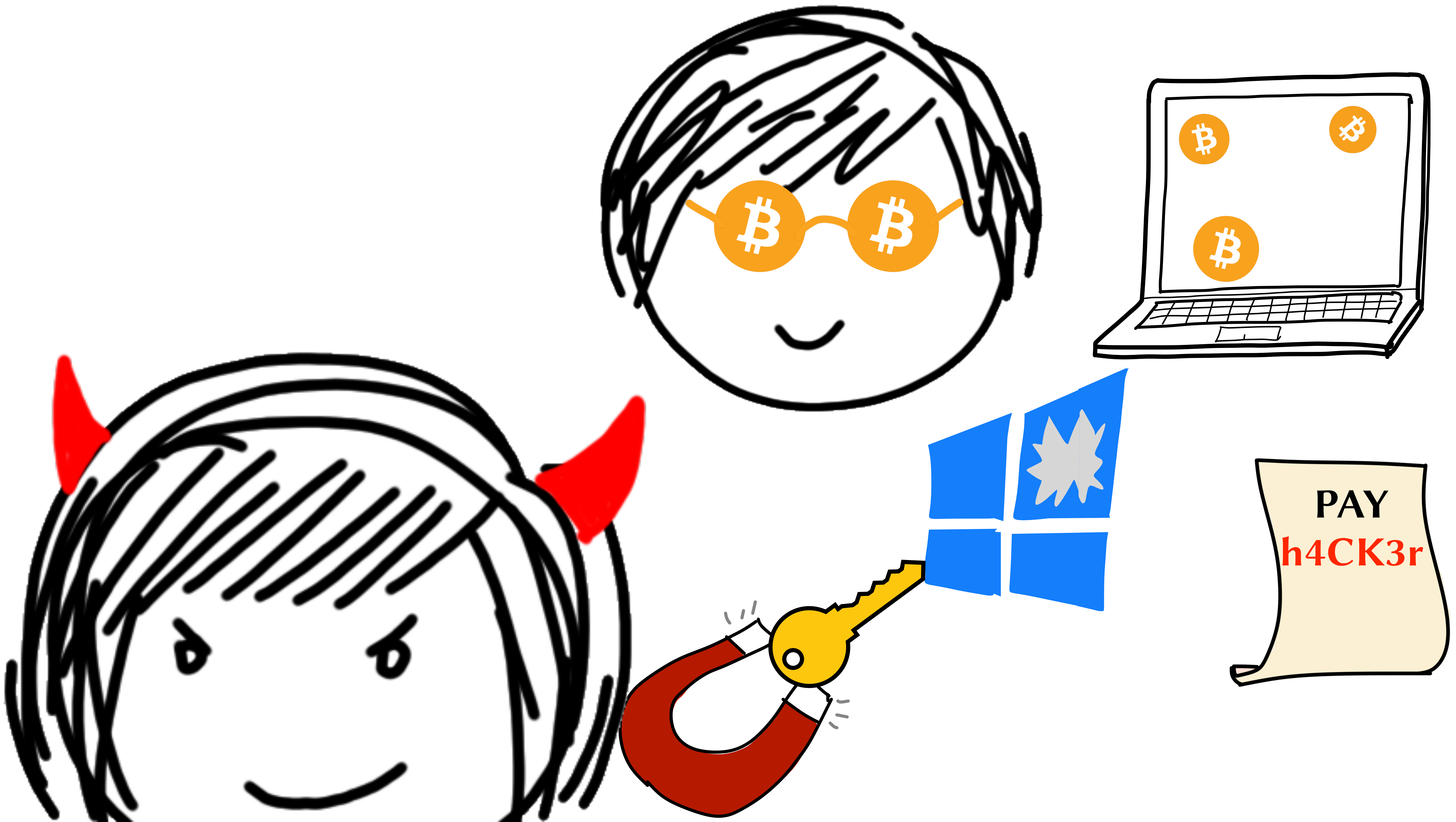
Ballad of Ƀitcoin Bob



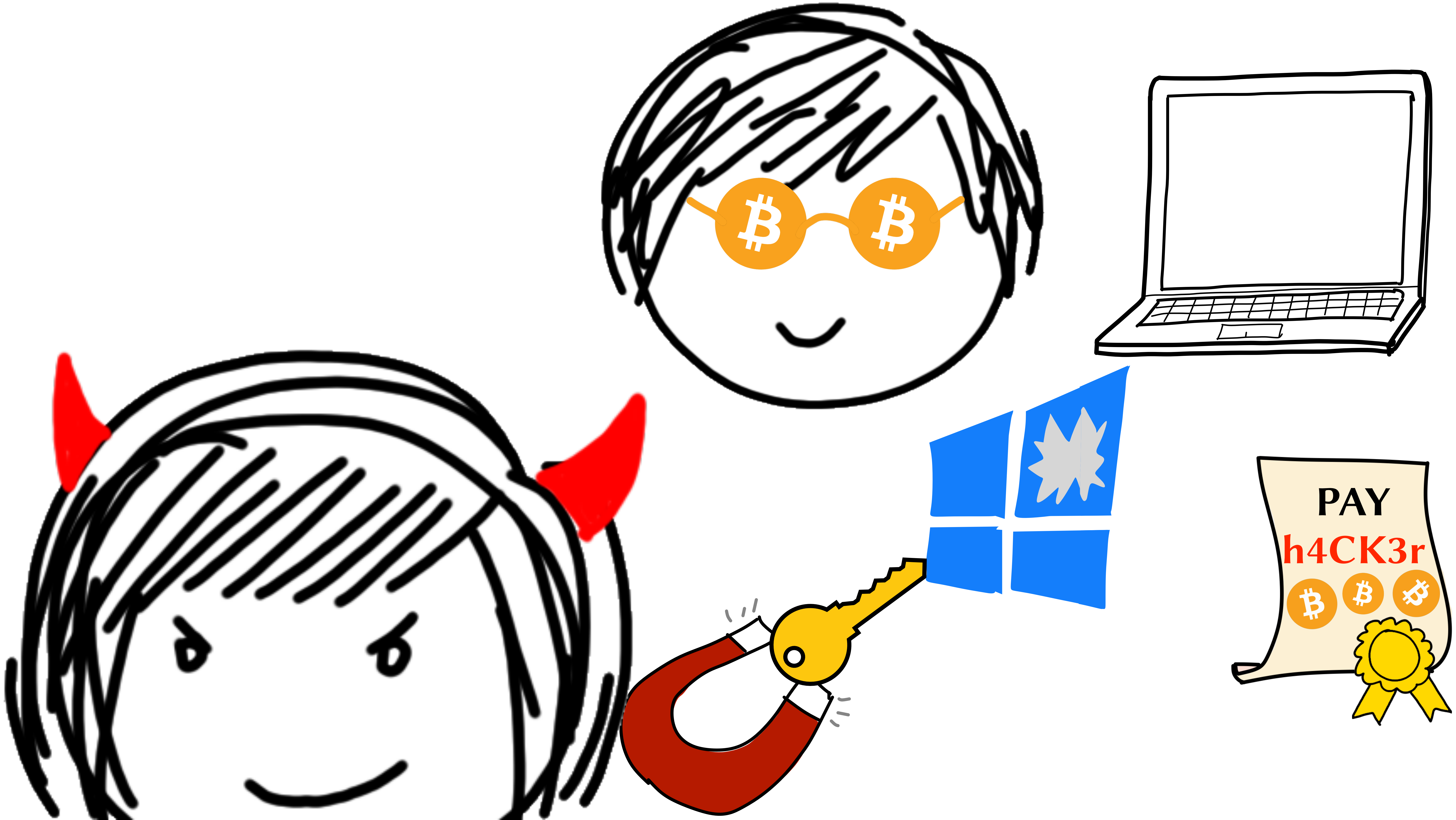
Ballad of ₿itcoin Bob



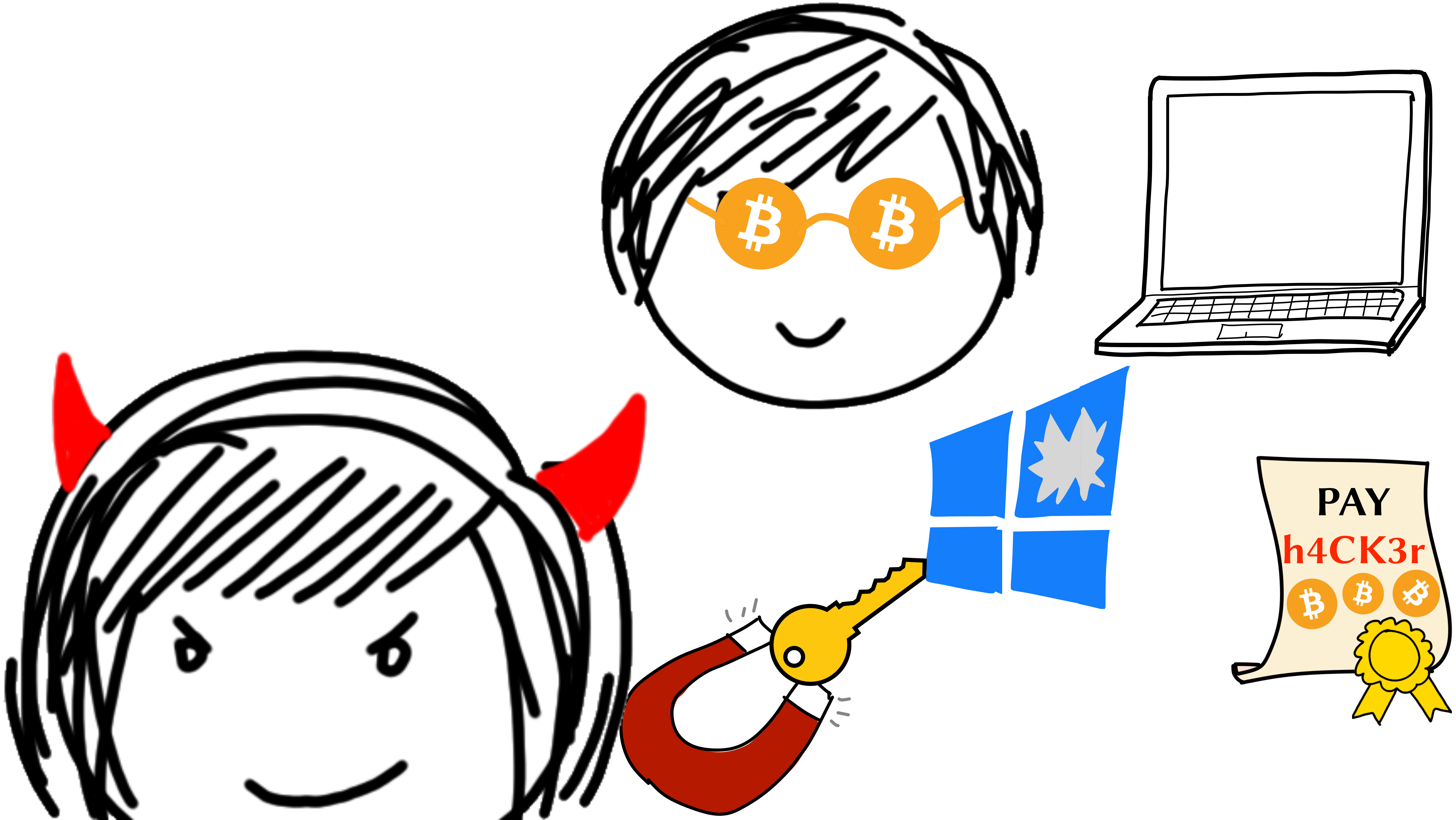
Ballad of Ƀitcoin Bob



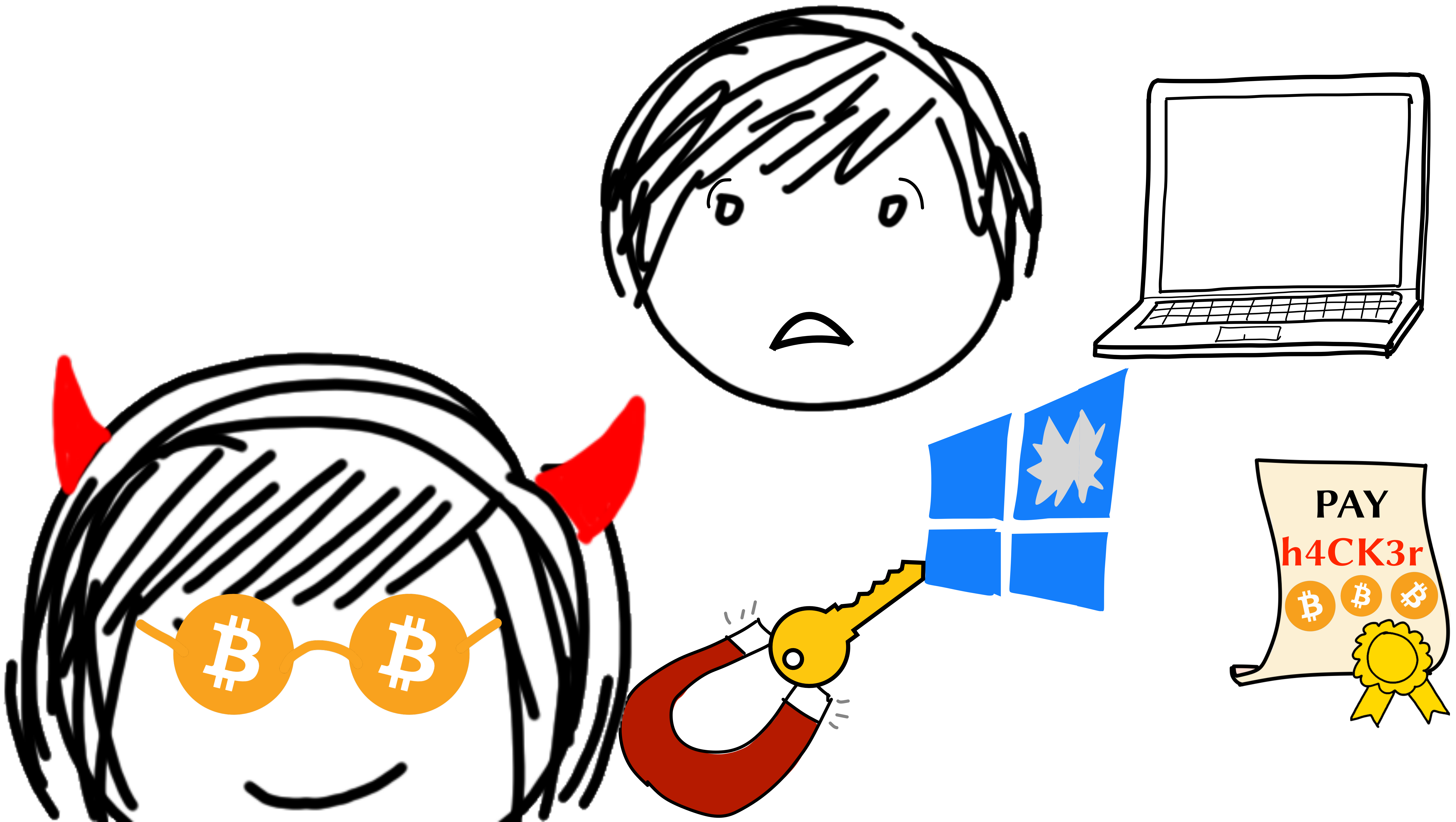
Ballad of Ƀitcoin Bob



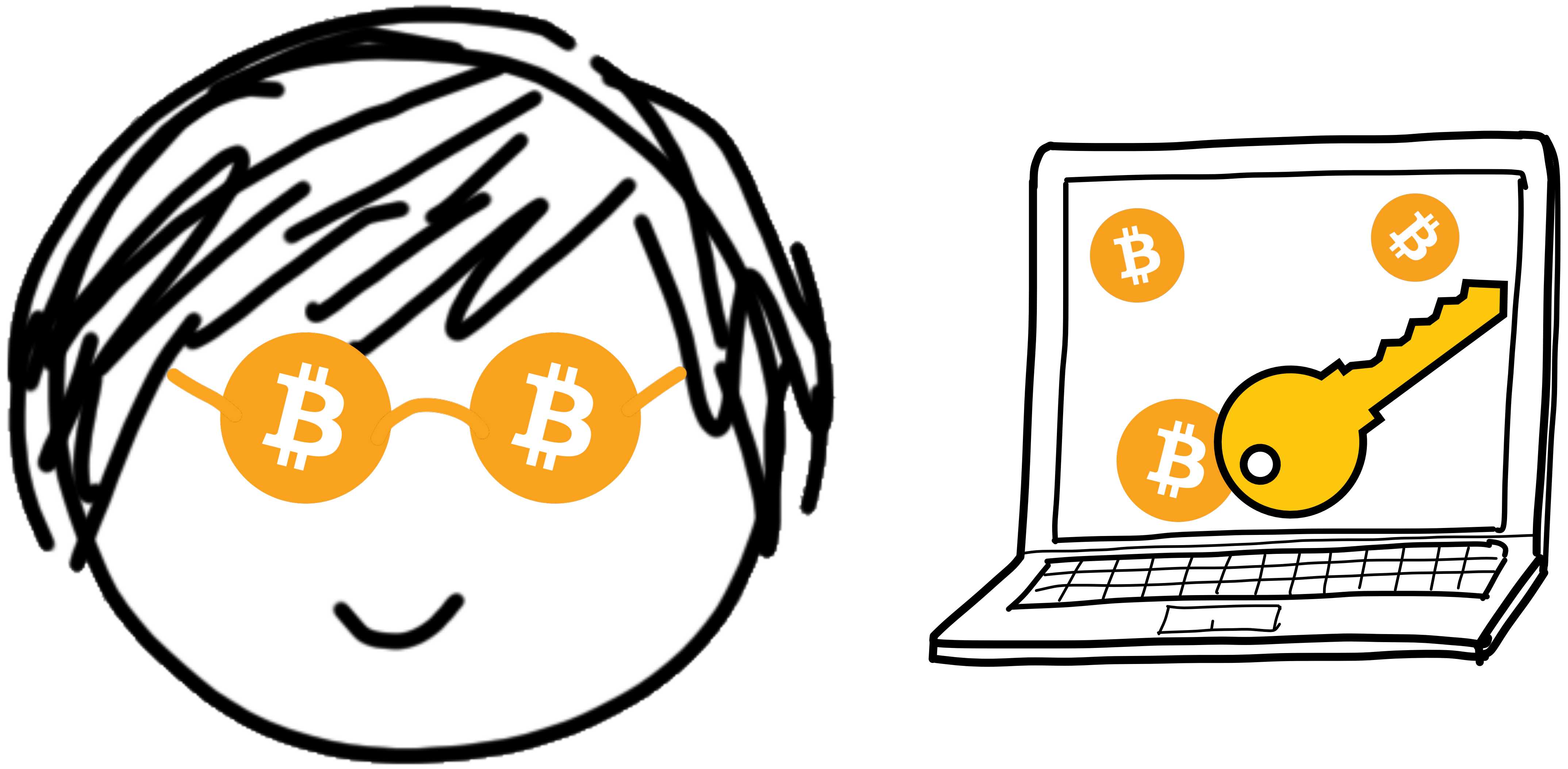
Ballad of ₿itcoin Bob



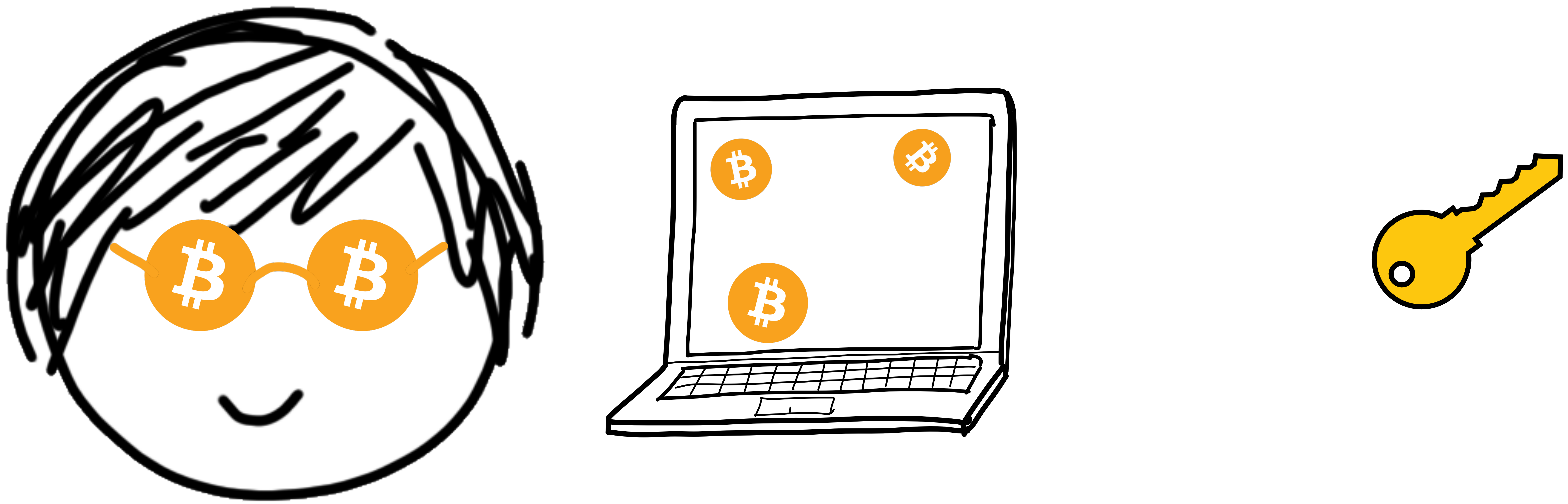
Ballad of ₿itcoin Bob



Ballad of ₿itcoin Bob

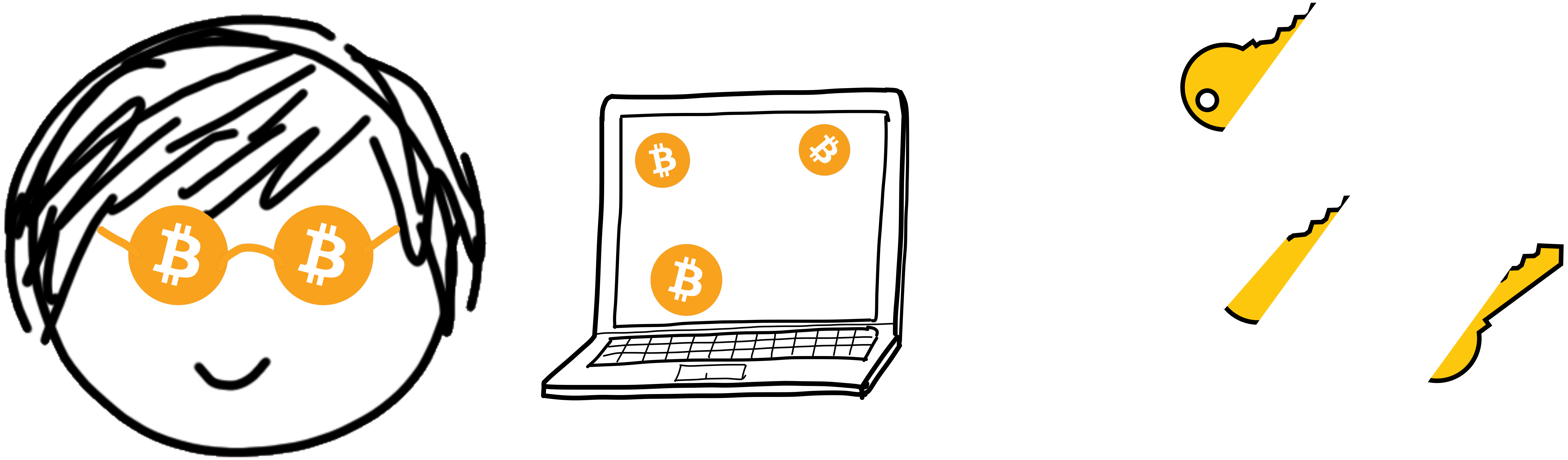


Threshold Signing

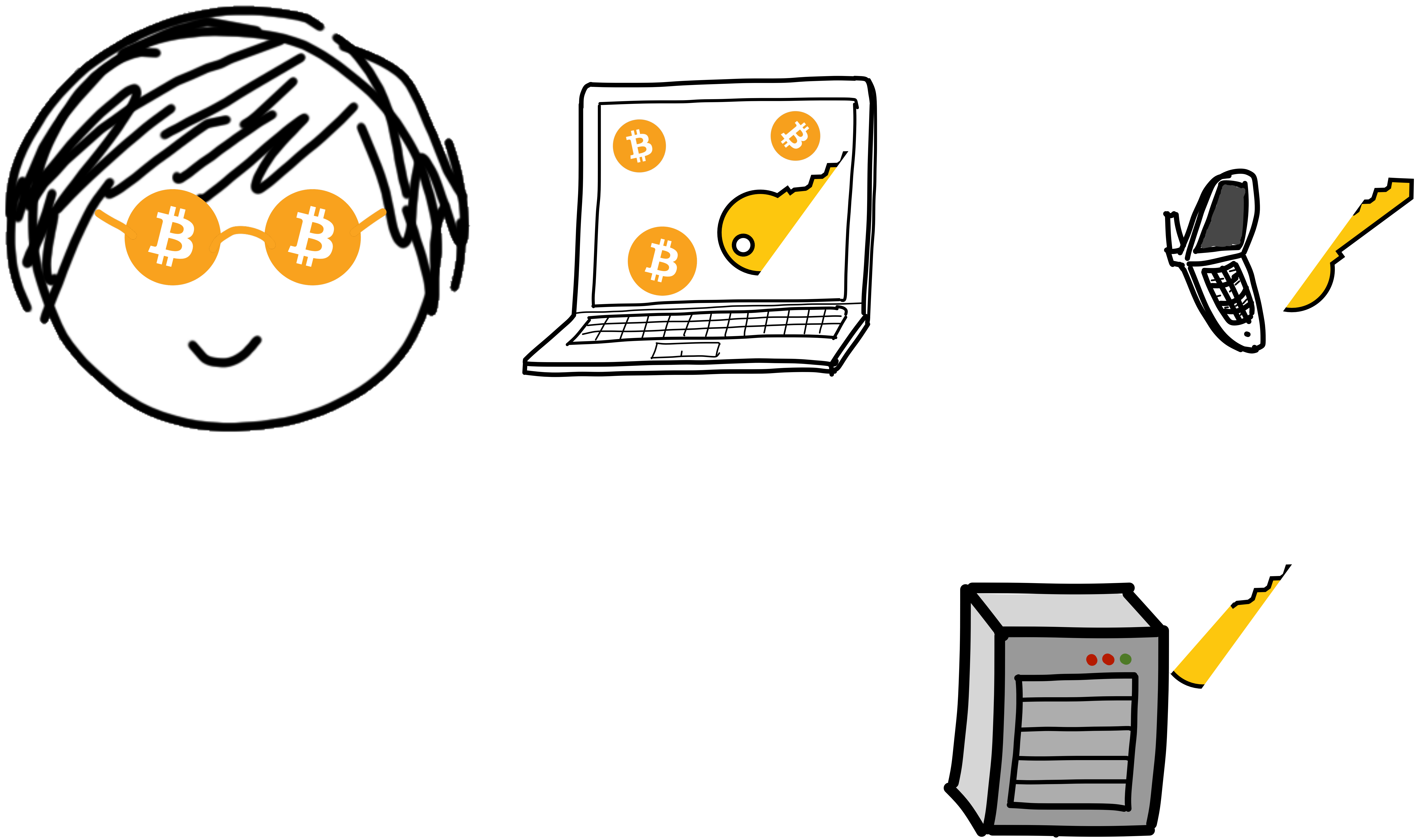


Threshold Signing

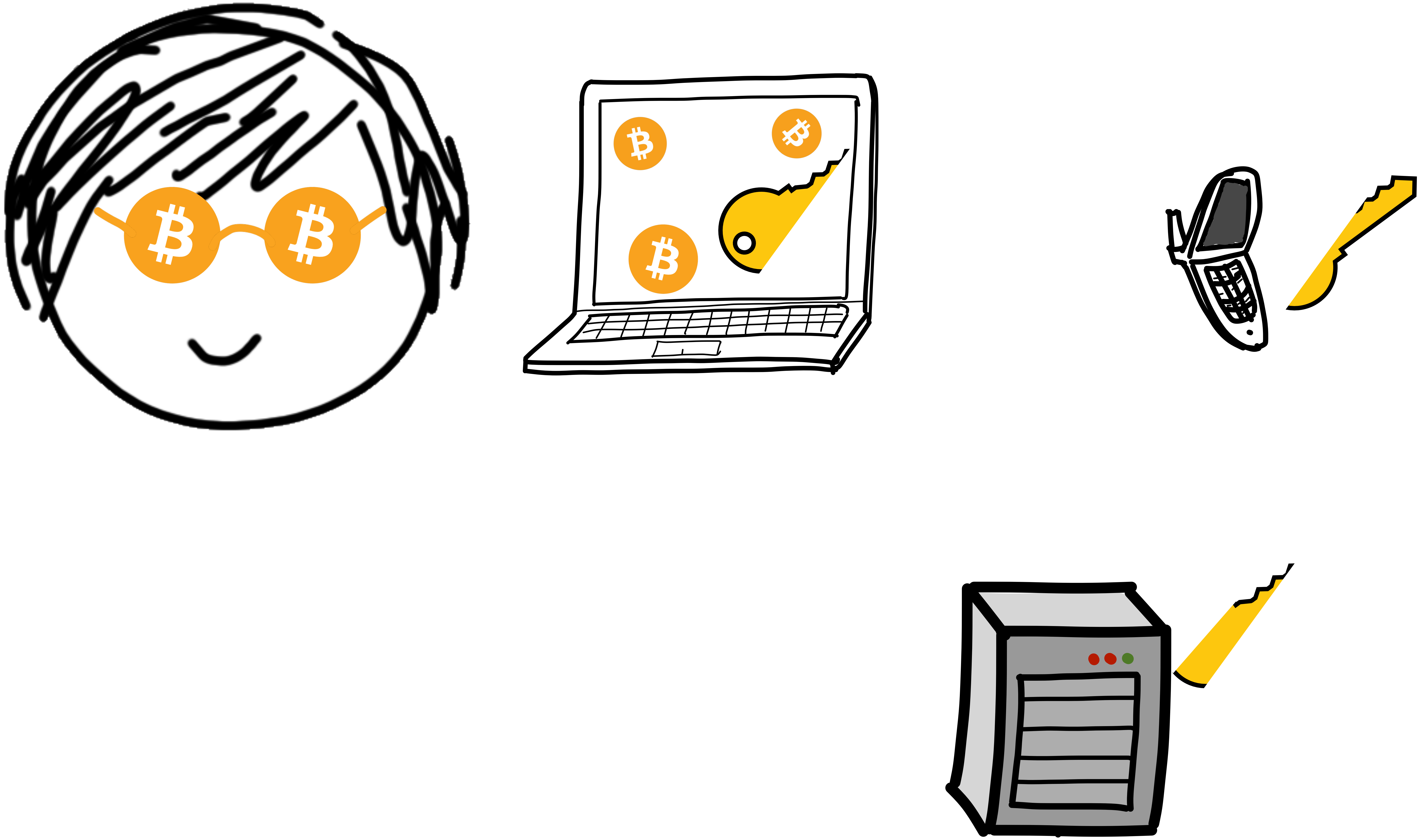
Threshold Signing



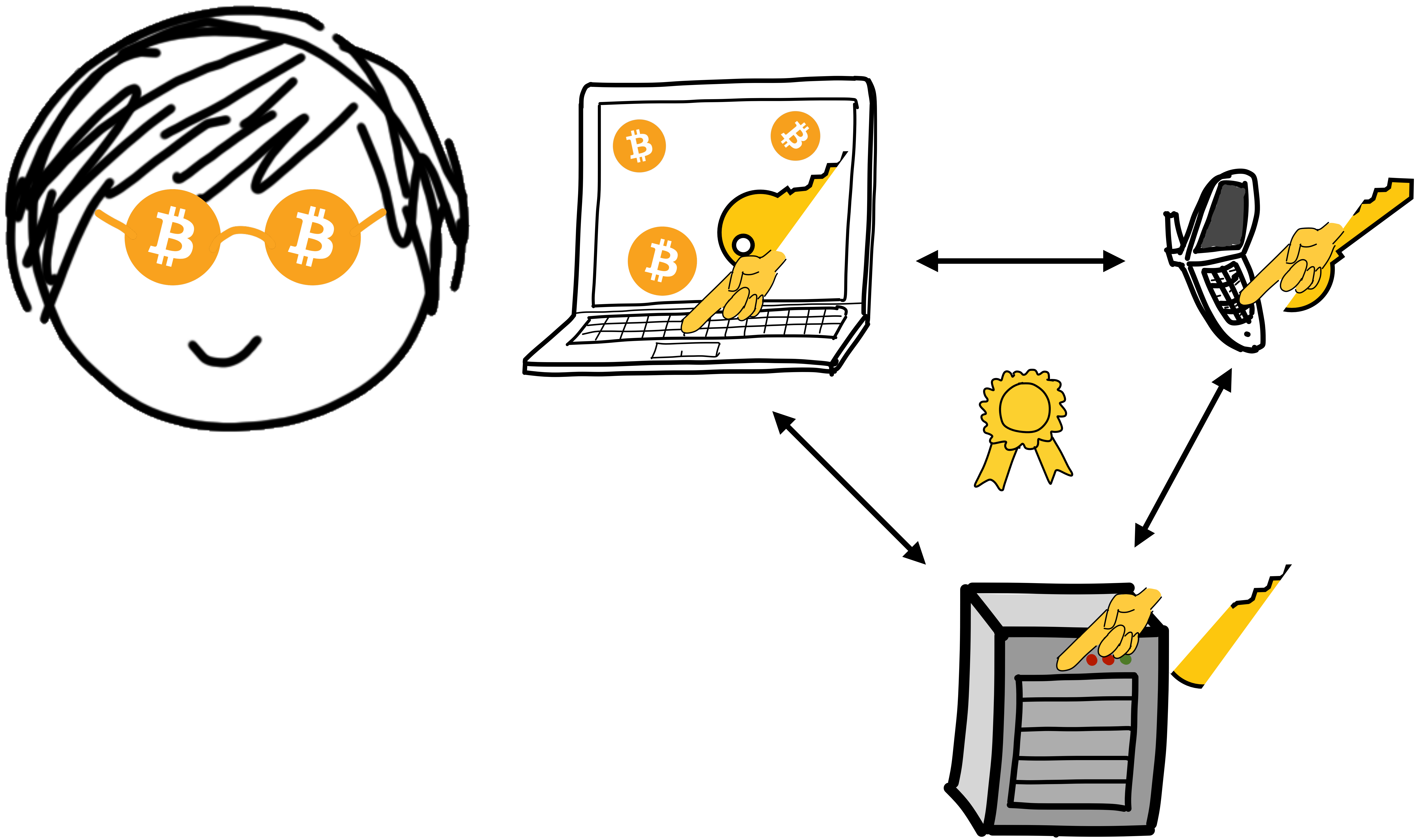
Threshold Signing



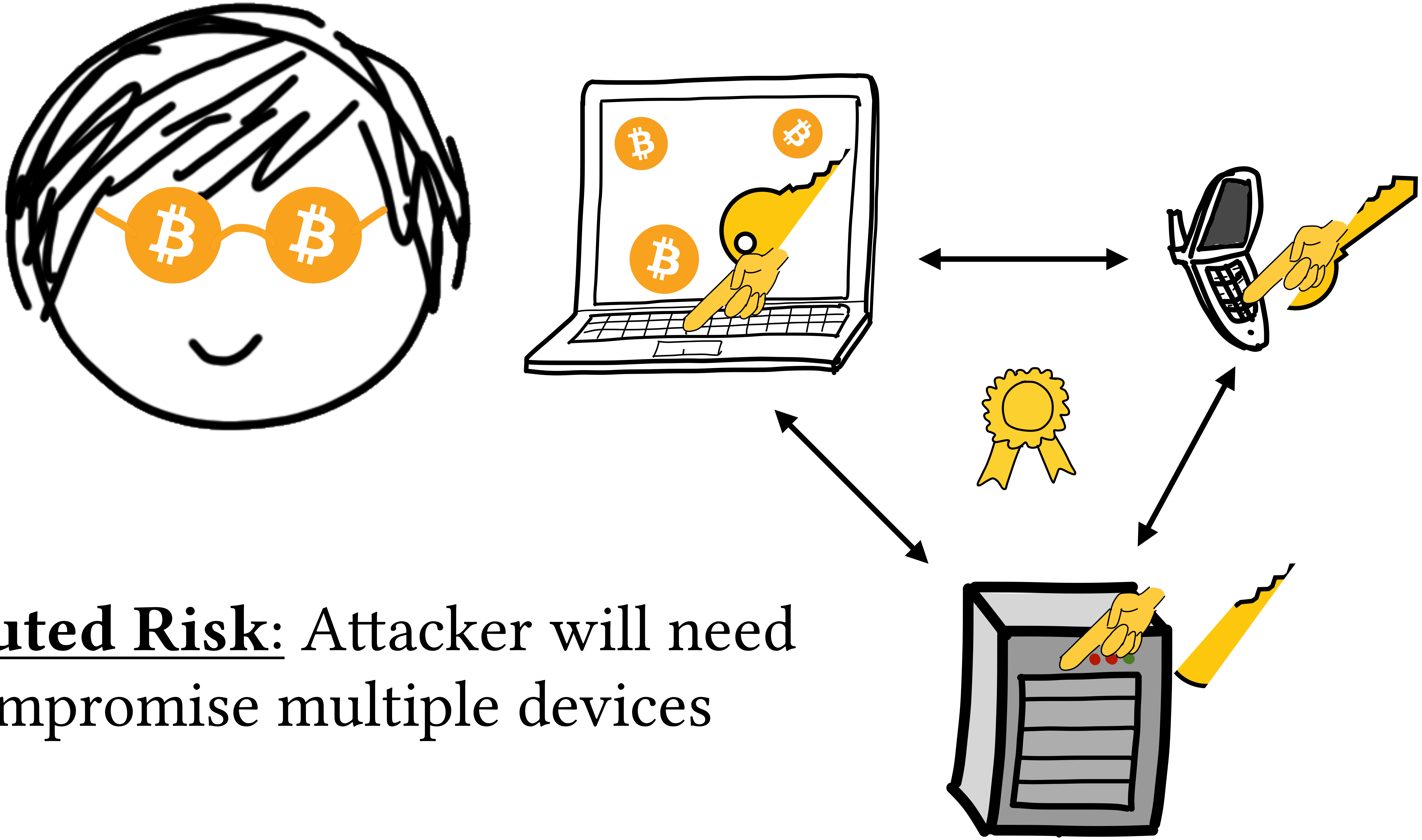
Threshold Signing



Threshold Signing



Threshold Signing



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

Intro

MP-Schnorr is easy

but not ECDSA

How to distribute ECDSA

Evolution of Techniques

ECDSA Tuples

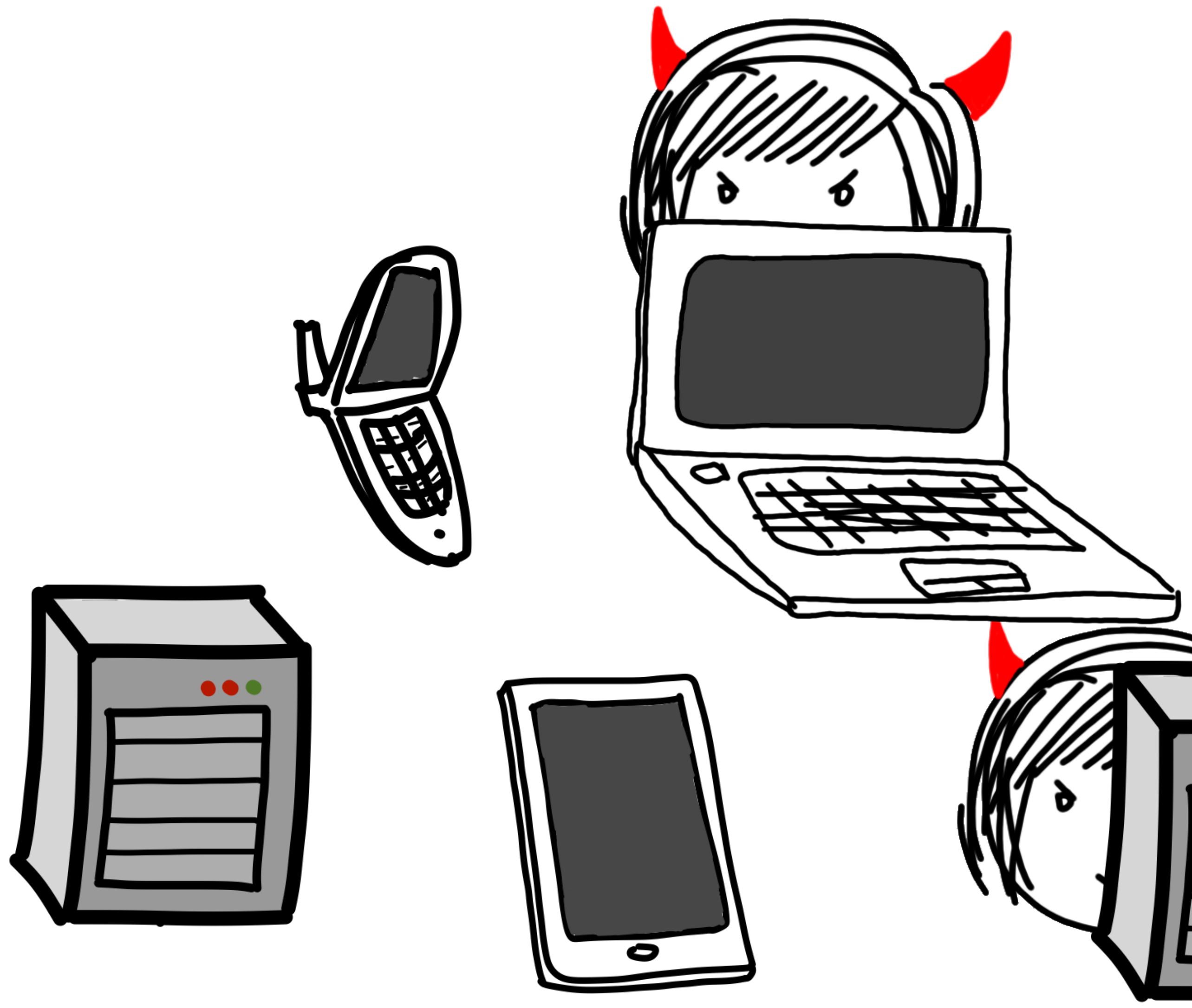
<u>New protocol</u>: Simple consistency check

Tradeoffs

OT vs AHE

Adversary Model

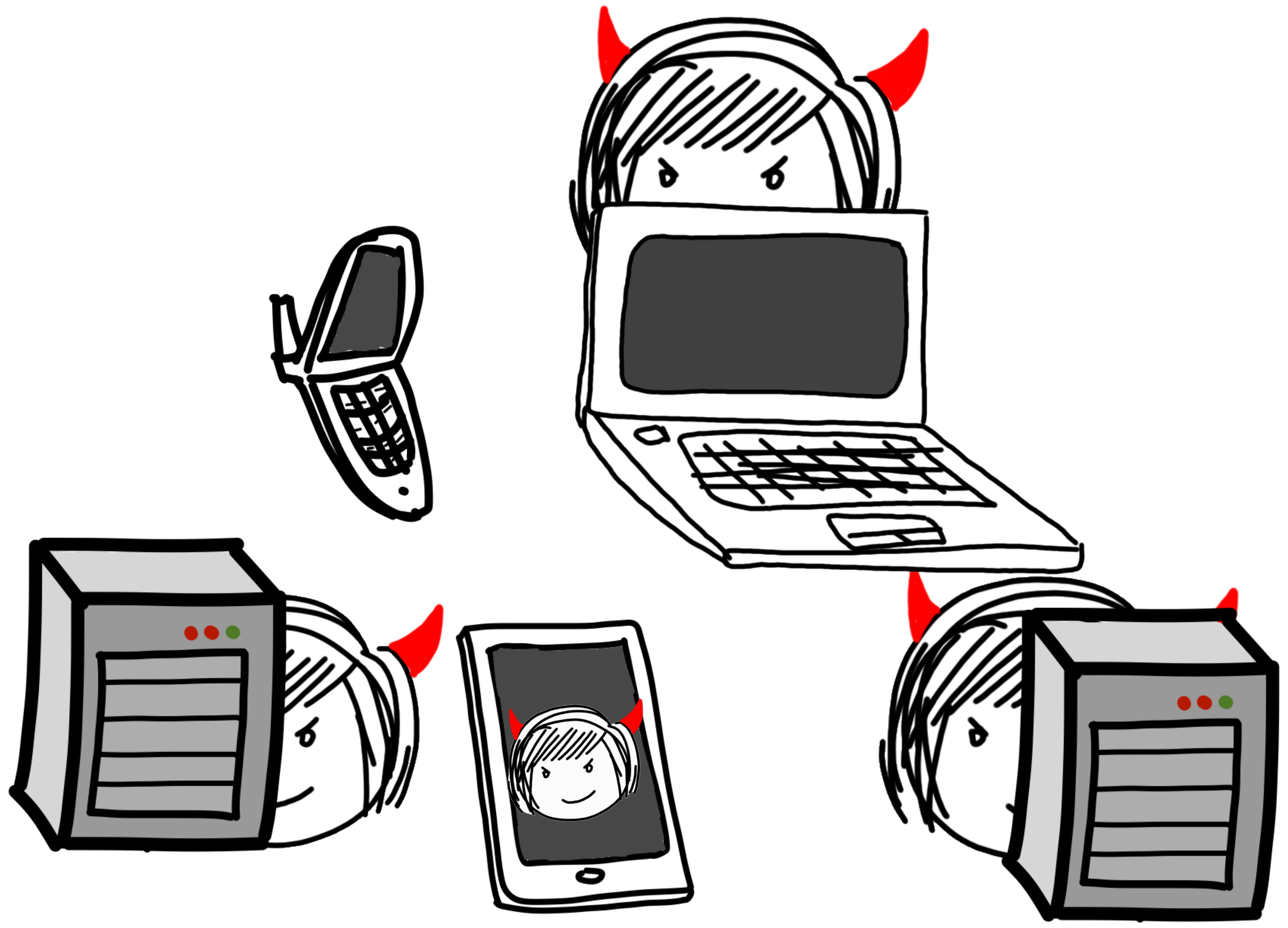
- Corruption threshold



Dishonest majority
(only one device uncompromised)

Adversary Model

- Corruption threshold



Dishonest majority
(only one device uncompromised)

Adversary Model

- Corruption threshold

Dishonest majority
(only one device uncompromised)

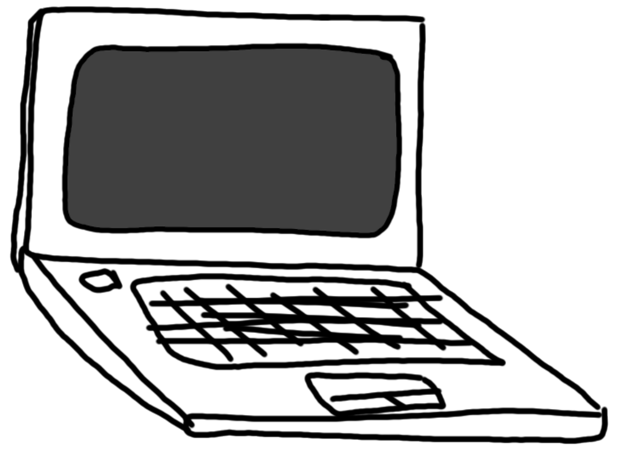
- Adversarial behaviour

Adversary Model

- Corruption threshold

Dishonest majority
(only one device uncompromised)

- Adversarial behaviour

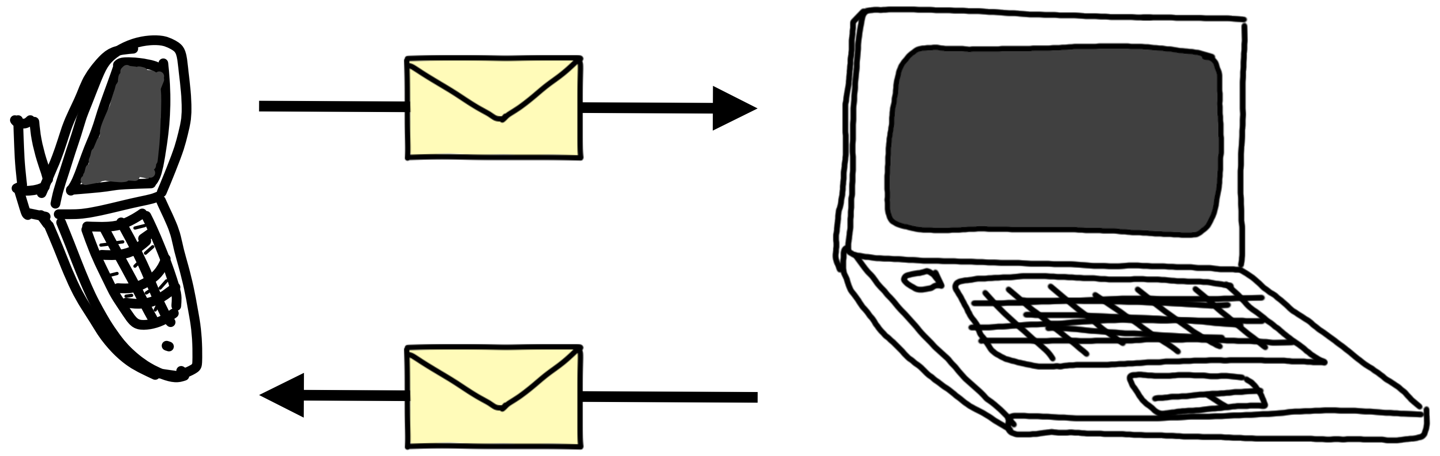


Adversary Model

- Corruption threshold

Dishonest majority
(only one device uncompromised)

- Adversarial behaviour

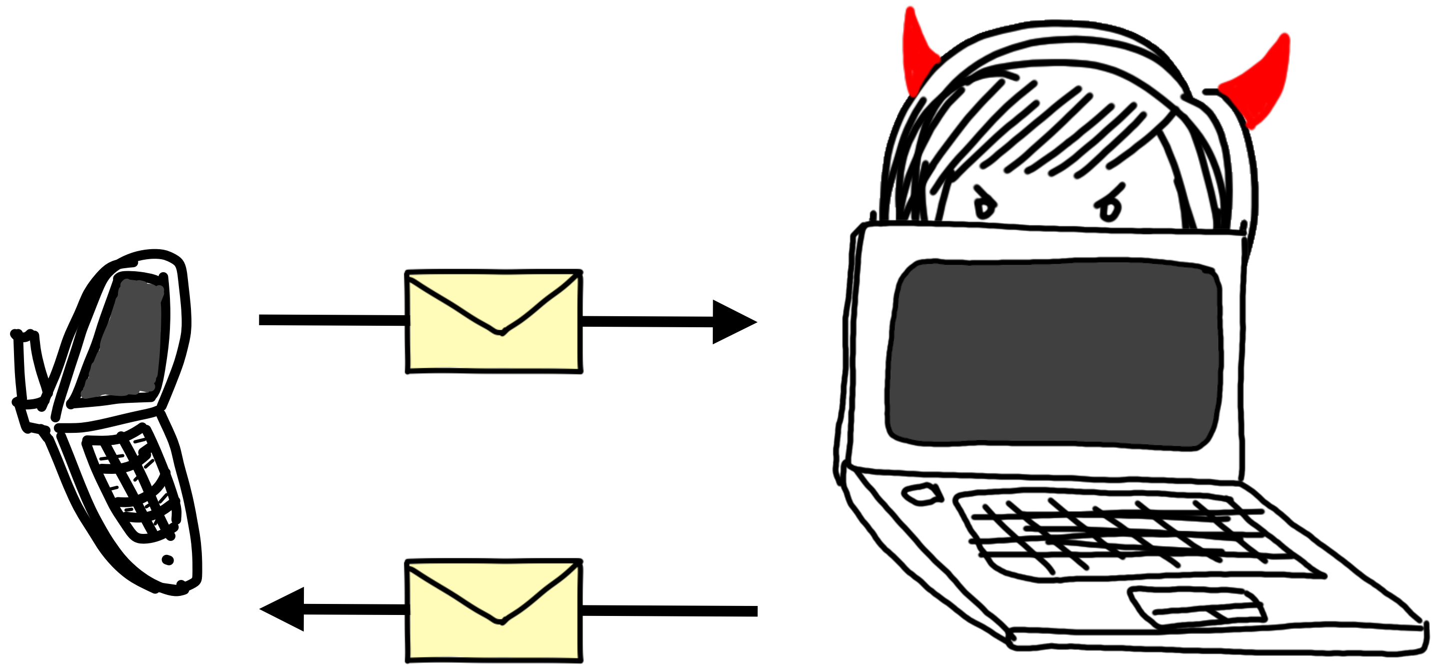


Adversary Model

- Corruption threshold

Dishonest majority
(only one device uncompromised)

- Adversarial behaviour

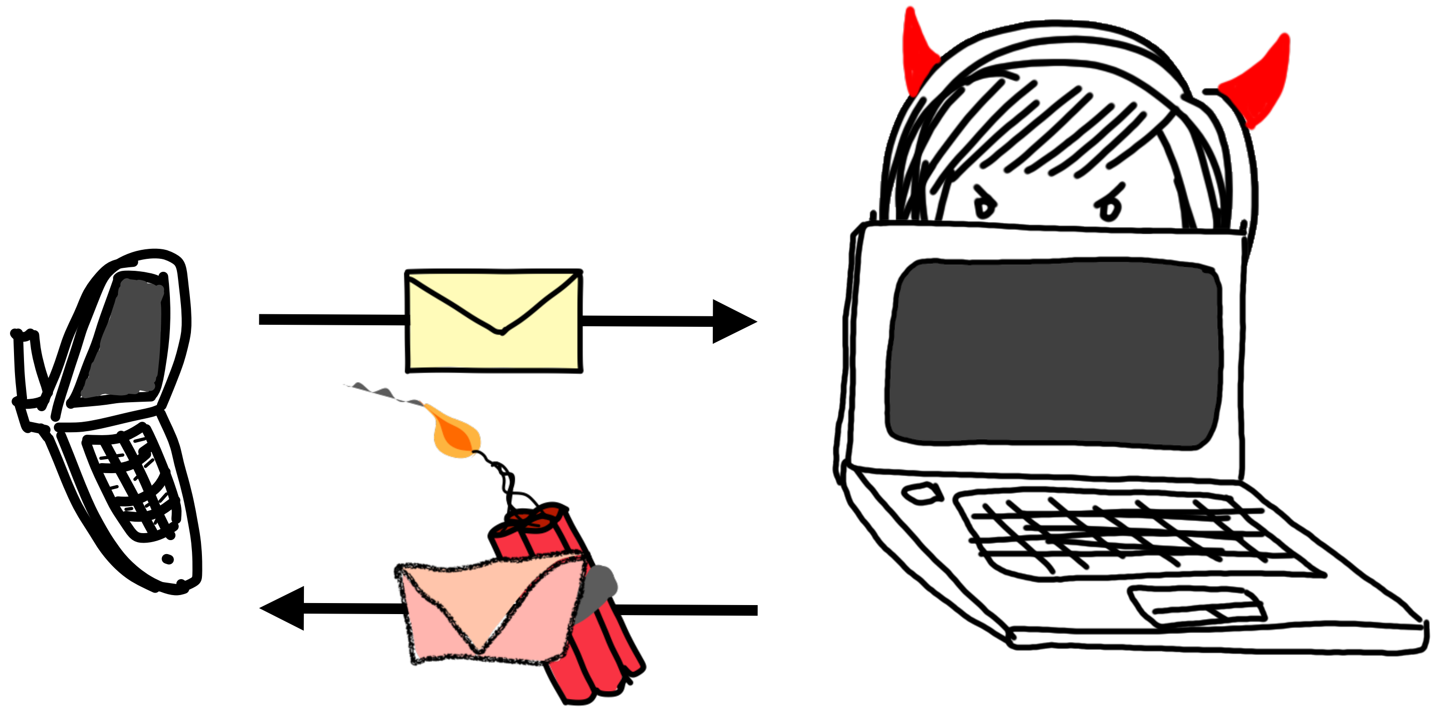


Adversary Model

- Corruption threshold

Dishonest majority
(only one device uncompromised)

- Adversarial behaviour



Malicious
(arbitrary deviations from protocol)

Concrete Example: Schnorr Signatures

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:
  - Hash function - $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:
  - Hash function - $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:
  - Hash function - $H: \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:
  - Hash function - $H : \{0,1\}^* \to \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:
  - Hash function - $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:
  - Hash function - $H : \{0,1\}^* \to \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:
  - Hash function - $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order

$\approx 2^{256}$

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:
  - Hash function - $H : \{0,1\}^* \to \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order

$\approx 2^{256}$

Addition law

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:
  - Hash function - $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order

$\approx 2^{256}$

Addition law

Sixty Seconds on Cyclic Groups

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]

- Tools:

  - Hash function - $H : \{0,1\}^* \to \mathbb{Z}_q$

  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order

$\approx 2^{256}$

Addition law

Sixty Seconds on Cyclic Groups

If $X, Y \in \mathbb{G}$ then $X + Y = Z \in \mathbb{G}$

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]

- Tools:
  - Hash function - $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Generator

(Large prime) order

Addition law

Points on an Elliptic Curve

$\approx 2^{256}$

Sixty Seconds on Cyclic Groups

If $X, Y \in \mathbb{G}$ then $X + Y = Z \in \mathbb{G}$

Any $X \in \mathbb{G}$ can be written as $x \cdot G$

$x \in \mathbb{Z}_q$ is the ***discrete logarithm*** of X

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:
  - Hash function - $H : \{0,1\}^* \to \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order

$\approx 2^{256}$

Addition law

Sixty Seconds on Cyclic Groups

If $X, Y \in \mathbb{G}$ then $X + Y = Z \in \mathbb{G}$

Any $X \in \mathbb{G}$ can be written as $x \cdot G$

$x \in \mathbb{Z}_q$ is the ***discrete logarithm*** of X

$(x + y) \cdot G = x \cdot G + y \cdot G$

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]

- Tools:
  - Hash function - $H : \{0,1\}^* \to \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order

$\approx 2^{256}$

Addition law

Sixty Seconds on Cyclic Groups

If $X, Y \in \mathbb{G}$ then $X + Y = Z \in \mathbb{G}$

Any $X \in \mathbb{G}$ can be written as $x \cdot G$

$x \in \mathbb{Z}_q$ is the ***discrete logarithm*** of X

Integer addition mod q Group addition

$(x \boxed{+} y) \cdot G = x \cdot G \boxed{+} y \cdot G$

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]

- Tools:
  - Hash function - $H : \{0,1\}^* \to \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order

$\approx 2^{256}$

Addition law

Sixty Seconds on Cyclic Groups

If $X, Y \in \mathbb{G}$ then $X + Y = Z \in \mathbb{G}$

Any $X \in \mathbb{G}$ can be written as $x \cdot G$

$x \in \mathbb{Z}_q$ is the ***discrete logarithm*** of X

Integer addition mod q Group addition

$$(x \boxed{+} y) \cdot G = x \cdot G \boxed{+} y \cdot G$$

Discrete Logarithm Problem: Given random $X \in \mathbb{G}$, find its discrete logarithm

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]

- Tools:
  - Hash function - $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order $\approx 2^{256}$

Addition law

Sixty Seconds on Cyclic Groups

If $X, Y \in \mathbb{G}$ then $X + Y = Z \in \mathbb{G}$

Any $X \in \mathbb{G}$ can be written as $x \cdot G$
$x \in \mathbb{Z}_q$ is the ***discrete logarithm*** of X

Integer addition mod q Group addition
$$(x \boxed{+} y) \cdot G = x \cdot G \boxed{+} y \cdot G$$

Discrete Logarithm Problem: Given random $X \in \mathbb{G}$, find its discrete logarithm

For certain elliptic curves, best known algorithms for DLP run in time $\Theta\left(\sqrt{q}\right)$

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]

- Tools:
  - Hash function - $H : \{0,1\}^* \to \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order $\approx 2^{256}$

Addition law

Very informally:

Sixty Seconds on Cyclic Groups

If $X, Y \in \mathbb{G}$ then $X + Y = Z \in \mathbb{G}$

Any $X \in \mathbb{G}$ can be written as $x \cdot G$

$x \in \mathbb{Z}_q$ is the ***discrete logarithm*** of X

Integer addition mod q Group addition

$$(x \boxed{+} y) \cdot G = x \cdot G \boxed{+} y \cdot G$$

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]
- Tools:
  - Hash function - $H : \{0,1\}^* \to \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order

$\approx 2^{256}$

Addition law

Very informally: $x \to X$ ***EASY***

Sixty Seconds on Cyclic Groups

If $X, Y \in \mathbb{G}$ then $X + Y = Z \in \mathbb{G}$

Any $X \in \mathbb{G}$ can be written as $x \cdot G$

$x \in \mathbb{Z}_q$ is the ***discrete logarithm*** of X

Integer addition mod q Group addition

$$(x \boxed{+} y) \cdot G = x \cdot G \boxed{+} y \cdot G$$

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]

- Tools:
  - Hash function - $H : \{0,1\}^* \to \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order

$\approx 2^{256}$

Addition law

Sixty Seconds on Cyclic Groups

If $X, Y \in \mathbb{G}$ then $X + Y = Z \in \mathbb{G}$

Any $X \in \mathbb{G}$ can be written as $x \cdot G$

$x \in \mathbb{Z}_q$ is the ***discrete logarithm*** of X

Integer addition mod q Group addition

$$(x \boxed{+} y) \cdot G = x \cdot G \boxed{+} y \cdot G$$

Very informally: $x \to X$ *EASY*

$X \to x$ HARD

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]

- Tools:
  - Hash function - $H : \{0,1\}^* \to \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order $\approx 2^{256}$

Addition law

Sixty Seconds on Cyclic Groups

If $X, Y \in \mathbb{G}$ then $X + Y = Z \in \mathbb{G}$

Any $X \in \mathbb{G}$ can be written as $x \cdot G$

$x \in \mathbb{Z}_q$ is the ***discrete logarithm*** of X

Integer addition mod q Group addition

$$(x \boxed{+} y) \cdot G = x \cdot G \boxed{+} y \cdot G$$

Very informally: $x \to X$ *EASY* $30\mu s$

$X \to x$ HARD

Concrete Example: Schnorr Signatures

- Elegant signature scheme based on the Discrete Logarithm problem [Schnorr 89]

- Tools:
  - Hash function - $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
  - Group - $(\mathbb{G}, G, q, +)$

Group elements

Points on an Elliptic Curve

Generator

(Large prime) order $\approx 2^{256}$

Addition law

Sixty Seconds on Cyclic Groups

If $X, Y \in \mathbb{G}$ then $X + Y = Z \in \mathbb{G}$

Any $X \in \mathbb{G}$ can be written as $x \cdot G$

$x \in \mathbb{Z}_q$ is the ***discrete logarithm*** of X

Integer addition mod q Group addition

$$(x \boxed{+} y) \cdot G = x \cdot G \boxed{+} y \cdot G$$

Very informally:

$x \rightarrow X$	*EASY*	$30\mu s$
$X \rightarrow x$	HARD	Many billion billions of years

Schnorr Key Generation

$$\mathrm{SchnorrKeyGen}(\mathbb{G}, G, q):$$

$$\mathsf{sk} \leftarrow \mathbb{Z}_q$$

$$\mathsf{PK} = \mathsf{sk} \cdot G$$

$$\text{output } (\mathsf{sk}, \mathsf{PK})$$

secret **k**ey: kept private

Public **K**ey: exposed to the outside world

Schnorr Signing

$\mathsf{SchnorrKeyGen}(\mathbb{G}, G, q):$

$$\mathsf{sk} \leftarrow \mathbb{Z}_q$$

$$\mathsf{PK} = \mathsf{sk} \cdot G$$

$$\text{output } (\mathsf{sk}, \mathsf{PK})$$

$\mathsf{SchnorrSign}(\mathsf{sk}, m):$

Schnorr Signing

$\mathsf{SchnorrKeyGen}(\mathbb{G}, G, q):$

$$\mathsf{sk} \leftarrow \mathbb{Z}_q$$

$$\mathsf{PK} = \mathsf{sk} \cdot G$$

$$\text{output } (\mathsf{sk}, \mathsf{PK})$$

$\mathsf{SchnorrSign}(\mathsf{sk}, m):$

$$k \leftarrow \mathbb{Z}_q$$

Schnorr Signing

$\mathrm{SchnorrKeyGen}(\mathbb{G}, G, q):$

$\mathsf{sk} \leftarrow \mathbb{Z}_q$

$\mathsf{PK} = \mathsf{sk} \cdot G$

output $(\mathsf{sk}, \mathsf{PK})$

$\mathrm{SchnorrSign}(\mathsf{sk}, m):$

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

Schnorr Signing

$\mathrm{SchnorrKeyGen}(\mathbb{G}, G, q):$

$$\mathsf{sk} \leftarrow \mathbb{Z}_q$$

$$\mathsf{PK} = \mathsf{sk} \cdot G$$

$$\text{output } (\mathsf{sk}, \mathsf{PK})$$

$\mathrm{SchnorrSign}(\mathsf{sk}, m):$

$$k \leftarrow \mathbb{Z}_q$$

$$\boxed{R} = k \cdot G$$

NONCE
One-time use
value

Schnorr Signing

$\mathrm{SchnorrKeyGen}(\mathbb{G}, G, q):$

$$\mathsf{sk} \leftarrow \mathbb{Z}_q$$

$$\mathsf{PK} = \mathsf{sk} \cdot G$$

$$\text{output } (\mathsf{sk}, \mathsf{PK})$$

$\mathrm{SchnorrSign}(\mathsf{sk}, m):$

$$k \leftarrow \mathbb{Z}_q$$

$$\boxed{R} = k \cdot G$$

NONCE
One-time use
value

$$e = H(R \| m)$$

Schnorr Signing

$\text{SchnorrKeyGen}(\mathbb{G}, G, q):$

$$\mathsf{sk} \leftarrow \mathbb{Z}_q$$

$$\mathsf{PK} = \mathsf{sk} \cdot G$$

$$\text{output } (\mathsf{sk}, \mathsf{PK})$$

$\text{SchnorrSign}(\mathsf{sk}, m):$

$$k \leftarrow \mathbb{Z}_q$$

$$\boxed{R} = k \cdot G$$

NONCE
One-time use
value

$$e = H(R \| m)$$

$$s = k - \mathsf{sk} \cdot e \pmod q$$

Schnorr Signing

$$\text{SchnorrKeyGen}(\mathbb{G}, G, q):$$

$$\mathsf{sk} \leftarrow \mathbb{Z}_q$$

$$\mathsf{PK} = \mathsf{sk} \cdot G$$

$$\text{output } (\mathsf{sk}, \mathsf{PK})$$

$$\text{SchnorrSign}(\mathsf{sk}, m):$$

$$k \leftarrow \mathbb{Z}_q$$

$$\boxed{R} = k \cdot G$$

NONCE
One-time use value

$$e = H(R \| m)$$

$$s = k - \mathsf{sk} \cdot e \pmod{q}$$

$$\sigma = (s, R)$$

$$\text{output } \sigma$$

Schnorr Signing

$\mathrm{SchnorrKeyGen}(\mathbb{G}, G, q)$:

$$\mathsf{sk} \leftarrow \mathbb{Z}_q$$

$$\mathsf{PK} = \mathsf{sk} \cdot G$$

$$\text{output } (\mathsf{sk}, \mathsf{PK})$$

$\mathrm{SchnorrSign}(\mathsf{sk}, m)$:

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

NONCE
One-time use
value

$$e = H(R \| m)$$

$$s = k - \mathsf{sk} \cdot e \pmod{q}$$

$$\sigma = (s, R)$$

$$\text{output } \sigma$$

Verifying a signature: $s \cdot G \stackrel{?}{=} R - e \cdot \mathsf{PK}$

Schnorr Signing

$\text{SchnorrKeyGen}(\mathbb{G}, G, q):$

$$\mathsf{sk} \leftarrow \mathbb{Z}_q$$

$$\mathsf{PK} = \mathsf{sk} \cdot G$$

$$\text{output } (\mathsf{sk}, \mathsf{PK})$$

$\text{SchnorrSign}(\mathsf{sk}, m):$

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

NONCE
One-time use value

$$e = H(R \| m)$$

$$s = k - \mathsf{sk} \cdot e \pmod{q}$$

$$\sigma = (s, R)$$

$$\text{output } \sigma$$

Verifying a signature: $s \cdot G \stackrel{?}{=} \boxed{R} - e \cdot \boxed{\mathsf{PK}}$

$\boxed{k \cdot G}$ $\boxed{\mathsf{sk} \cdot G}$

Schnorr Signing

$\mathrm{SchnorrKeyGen}(\mathbb{G}, G, q)$:

$$\mathsf{sk} \leftarrow \mathbb{Z}_q$$

$$\mathsf{PK} = \mathsf{sk} \cdot G$$

$$\text{output } (\mathsf{sk}, \mathsf{PK})$$

$\mathrm{SchnorrSign}(\mathsf{sk}, m)$:

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

NONCE
One-time use value

$$e = H(R \| m)$$

$$s = k - \mathsf{sk} \cdot e \pmod q$$

$$\sigma = (s, R)$$

$$\text{output } \sigma$$

Verifying a signature: $s \cdot G \stackrel{?}{=} \underbrace{\boxed{R}}_{k \cdot G} - e \cdot \underbrace{\boxed{\mathsf{PK}}}_{\mathsf{sk} \cdot G}$

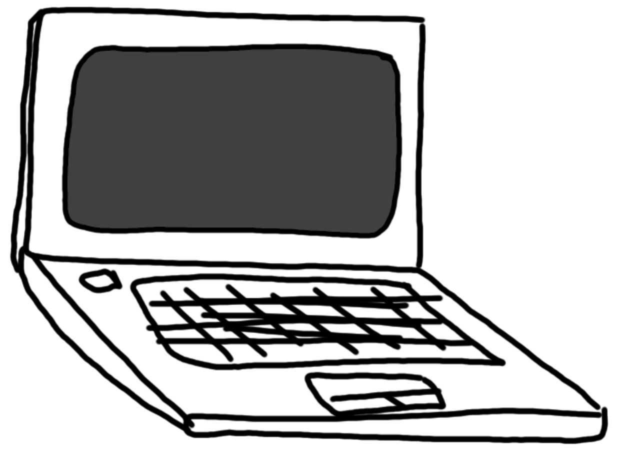
$$s \cdot G \stackrel{?}{=} (k - e \cdot \mathsf{sk}) \cdot G$$

Secret Sharing

- $[x]$ denotes that a value $x \in \mathbb{Z}_q$ is "secret-shared" across devices

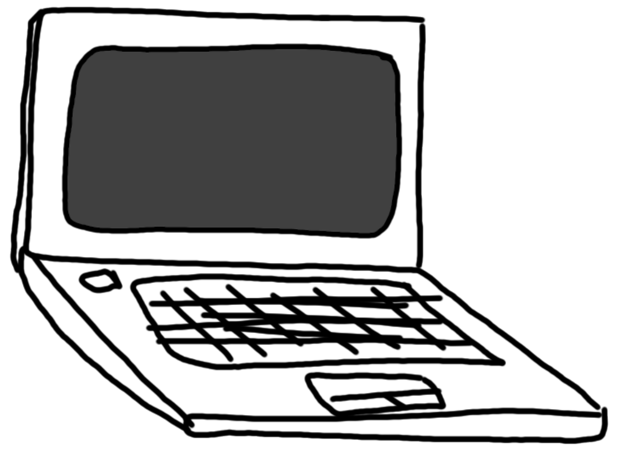
- We will only use "linear" secret sharing schemes
  $a[x] + b[y] = [ax + by]$

Additive Secret Sharing



Additive Secret Sharing

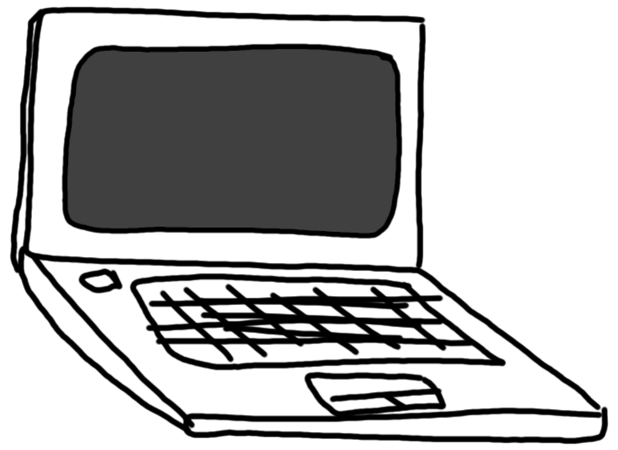
$x \in \mathbb{Z}_q$



Additive Secret Sharing

$$x \in \mathbb{Z}_q$$

$$x_A + x_B = x$$

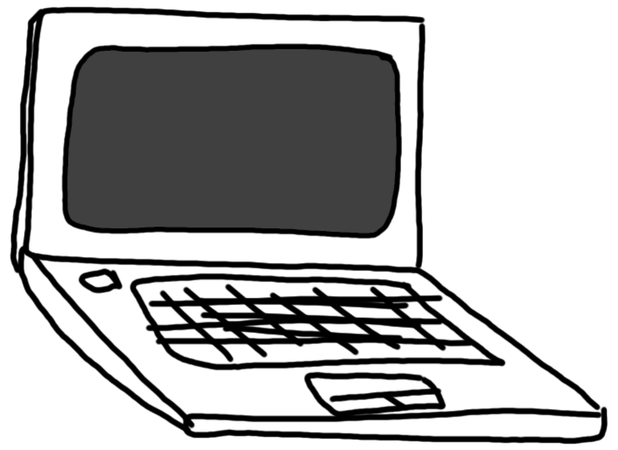


Additive Secret Sharing

$$x \in \mathbb{Z}_q$$

$$x_A + x_B = x$$

x_A



x_B

Additive Secret Sharing

$$x \in \mathbb{Z}_q$$

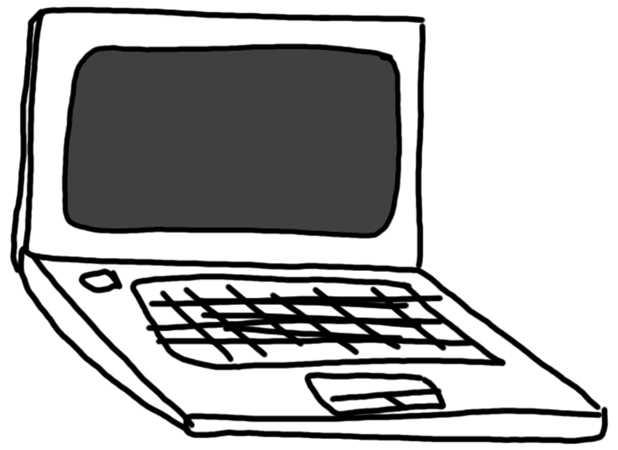
$$x_A + x_B = x$$

x_A $[x]$ x_B

Additive Secret Sharing

$$x \in \mathbb{Z}_q$$

$$x_A + x_B = x$$

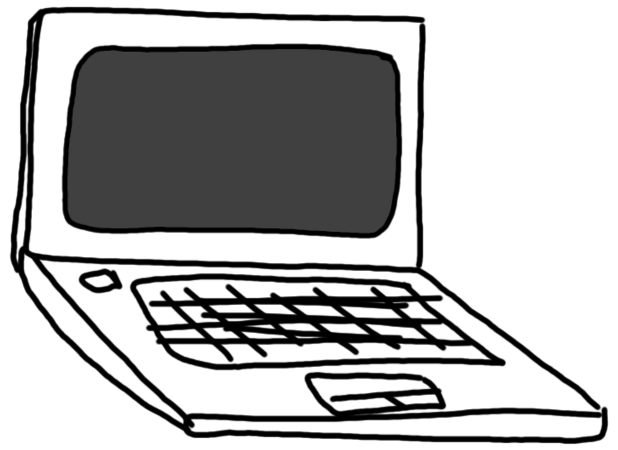


x_A $[x]$ x_B

Additive Secret Sharing

$$x \in \mathbb{Z}_q$$

$$x_A + x_B = x$$

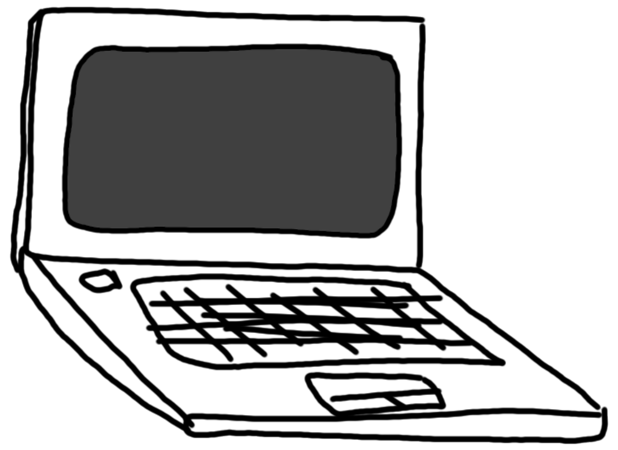


x_A $[x]$ x_B

Additive Secret Sharing

$$\boxed{?} \in \mathbb{Z}_q$$

$$x_A + \boxed{?} = \boxed{?}$$



x_A

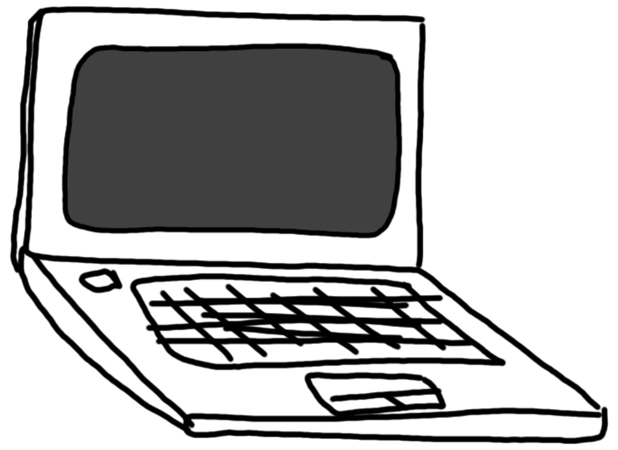
$[\boxed{?}]$

$\boxed{?}$

Additive Secret Sharing

$$x \in \mathbb{Z}_q$$

$$x_A + x_B = x$$



x_A

$$[x]$$

x_B

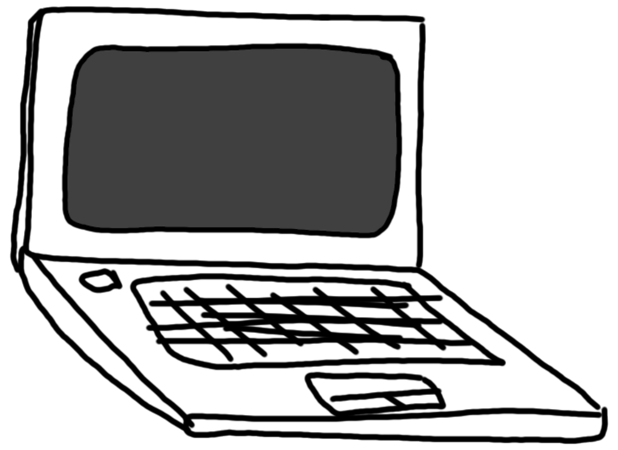
$$[y]$$

$$y_A + y_B = y$$

Additive Secret Sharing

$$x \in \mathbb{Z}_q$$

$$x_A + x_B = x$$

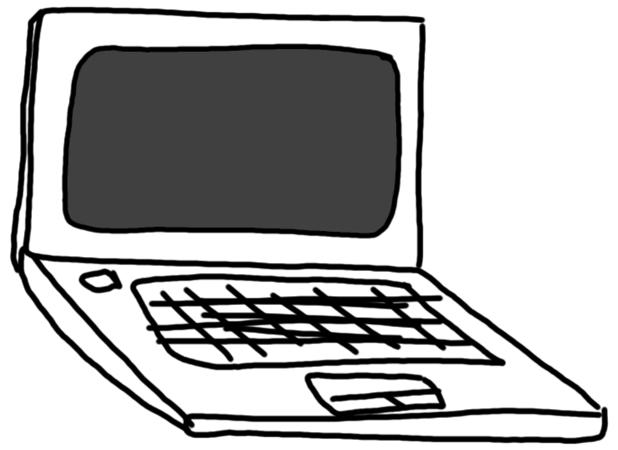


x_A	$[x]$	x_B
y_A	$[y]$	y_B

Additive Secret Sharing

$$x \in \mathbb{Z}_q$$

$$x_A + x_B = x$$



x_A $\qquad$ $[x]$ $\qquad$ x_B

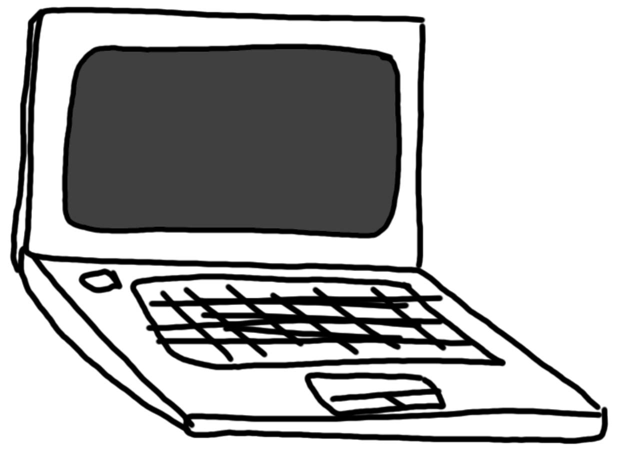
y_A $\qquad$ $[y]$ $\qquad$ y_B

$$[z = cx + y]$$

Additive Secret Sharing

$$x \in \mathbb{Z}_q$$

$$x_A + x_B = x$$

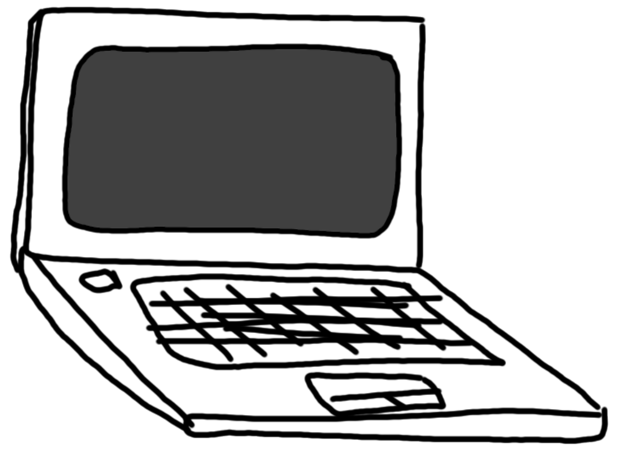


x_A	$[x]$	x_B
y_A	$[y]$	y_B
$z_A = cx_A + y_A$	$[z = cx + y]$	$z_B = cx_B + y_B$

Distributing Schnorr w. Additive Secret Sharing

$$x \in \mathbb{Z}_q$$

$$x_A + x_B = x$$

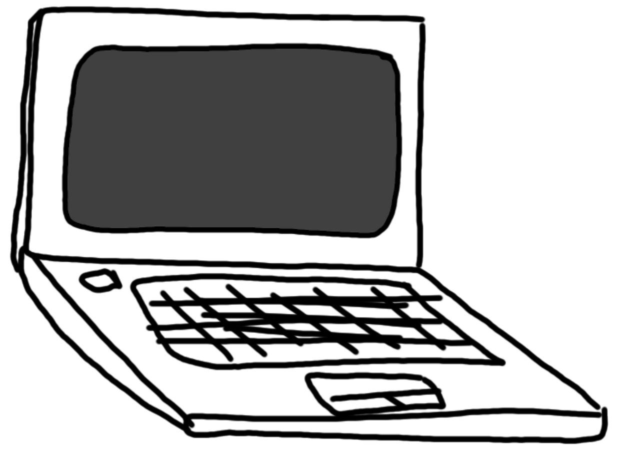


x_A	$[x]$	x_B
y_A	$[y]$	y_B
$z_A = cx_A + y_A$	$[z = cx + y]$	$z_B = cx_B + y_B$

Distributing Schnorr w. Additive Secret Sharing

$$\mathsf{sk} \in \mathbb{Z}_q$$

$$\mathsf{sk}_A + \mathsf{sk}_B = \mathsf{sk}$$

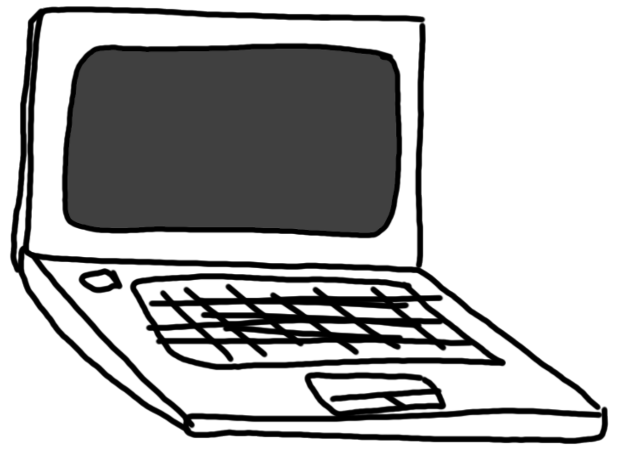


sk_A	$[\mathsf{sk}]$	sk_B
k_A	$[k]$	k_B
$s_A = e\mathsf{sk}_A + k_A$	$[s = e\mathsf{sk} + k]$	$s_B = e\mathsf{sk}_B + k_B$

Distributing Schnorr w. Additive Secret Sharing

$$\mathsf{sk} \in \mathbb{Z}_q$$

$$\mathsf{sk}_A + \mathsf{sk}_B = \mathsf{sk}$$



$$\mathsf{sk}_A \qquad [\mathsf{sk}] \qquad \mathsf{sk}_B$$

$$k_A \qquad [k] \qquad k_B$$

$$R = k \cdot G$$

$$e = H(m, R)$$

$$s_A = e\mathsf{sk}_A + k_A \qquad [s = e\mathsf{sk} + k] \qquad s_B = c\mathsf{sk}_B + k_B$$

3 Round Schnorr Signing

Folklore, [Lindell 22]

Input : $\mathsf{pk} = [\mathsf{sk}] \cdot G$, $[\mathsf{sk}]$, $[k]$

Establish $R = [k] \cdot G$

Round 1	Exchange Commit $\left(R_i = [k]_i \cdot G\right)$
Round 2	Release R_i, set $R = \Sigma_i \, R_i$
Round 3	Reveal $s = [\mathsf{sk}] \cdot H(m, R) + [k]$

Output (R, s)

(Threshold) Schnorr in Practice?

- Schnorr signatures are **old** (well-studied), **compact**, **fast**, and **easy to distribute with MPC (i.e. thresholdize)**
- However it was patented—major barrier for internet adoption
- Patent expired recently; adoption is increasing but much of the **internet infrastructure does not support Schnorr**

ECDSA

- **E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm
- Devised by Scott Vanstone in 1992, standardized by NIST
- Differs from Schnorr enough so that **patent doesn't apply**
- **Widespread adoption** across the internet

… but **MPC-unfriendly**

Threshold ECDSA: Challenges

$$\begin{aligned} &\mathsf{SchnorrSign}(\mathsf{sk}, m): \\ &\quad k \leftarrow \mathbb{Z}_q \\ &\quad R = k \cdot G \\ &\quad e = H(R \| m) \\ &\quad s = k - \mathsf{sk} \cdot e \\ &\quad \sigma = (s, R) \\ &\quad \text{output } \sigma \end{aligned}$$

$$\begin{aligned} &\mathsf{ECDSASign}(\mathsf{sk}, m): \\ &\quad k \leftarrow \mathbb{Z}_q \\ &\quad R = k \cdot G \\ &\quad e = H(m) \end{aligned}$$

Threshold ECDSA: Challenges

$\mathsf{SchnorrSign}(\mathsf{sk}, m):$

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

$e = H(R \| m)$

$s = k - \mathsf{sk} \cdot e$

$\sigma = (s, R)$

output σ

$\mathsf{ECDSASign}(\mathsf{sk}, m):$

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

$e = H(m)$

Standard 2 round sampling

Threshold ECDSA: Challenges

SchnorrSign(sk, m) :

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

$e = H(R \| m)$

$s = k - \mathsf{sk} \cdot e$

$\sigma = (s, R)$

output σ

ECDSASign(sk, m) :

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

Standard 2 round sampling

$e = H(m)$

$$s = \frac{e + \mathsf{sk} \cdot r_x}{k}$$

output $\sigma = (s, R)$

Threshold ECDSA: Challenges

$$\mathrm{ECDSASign}(\mathsf{sk}, m):$$

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

$$e = H(m)$$

$$\boxed{s = \frac{e + \mathsf{sk} \cdot r_x}{k}}$$

$$\text{output } \sigma = (s, R)$$

Threshold ECDSA: Challenges

$$\mathrm{ECDSASign}(\mathsf{sk}, m):$$

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

$$e = H(m)$$

$$s = \frac{e + \boxed{\mathsf{sk}} \cdot r_x}{\boxed{k}}$$

$$\text{output } \sigma = (s, R)$$

Multiplication of secret values

Threshold ECDSA: Challenges

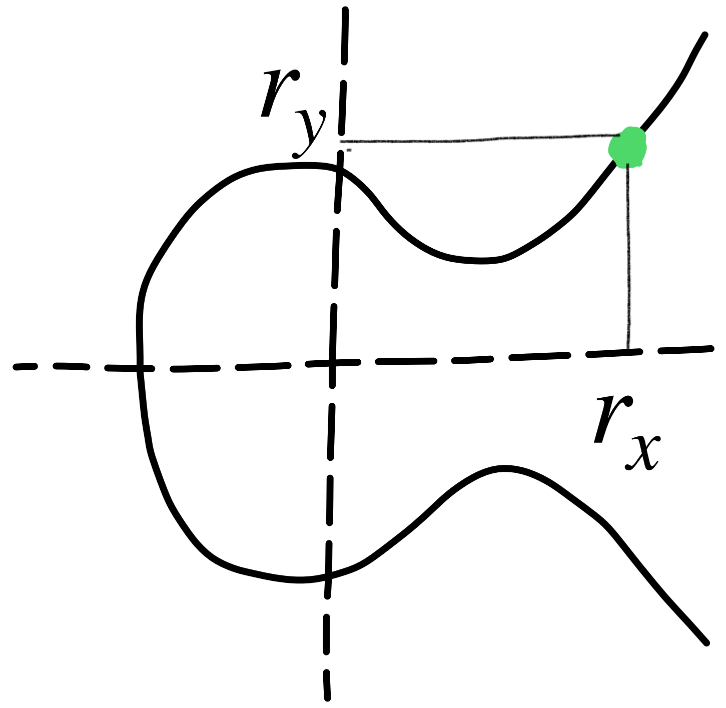
$$\begin{aligned}
&\mathrm{ECDSASign}(\mathsf{sk}, m):\\
&k \leftarrow \mathbb{Z}_q\\
&R = k \cdot G\\
&e = H(m)\\
&s = \frac{e + \mathsf{sk} \cdot r_x}{\boxed{k}}\\
&\text{output } \sigma = (s, R)
\end{aligned}$$

Multiplication of secret values

Division (Modular inverse)

Threshold ECDSA: Challenges

$R = (r_x, r_y)$



$$\mathsf{ECDSASign}(\mathsf{sk}, m):$$

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

$$e = H(m)$$

$$s = \frac{e + \mathsf{sk} \cdot r_x}{k}$$

$$\text{output } \sigma = (s, R)$$

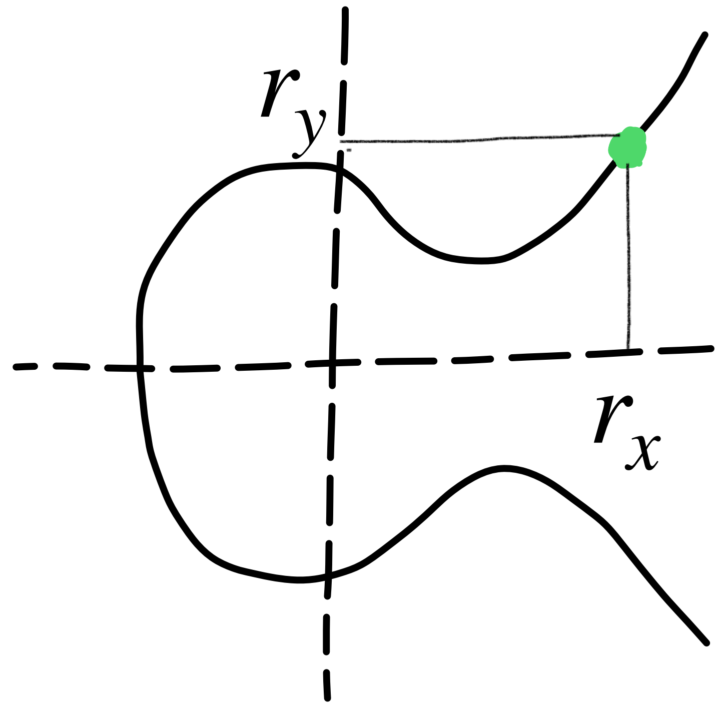
Multiplication of secret values

x-coordinate of ***R*** (not secret)

Division (Modular inverse)

Threshold ECDSA: Challenges

$R = (r_x, r_y)$



$$\mathsf{ECDSASign}(\mathsf{sk}, m):$$

$$k \leftarrow \mathbb{Z}_q$$

$$R = k \cdot G$$

$$e = H(m)$$

$$s = \frac{e + \mathsf{sk} \cdot r_x}{\boxed{k}}$$

$$\text{output } \sigma = (s, R)$$

There is no one-size-fits-all solution

Multiplication of secret values

x-coordinate of $\boldsymbol{R}$ (not secret)

Division (Modular inverse)

Secure Two-Party Multiplication

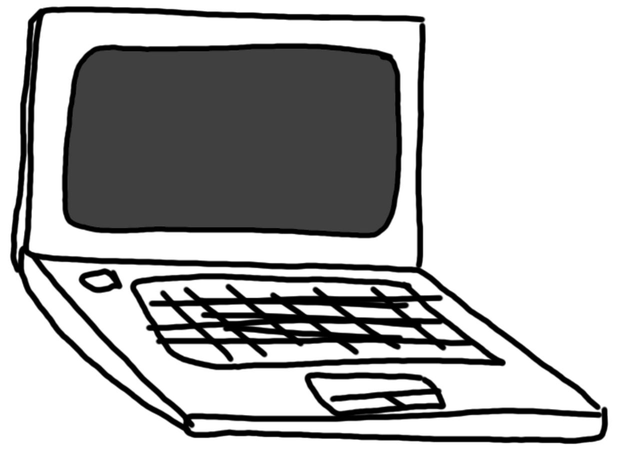
a.k.a. OLE, Mult2Add

Underlies many dishonest majority MPC protocols

α

2P-MUL

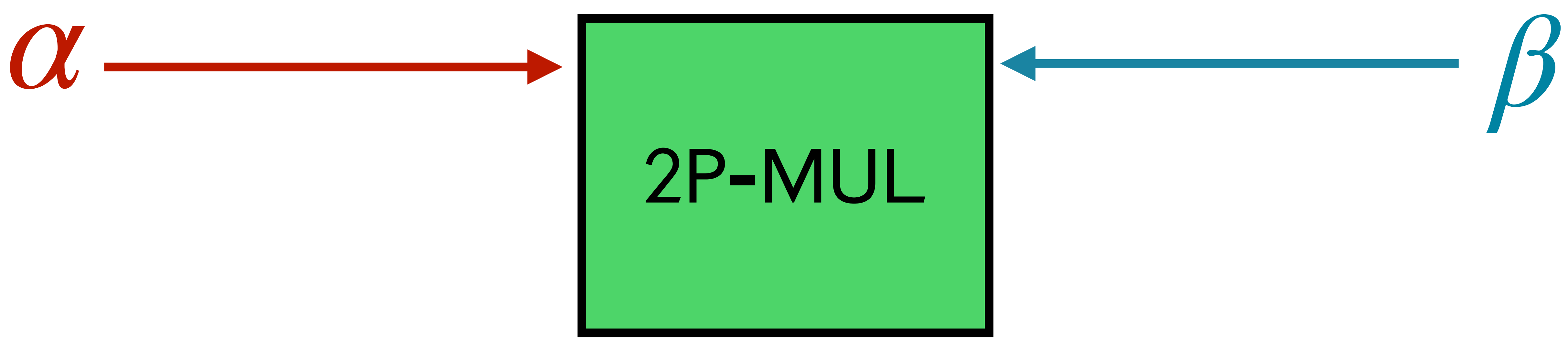
β

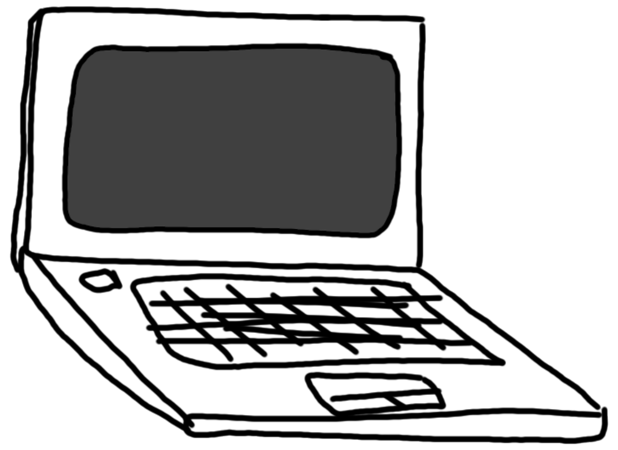


Secure Two-Party Multiplication

a.k.a. OLE, Mult2Add

Underlies many dishonest majority MPC protocols

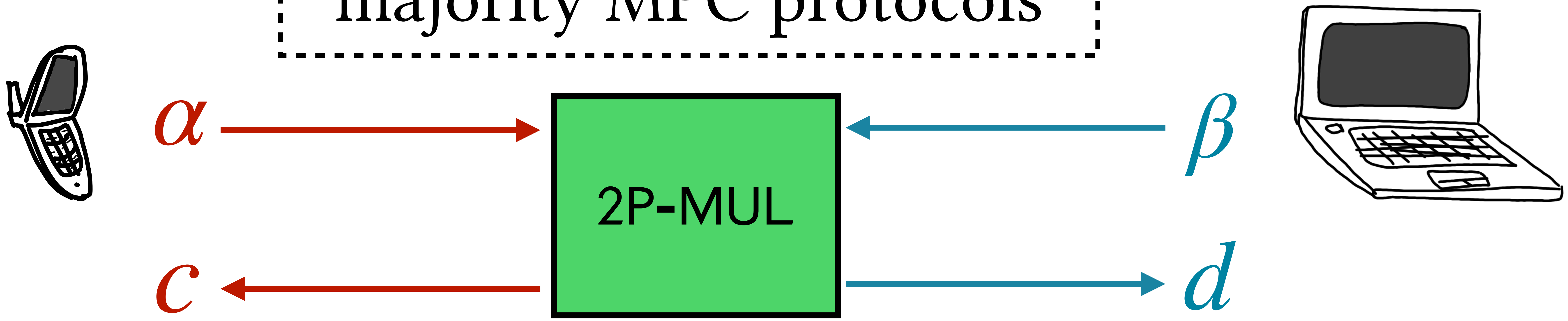




Secure Two-Party Multiplication

a.k.a. OLE, Mult2Add

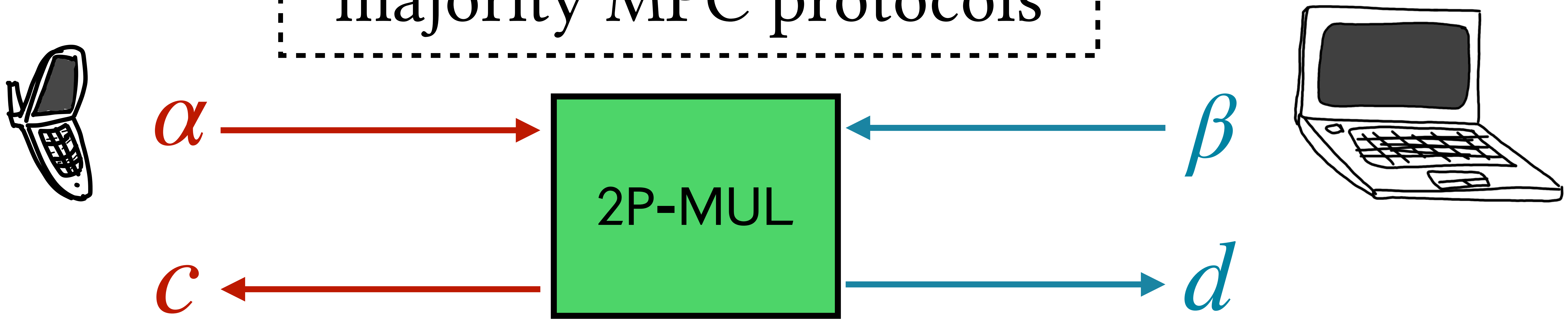
Underlies many dishonest majority MPC protocols



Secure Two-Party Multiplication

a.k.a. OLE, Mult2Add

Underlies many dishonest majority MPC protocols

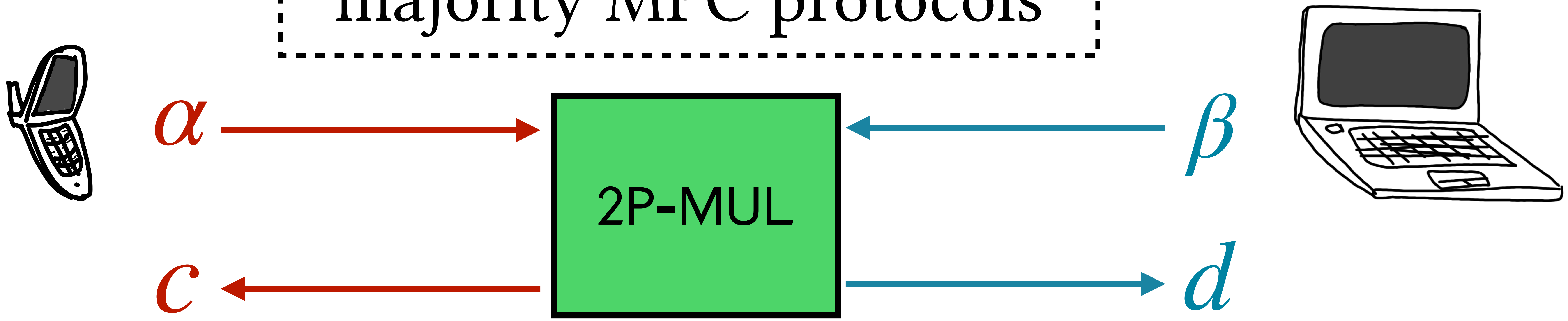


$$c + d = \alpha \cdot \beta$$

Secure Two-Party Multiplication

a.k.a. OLE, Mult2Add

Underlies many dishonest majority MPC protocols



$$c + d = \alpha \cdot \beta$$

Tool to split a product of secret inputs $\alpha\beta$ into additive secrets c, d

Instantiable efficiently from: OT, Paillier, Class Groups

Threshold ECDSA: State of the Art

- Rough costs with 256-bit curve, for each additional party (computation aggregated across [Gavenda 21, XAXYC 21, BMP 22]):

Protocol	2P-MUL	Rounds	Bandwidth (KB)	Computation (ms)
[DKLs 19]	OT	$\log(t) + 6$	90	<10
[HLNR 18/23]	OT+	5	40	50—100
[CGGMP 20]	Paillier	4	15	Hundreds
[GG 18]	Paillier	8	7	Hundreds
[CCLST20, YCX21]	Class Groups	4	4	> 1000

Threshold ECDSA: Goal

- Rough costs with 256-bit curve, for each additional party (computation aggregated across [Gavenda 21, XAXYC 21, BMP 22]):

	2P-MUL	Bandwidth (KB)	Computation (ms)
Simple, unified protocol	OT	90	<10
	Paillier	15	Hundreds
	Class Groups	4	> 1000

Threshold ECDSA: Goal

- Rough costs with 256-bit curve, for each additional party (computation aggregated across [Gavenda 21, XAXYC 21, BMP 22]):

	2P-MUL	Bandwidth (KB)	Computation (ms)
Simple, unified protocol	OT	90	<10
	Paillier	15	Hundreds
	Class Groups	4	> 1000

This work:
3 Round Signing from 2 round 2P-MUL

Threshold ECDSA: Goal

- Rough costs with 256-bit curve, for each additional party (computation aggregated across [Gavenda 21, XAXYC 21, BMP 22]):

	2P-MUL	Bandwidth (KB)	Computation (ms)
Simple, unified protocol	OT	90	<10
	Paillier	15	Hundreds
	Class Groups	4	> 1000

<u>This work:</u>
3 Round Signing from 2 round 2P-MUL

mild/no overhead

Threshold ECDSA: Goal

- Rough costs with 256-bit curve, for each additional party (computation aggregated across [Gavenda 21, XAXYC 21, BMP 22]):

	2P-MUL	Bandwidth (KB)	Computation (ms)
Simple, unified protocol	OT	90	<10
	Paillier	15	Hundreds
	Class Groups	4	> 1000

This work:
3 Round Signing from 2 round 2P-MUL

mild/no overhead

Insight: well-chosen rewriting of ECDSA +simple consistency check

Threshold ECDSA: Goal

- Rough costs with 256-bit curve, for each additional party (computation aggregated across [Gavenda 21, XAXYC 21, BMP 22]):

	2P-MUL	Bandwidth (KB)	Computation (ms)
Simple, unified protocol	OT	60	<10
	Paillier	15	Hundreds
	Class Groups	4	> 1000

<u>This work:</u>
3 Round Signing from 2 round 2P-MUL

mild/no overhead

<u>Insight</u>:
well-chosen <u>rewriting</u> of ECDSA +simple consistency check

Intro		How to distribute ECDSA			Tradeoffs
MP-Schnorr is easy	but not ECDSA	Evolution of Techniques	ECDSA Tuples	Our protocol: Simple consistency check	

A Brief History of Threshold ECDSA

- "End result" protocols are typically compared by security models, assumptions, concrete efficiency (bandwidth, rounds), and benchmarks.

- This doesn't tell the full story of *techniques*
  $\Rightarrow$ necessary context for "simplicity"

- Qualitative comparison: trace how Threshold ECDSA protocol structure has evolved over time

MPC for ECDSA

- Computing $[k^{-1}]$ given $[k]$ (as used in ECDSA signing) naively as an arithmetic circuit is prohibitively expensive—warrants custom protocols

- Standard recipe in the literature:
  - Rewrite ECDSA signing equation to an "MPC-friendly" equivalent i.e. only additions and multiplications of secret values
  - Cryptographic Machinery for secure multiplication
  - Verify that all operations were performed honestly

Inverted Nonce Rewriting

[Langford 95][Gennaro Jarecki Krawczyk Rabin 96]

$$\mathrm{ECDSASign}(\mathsf{sk}, m):$$

$$[k] \leftarrow \mathbb{Z}_q$$

$$R = [k] \cdot G$$

$$e = H(m)$$

$$s = \frac{e + [\mathsf{sk}] \cdot r_x}{[k]}$$

$$\text{output } \sigma = (s, R)$$

Inverted Nonce Rewriting

[Langford 95][Gennaro Jarecki Krawczyk Rabin 96]

$$\mathrm{ECDSASign}(\mathsf{sk}, m):$$

$$[k] \leftarrow \mathbb{Z}_q$$

$$R = [k] \cdot G$$

$$e = H(m)$$

$$s = (e + [\mathsf{sk}] \cdot r_x)\,[k]$$

$$\text{output } \sigma = (s, R)$$

Inverted Nonce Rewriting

[Langford 95][Gennaro Jarecki Krawczyk Rabin 96]

$$\mathrm{ECDSASign}(\mathsf{sk}, m):$$

$$[k] \leftarrow \mathbb{Z}_q$$

$$\boxed{R = [k] \cdot G}$$

$$e = H(m)$$

$$s = (e + [\mathsf{sk}] \cdot r_x)\,[k]$$

$$\text{output } \sigma = (s, R)$$

Inverted Nonce Rewriting

[Langford 95][Gennaro Jarecki Krawczyk Rabin 96]

$$\mathrm{ECDSASign}(\mathsf{sk}, m):$$

$$[k] \leftarrow \mathbb{Z}_q$$

$$\boxed{R = [k^{-1}] \cdot G}$$

$$e = H(m)$$

Equivalent to ECDSA

But how to securely compute $k^{-1}G$?

$$s = (e + [\mathsf{sk}] \cdot r_x)\,[k]$$

$$\text{output } \sigma = (s, R)$$

Inverted Nonce Rewriting

[Langford 95][Gennaro Jarecki Krawczyk Rabin 96]

$\mathsf{ECDSASign}(\mathsf{sk}, m):$

$[k] \leftarrow \mathbb{Z}_q$

$[\phi] \leftarrow \mathbb{Z}_q$

reveal $[\phi] \cdot [k]$

reveal $\Phi = \phi \cdot G$

$R = (\phi k)^{-1} \cdot \Phi = [k^{-1}] \cdot G$

$e = H(m)$

$s = (e + [\mathsf{sk}] \cdot r_x)[k]$

output $\sigma = (s, R)$

First appears in [Bar-Ilan Beaver 89]

Equivalent to ECDSA

But how to securely compute $k^{-1}G$?

A Brief History of Threshold ECDSA

	1990s
	[Lan95, GJKR96]
Rewriting	Inverted Nonce
Machinery	Honest Majority Magic

A Brief History of Threshold ECDSA

1990s	[Lan95, GJKR96]	[MR01]	[GGN16, BGG17]	[Lin17]	[DKLs 18, 19]	2018
Rewriting	Inverted Nonce	Multiplicative				
Machinery	Honest Majority Magic	(Threshold) Paillier		Paillier	OT	

A Brief History of Threshold ECDSA

Now showing

1990s → 2018

	[Lan95, GJKR96]	[MR01]	[GGN16, BGG17]	[Lin17]	[DKLs 18, 19]	
Rewriting	Inverted Nonce	Multiplicative				
Machinery	Honest Majority Magic	(Threshold) Paillier		Paillier	OT	

A Brief History of Threshold ECDSA

Now showing MacKenzie Reiter 01

1990s	[Lan95, GJKR96]	[MR01]	[GGN16, BGG17]	[Lin17]	[DKLs 18, 19]	2018
Rewriting	Inverted Nonce	Multiplicative: $s = \left(\frac{a}{k}\right) + \left(\frac{b\mathsf{sk}}{k}\right)$				
Machinery	Honest Majority Magic	(Threshold) Paillier		Paillier	OT	

A Brief History of Threshold ECDSA

Now showing Gennaro Goldfeder Narayanan 16, Boneh, Gennaro, Goldfeder 17

1990s → 2018

	[Lan95, GJKR96]	[MR01]	[GGN16, BGG17]	[Lin17]	[DKLs 18, 19]
Rewriting	Inverted Nonce	Multiplicative: $s = \left(\frac{a}{k}\right) + \left(\frac{b\mathsf{sk}}{k}\right)$			
Machinery	Honest Majority Magic	(Threshold) Paillier		Paillier	OT

A Brief History of Threshold ECDSA

Now showing Lindell 17

1990s	[Lan95, GJKR96]	[MR01]	[GGN16, BGG17]	[Lin17]	[DKLs 18, 19]	2018
Rewriting	Inverted Nonce	Multiplicative: $s = \left(\frac{a}{k}\right) + \left(\frac{\mathsf{bsk}}{k}\right)$				
Machinery	Honest Majority Magic	(Threshold) Paillier		Paillier	OT	

A Brief History of Threshold ECDSA

Now showing Doerner, K, Lee, shelat 18 & 19

1990s → 2018

	[Lan95, GJKR96]	[MR01]	[GGN16, BGG17]	[Lin17]	[DKLs 18, 19]
Rewriting	Inverted Nonce	Multiplicative $s = \left(\frac{a}{k}\right) + \left(\frac{b\mathsf{sk}}{k}\right)$			
Machinery	Honest Majority Magic	(Threshold) Paillier		Paillier	OT

A Brief History of Threshold ECDSA

2018 now-ish

	[Lin17]	[DKLs 18, 19]	[GG 18, 20]	[CGGMP 20]	[HLNR 18, 23]	[ANOSS 22]	[ST19, DOKSS20]
Rewriting	Multiplicative		Inverted Nonce		ECDSA tuple		<flexible>
Machinery	Paillier	OT	Paillier		OT++	PCG	<flexible>

A Brief History of Threshold ECDSA

Now showing Gennaro Goldfeder 18 & 20

now-ish

	[Lin17]	[DKLs 18, 19]	[GG 18, 20]	[CGGMP 20]	[HLNR 18, 23]	[ANOSS 22]	[ST19, DOKSS20]
Rewriting	Multiplicative		Inverted Nonce		ECDSA tuple		<flexible>
Machinery	Paillier	OT	Paillier		OT++	PCG	<flexible>

A Brief History of Threshold ECDSA

Now showing Canetti, Gennaro, Goldfeder, Makriyannis, Peled 20 now-ish

	[Lin17]	[DKLs 18, 19]	[GG 18, 20]	[CGGMP 20]	[HLNR 18, 23]	[ANOSS 22]	[ST19, DOKSS20]
Rewriting	Multiplicative		Inverted Nonce		ECDSA tuple		<flexible>
Machinery	Paillier	OT	Paillier		OT++	PCG	<flexible>

A Brief History of Threshold ECDSA

Now showing Lindell Nof 18, Haitner, Lindell, Nof, Ranellucci 23

now-ish

	[Lin17]	[DKLs 18, 19]	[GG 18, 20]	[CGGMP 20]	[HLNR 18, 23]	[ANOSS 22]	[ST19, DOKSS20]
Rewriting	Multiplicative	Multiplicative	Inverted Nonce	Inverted Nonce	ECDSA tuple	ECDSA tuple	<flexible>
Machinery	Paillier	OT	Paillier	Paillier	OT++	PCG	<flexible>

A Brief History of Threshold ECDSA

Now showing: Abram Nof Orlandi Scholl Shlomovits 22 — now-ish

	[Lin17]	[DKLs 18, 19]	[GG 18, 20]	[CGGMP 20]	[HLNR 18, 23]	[ANOSS 22]	[ST19, DOKSS20]
Rewriting	Multiplicative		Inverted Nonce		ECDSA tuple		<flexible>
Machinery	Paillier	OT	Paillier		OT++	PCG	<flexible>

A Brief History of Threshold ECDSA

Now showing Smart Talibi 19, Dalskov, Orlandi, Keller, Shrishak, Shulman 20

	[Lin17]	[DKLs 18, 19]	[GG 18, 20]	[CGGMP 20]	[HLNR 18, 23]	[ANOSS 22]	[ST19, DOKSS20]
Rewriting	Multiplicative		Inverted Nonce		ECDSA tuple		<flexible>
Machinery	Paillier	OT	Paillier		OT++	PCG	<flexible>

A Brief History of Threshold ECDSA

now-ish

	[Lin17]	[DKLs 18, 19]	[GG 18, 20]	[CGGMP 20]	[HLNR 18, 23]	[ANOSS 22]	[ST19, DOKSS20]
Rewriting	Multiplicative		Inverted Nonce		ECDSA tuple		<flexible>
Machinery	Paillier	OT	Paillier		OT++	PCG	<flexible>
Verification	2P magic	Check relations in exponent	ZK in $\mathbb{Z}_N$ + Masked sig verification	GMW-style ZK in $\mathbb{Z}_N$	Replay in committed form ZK in $\mathbb{Z}_q$	BDOZ MAC	Any $\mathbb{Z}_q$ MAC

A Brief History of Threshold ECDSA

now-ish

	[Lin17]	[DKLs 18, 19]	[GG 18, 20]	[CGGMP 20]	[HLNR 18, 23]	[ANOSS 22]	[ST19, DOKSS20]
Rewriting	Multiplicative		Inverted Nonce		ECDSA tuple		<flexible>
Machinery	Paillier	OT	Paillier		OT++	PCG	<flexible>
Verification	2P magic	Check relations in exponent	ZK in $\mathbb{Z}_N$ + Masked sig verification	GMW-style ZK in $\mathbb{Z}_N$	Replay in committed form ZK in $\mathbb{Z}_q$	BDOZ MAC	Any $\mathbb{Z}_q$ MAC
	2P-only	Involved analysis	Interactive	Machinery-specific	Expensive proofs	UC ZK	Creating MACs is extra work

A Brief History of Threshold ECDSA

	now-ish
	This work
Rewriting	ECDSA tuple
Machinery	Any 2P-MUL
Verification	Cheap! Generic! Simple statistical check Straightforward analysis No extra work: re-uses byproduct of 2P-MUL

Rewriting ECDSA with ECDSA Tuples

[Lindell Nof Ranellucci 18] [Abram Nof Orlandi Scholl Shlomovits 22]

$$\mathsf{ECDSASign}(\mathsf{sk}, m):$$

$$[k] \leftarrow \mathbb{Z}_q$$

$$R = [k] \cdot G$$

$$e = H(m)$$

$$s = \frac{e + [\mathsf{sk}] \cdot r_x}{[k]}$$

$$\text{output } \sigma = (s, R)$$

Rewriting ECDSA with ECDSA Tuples

[Lindell Nof Ranellucci 18] [Abram Nof Orlandi Scholl Shlomovits 22]

$$\begin{aligned}
&\mathrm{ECDSASign}(\mathsf{sk}, m):\\
&\quad [k] \leftarrow \mathbb{Z}_q\\
&\quad R = [k] \cdot G\\
&\quad e = H(m)\\
&\quad s = \frac{e + [\mathsf{sk}] \cdot r_x}{[k]} \cdot \frac{[\phi]}{[\phi]}\\
&\quad \text{output } \sigma = (s, R)
\end{aligned}$$

Rewriting ECDSA with ECDSA Tuples

[Lindell Nof Ranellucci 18] [Abram Nof Orlandi Scholl Shlomovits 22]

$\mathsf{ECDSASign}(\mathsf{sk}, m):$

$$[k] \leftarrow \mathbb{Z}_q$$

$$R = [k] \cdot G$$

$$e = H(m)$$

$$\alpha = (e + [\mathsf{sk}] \cdot r_x)\,[\phi]$$

$$\beta = [k][\phi]$$

$$s =$$

$$\text{output } \sigma = (s, R)$$

Rewriting ECDSA with ECDSA Tuples

[Lindell Nof Ranellucci 18] [Abram Nof Orlandi Scholl Shlomovits 22]

$$\mathsf{ECDSASign}(\mathsf{sk}, m):$$

$$[k] \leftarrow \mathbb{Z}_q$$

$$R = [k] \cdot G$$

$$e = H(m)$$

$$\alpha = (e + [\mathsf{sk}] \cdot r_x)\,[\phi]$$

$$\beta = [k][\phi]$$

$$s = \frac{\alpha}{\beta}$$

$$\text{output } \sigma = (s, R)$$

Rewriting ECDSA with ECDSA Tuples

[Lindell Nof Ranellucci 18] [Abram Nof Orlandi Scholl Shlomovits 22]

$$\mathrm{ECDSASign}(\mathsf{sk}, m):$$

$$[k] \leftarrow \mathbb{Z}_q$$

$$R = [k] \cdot G$$

$$e = H(m)$$

$$[\phi] \leftarrow \mathbb{Z}_q$$

$$\alpha = (e + [\mathsf{sk}] \cdot r_x)\,[\phi]$$

$$\beta = [k][\phi]$$

$$s = \frac{\alpha}{\beta}$$

$$\text{output } \sigma = (s, R)$$

Rewriting ECDSA with ECDSA Tuples

[Lindell Nof Ranellucci 18] [Abram Nof Orlandi Scholl Shlomovits 22]

$$\mathrm{ECDSASign}(\mathsf{sk}, m):$$

$$[k] \leftarrow \mathbb{Z}_q$$

$$R = [k] \cdot G$$

$$e = H(m)$$

$$[\phi] \leftarrow \mathbb{Z}_q$$

Public values

$$\alpha = (e + [\mathsf{sk}] \cdot r_x)\,[\phi]$$

$$\beta = [k][\phi]$$

$$s = \frac{\alpha}{\beta}$$

$$\text{output } \sigma = (s, R)$$

Rewriting ECDSA with ECDSA Tuples

[Lindell Nof Ranellucci 18] [Abram Nof Orlandi Scholl Shlomovits 22]

$$\mathrm{ECDSASign}(\mathsf{sk}, m):$$

$$[k] \leftarrow \mathbb{Z}_q$$

$$R = [k] \cdot G$$

$$e = H(m)$$

$$[\phi] \leftarrow \mathbb{Z}_q$$

$$\alpha = (e + [\mathsf{sk}] \cdot r_x)\,[\phi]$$

$$\beta = [k][\phi]$$

$$s = \frac{\alpha}{\beta}$$

output $\sigma = (s, R)$

Public values

Safe to reveal
β: because ϕ is OTP
α: because fixed by β, s

Rewriting ECDSA with ECDSA Tuples

[Lindell Nof Ranellucci 18] [Abram Nof Orlandi Scholl Shlomovits 22]

$$\mathrm{ECDSASign}(\mathsf{sk}, m):$$

$$[k] \leftarrow \mathbb{Z}_q$$

$$R = [k] \cdot G$$

$$e = H(m)$$

$$[\phi] \leftarrow \mathbb{Z}_q$$

$$\alpha = (e + [\mathsf{sk}] \cdot r_x)\,[\phi]$$

$$\beta = [k][\phi]$$

$$s = \frac{\alpha}{\beta}$$

$$\text{output } \sigma = (s, R)$$

Public values

Safe to reveal
β: because ϕ is OTP
α: because fixed by β, s

<u>Secure mult</u>: Only (nonlinear) combination of secret values

Signing from ECDSA Tuples

[Abram Nof Orlandi Scholl Shlomovits 22]

$[\mathsf{sk}]$ $[k]$ $[\phi]$ $[\phi k]$ $[\phi\mathsf{sk}]$

Signing from ECDSA Tuples

[Abram Nof Orlandi Scholl Shlomovits 22]

$[\mathsf{sk}]$ $[k]$ $[\phi]$ $[\phi k]$ $[\phi \mathsf{sk}]$

Round 1

Round 2

Establish $R = [k] \cdot G$

Signing from ECDSA Tuples

[Abram Nof Orlandi Scholl Shlomovits 22]

$[\mathsf{sk}]$ $[k]$ $[\phi]$ $[\phi k]$ $[\phi\mathsf{sk}]$

Round 1

Round 2

Establish $R = [k] \cdot G$

Round 3

Reveal $\alpha = e + r_x[\phi\mathsf{sk}]$ and $\beta = [\phi k]$

Output $(R, s = \alpha/\beta)$

ECDSA Tuple Generation

$[\mathsf{sk}][k]$

$[\phi]$

$[\phi k]$

$[\phi \mathsf{sk}]$

ECDSA Tuple Generation

Input : $[\mathsf{sk}][k]$

Sample : $[\phi]$

$$[\phi k]$$

$$[\phi \mathsf{sk}]$$

ECDSA Tuple Generation

Local

Input : $[\mathsf{sk}][k]$

Sample : $[\phi]$

$[\phi k]$

$[\phi \mathsf{sk}]$

ECDSA Tuple Generation

Local

$$\text{Input}: [\mathsf{sk}][k]$$
$$\text{Sample}: [\phi]$$

$$[\phi k] = \mathsf{MULT}([\phi], [k])$$
$$[\phi \mathsf{sk}] = \mathsf{MULT}([\phi], [\mathsf{sk}])$$

ECDSA Tuple Generation

Local

$$\text{Input}: [\mathsf{sk}][k]$$
$$\text{Sample}: [\phi]$$

Consistency: straightforward

$$[\phi k] = \mathsf{MULT}([\phi],[k])$$
$$[\phi \mathsf{sk}] = \mathsf{MULT}([\phi],[\mathsf{sk}])$$

ECDSA Tuple Generation

Local

$$\text{Input} : [\mathsf{sk}][k]$$
$$\text{Sample} : [\phi]$$

Consistency: straightforward

$$[\phi k] = \mathsf{MULT}([\phi], [k])$$
$$[\phi \mathsf{sk}] = \mathsf{MULT}([\phi], [\mathsf{sk}])$$

Verify: $[k] \cdot G = R$

$[\mathsf{sk}] \cdot G = \mathsf{pk}$

ECDSA Tuple Generation

Local

$$\text{Input}: [\mathsf{sk}][k]$$
$$\text{Sample}: [\phi]$$

Consistency: straightforward

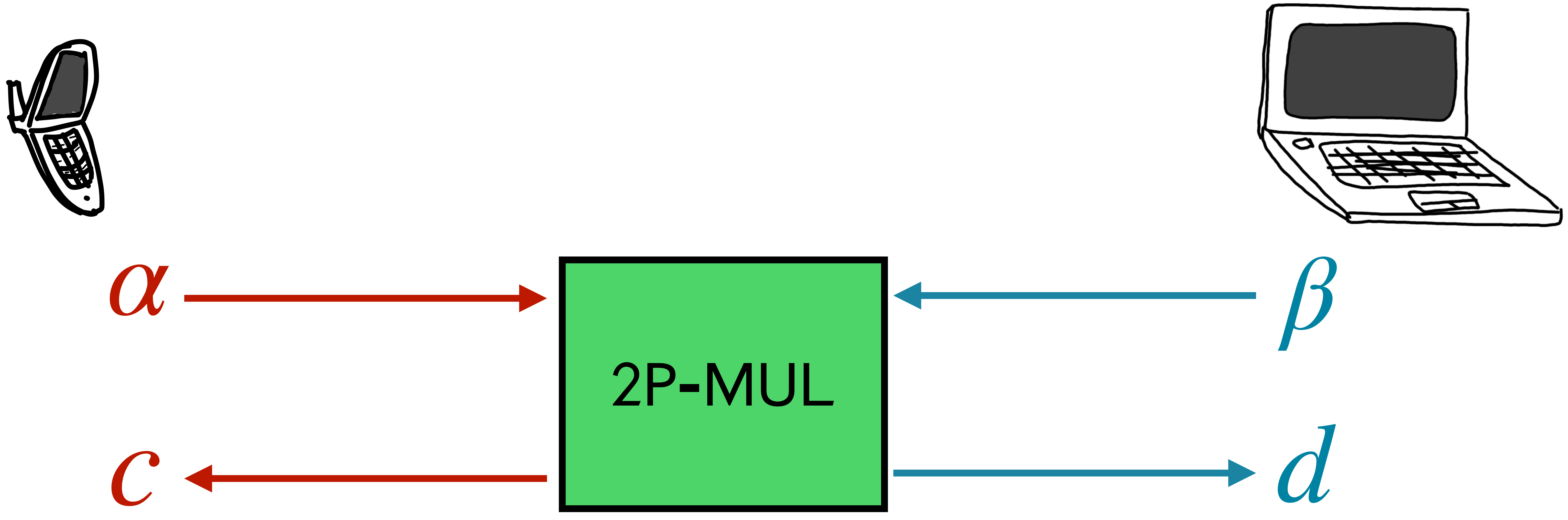
$$[\phi k] = \mathsf{MULT}([\phi], [k])$$
$$[\phi \mathsf{sk}] = \mathsf{MULT}([\phi], [\mathsf{sk}])$$

Previous works: ZK proofs, MACs, etc.
This work: Simple pairwise check

Verify: $[k] \cdot G = R$
$[\mathsf{sk}] \cdot G = \mathsf{pk}$

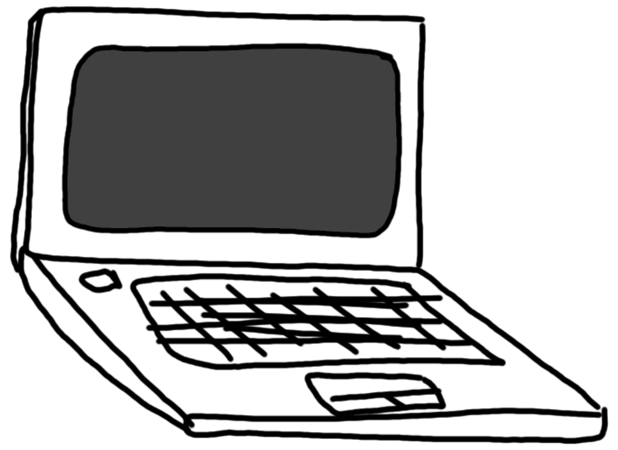
Secure Two-Party Multiplication

a.k.a. OLE, Mult2Add



$$c + d = \alpha \cdot \beta$$

Two-Round 2P-MUL



$$e = \mathsf{Msg}_1(\beta)$$

$$\longleftarrow$$

$$\longrightarrow$$

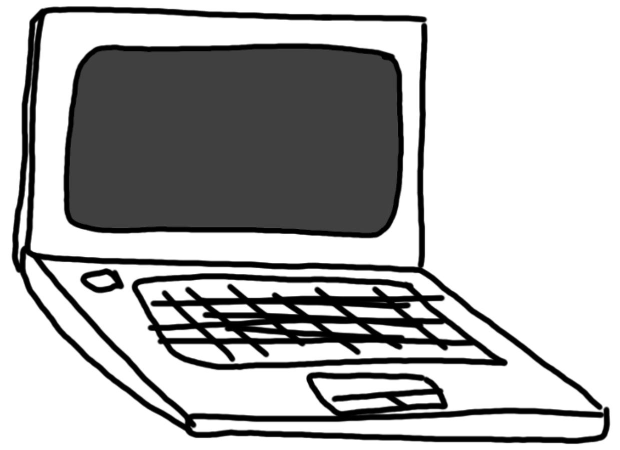
$$\mathsf{Msg}_2(e, \alpha)$$

c

d

$$c + d = \alpha \cdot \beta$$

Two-Round 2P-MUL



$e = \mathsf{Msg}_1(\beta)$

$\longleftarrow$

$\longrightarrow$

$\mathsf{Msg}_2(e, \alpha)$
$\mathsf{Msg}_2(e, \hat{\alpha})$

c

d

$$c + d = \alpha \cdot \beta$$

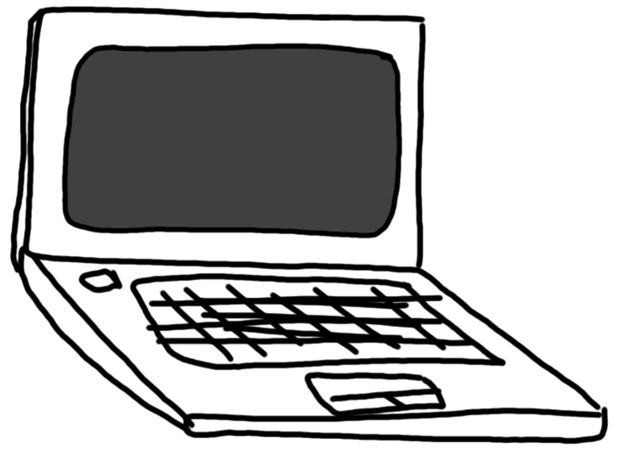
$$\hat{c} + \hat{d} = \hat{\alpha} \cdot \beta$$

Two-Round 2P-MUL

$$e = \mathsf{Msg}_1(\beta)$$

$$\mathsf{Msg}_2(e, \alpha)$$
$$\mathsf{Msg}_2(e, \hat{\alpha})$$

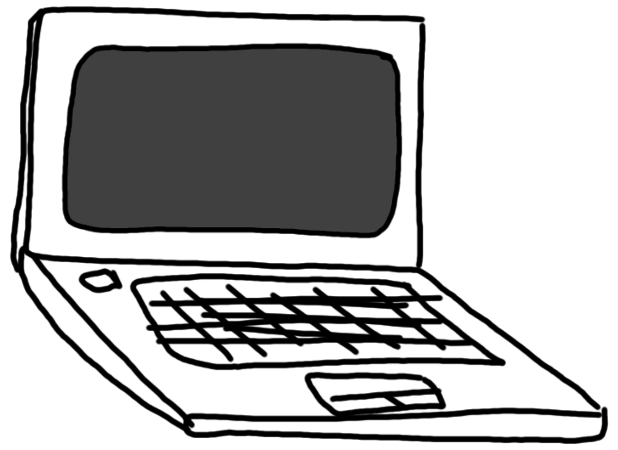
c d



$$c + d = \alpha \cdot \beta$$
$$\hat{c} + \hat{d} = \hat{\alpha} \cdot \beta$$

Consistency "for free"

Two-Round 2P-MUL



$e = \mathsf{Msg}_1(\beta)$

$\longleftarrow$

$\longrightarrow$

$\mathsf{Msg}_2(e, \alpha)$
$\mathsf{Msg}_2(e, \hat{\alpha})$

c

d

a.k.a. Vector OLE
(VOLE)

$$c + d = \alpha \cdot \beta$$
$$\hat{c} + \hat{d} = \hat{\alpha} \cdot \beta$$

Consistency
“for free”

ECDSA Tuple Generation

$$\text{Input} : [\mathsf{sk}][k]$$
$$\text{Sample} : [\phi]$$

Consistency: straightforward ✓

$$[\phi k] = \mathsf{MULT}([\phi], [k])$$
$$[\phi \mathsf{sk}] = \mathsf{MULT}([\phi], [\mathsf{sk}])$$

ECDSA Tuple Generation

$$\text{Input}: [\mathsf{sk}][k]$$
$$\text{Sample}: [\phi]$$

Consistency: straightforward ✓

$$[\phi k] = \mathsf{MULT}([\phi], [k])$$
$$[\phi \mathsf{sk}] = \mathsf{MULT}([\phi], [\mathsf{sk}])$$

Verify: $[k] \cdot G = R$
$[\mathsf{sk}] \cdot G = \mathsf{pk}$

ECDSA Tuple Generation

$\text{Input}: [\mathsf{sk}][k]$

$\text{Sample}: [\phi]$

Consistency: straightforward

ϕ is a MAC on k, sk

Verify MAC in $\mathbb{G}$

Byproduct of 2P-MUL: BDOZ MACs

Verify in parallel with MUL

$$[\phi k] = \mathsf{MULT}([\phi], [k])$$
$$[\phi \mathsf{sk}] = \mathsf{MULT}([\phi], [\mathsf{sk}])$$

Verify: $[k] \cdot G = R$

$[\mathsf{sk}] \cdot G = \mathsf{pk}$

Verifying Consistency w.r.t. $\mathbb{G}$

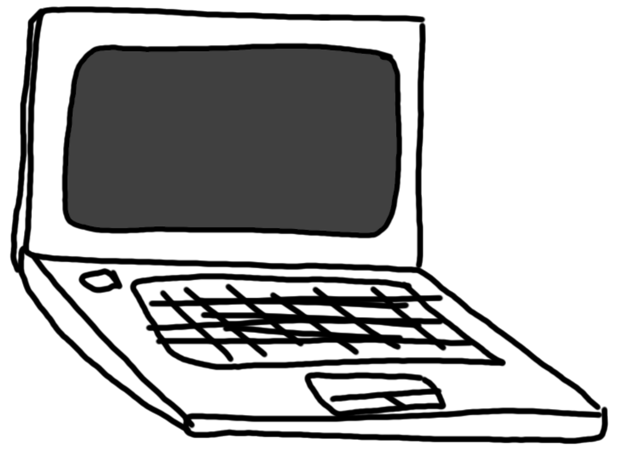
$\mathsf{MULT}([\phi], [k])$

Simplified :

ϕ

2P-MUL

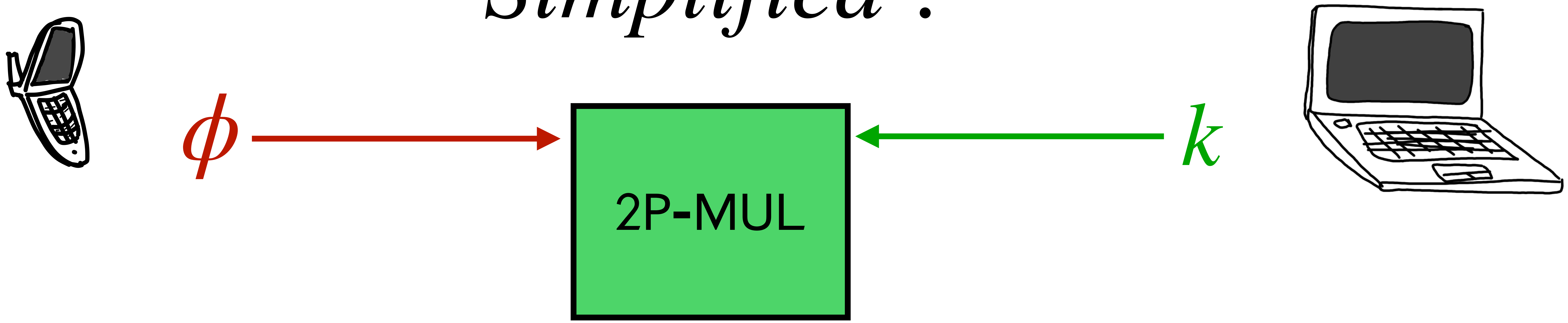
k



Verifying Consistency w.r.t. $\mathbb{G}$

$\mathsf{MULT}([\phi], [k])$

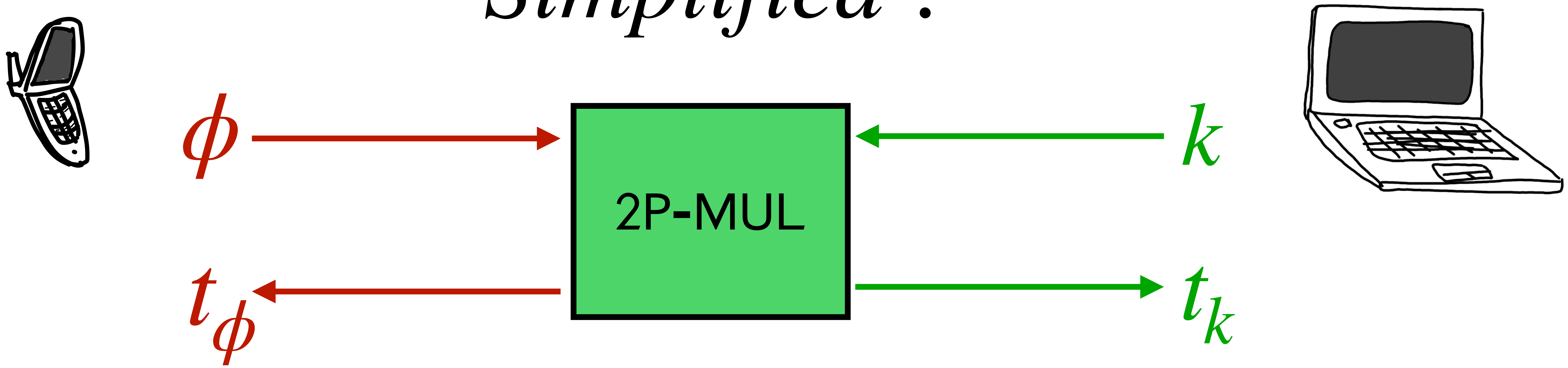
Simplified :



Verifying Consistency w.r.t. $\mathbb{G}$

$\mathsf{MULT}([\phi], [k])$

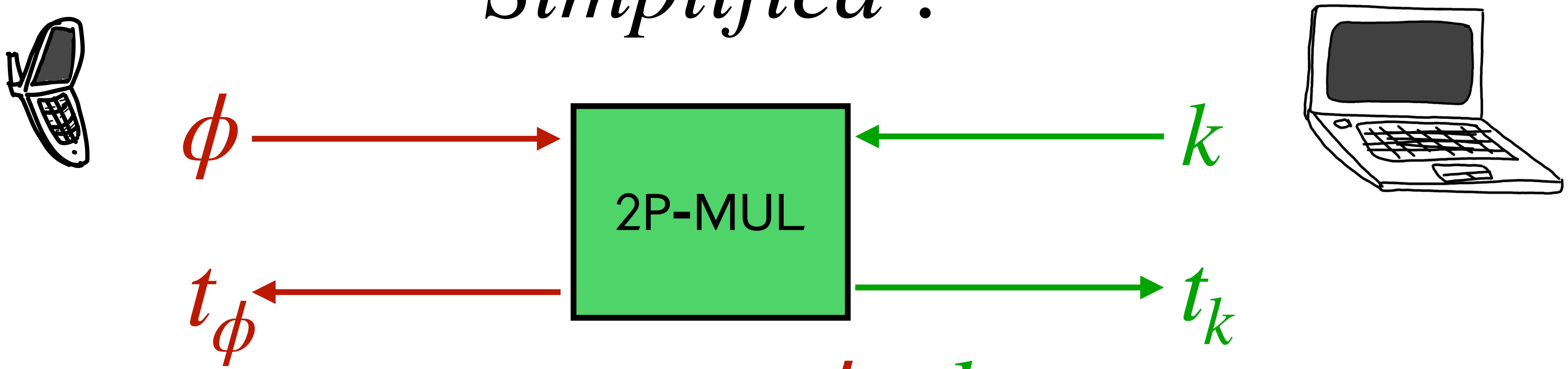
Simplified :



Verifying Consistency w.r.t. $\mathbb{G}$

$\mathsf{MULT}([\phi], [k])$

Simplified :

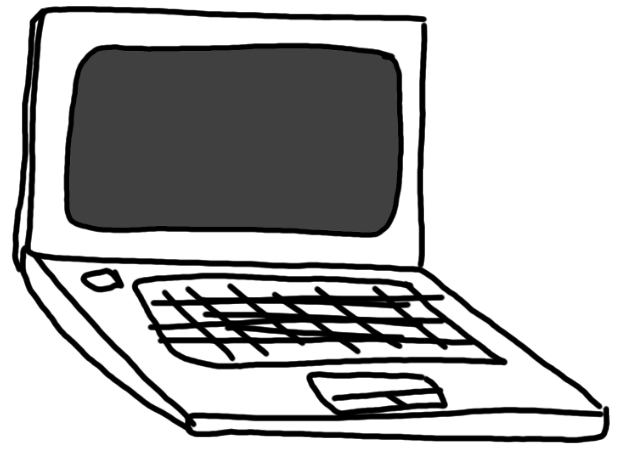


$$t_\phi + t_k = \phi \cdot k$$

Verifying Consistency w.r.t. $\mathbb{G}$

$\mathsf{MULT}([\phi], [k])$

Simplified :



$$\phi \rightarrow \boxed{\text{2P-MUL}} \leftarrow k$$

$$t_\phi \leftarrow \boxed{\text{2P-MUL}} \rightarrow t_k$$

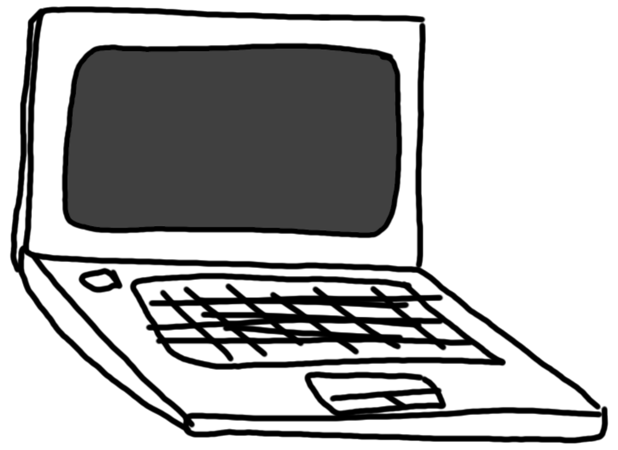
$$t_\phi + t_k = \phi \cdot k$$

Claim: $[k] \cdot G = R$

Verifying Consistency w.r.t. $\mathbb{G}$

$\mathrm{MULT}([\phi], [k])$

Simplified :



$\phi \longrightarrow$ 2P-MUL $\longleftarrow k$

$t_\phi \longleftarrow$ 2P-MUL $\longrightarrow t_k$

$$T_\phi = t_\phi \cdot G \qquad t_\phi + t_k = \phi \cdot k \qquad T_k = t_k \cdot G$$

Claim: $[k] \cdot G = R$

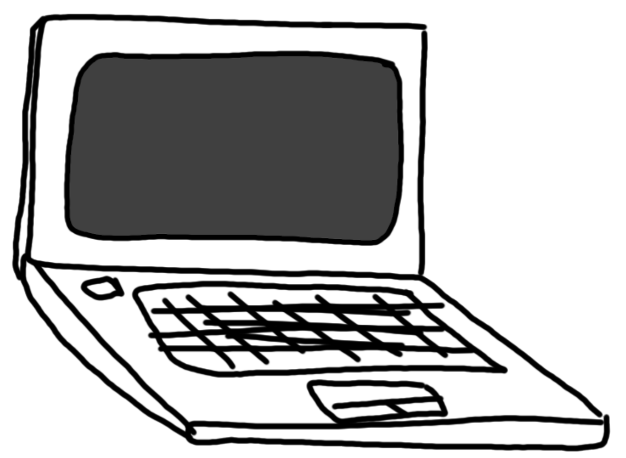
Verifying Consistency w.r.t. $\mathbb{G}$

MULT($[\phi]$, $[k]$)

Simplified :

$\phi \longrightarrow$ 2P-MUL $\longleftarrow k$

$t_\phi \longleftarrow$ 2P-MUL $\longrightarrow t_k$



$$t_\phi + t_k = \phi \cdot k$$

$$T_\phi = t_\phi \cdot G \qquad T_k = t_k \cdot G$$

Claim: $[k] \cdot G = R$

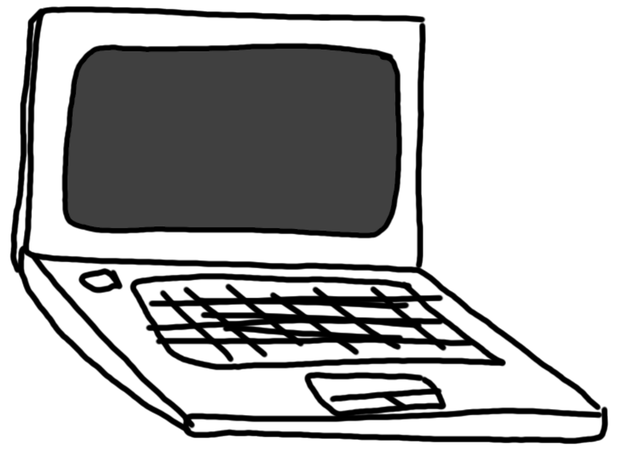
$$T_k = \phi R - T_\phi$$

Verifying Consistency w.r.t. $\mathbb{G}$

$\mathsf{MULT}([\phi], [k])$

Simplified :

$\phi \longrightarrow$ 2P-MUL $\longleftarrow k$



$t_\phi \longleftarrow$ 2P-MUL $\longrightarrow t_k$

$$T_\phi = t_\phi \cdot G \qquad t_\phi + t_k = \phi \cdot k \qquad T_k = t_k \cdot G$$

Claim: $[k] \cdot G = R$

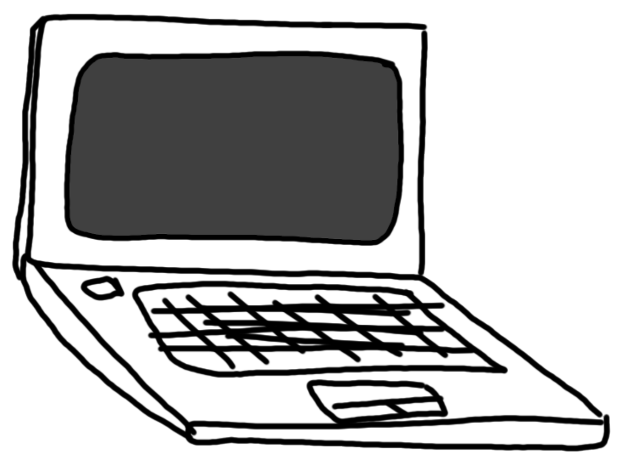
$$T_k = \phi R - T_\phi$$

$\xleftarrow{\quad T_k \quad}$

Verifying Consistency w.r.t. $\mathbb{G}$

MULT($[\phi]$, $[k]$)

Simplified :



$\phi \rightarrow$ 2P-MUL $\leftarrow k$

$t_\phi \leftarrow$ 2P-MUL $\rightarrow t_k$

$$T_\phi = t_\phi \cdot G \qquad t_\phi + t_k = \phi \cdot k \qquad T_k = t_k \cdot G$$

Claim: $[k] \cdot G = R$

$$\boxed{T_k} = \phi R - T_\phi$$

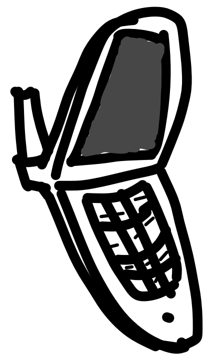
Match?

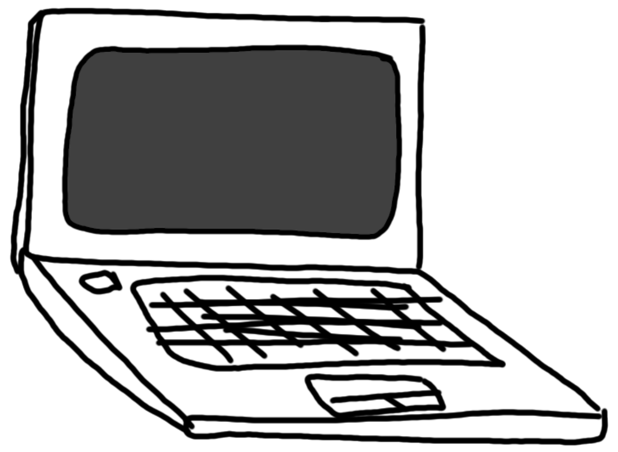
$\boxed{T_k}$

Verifying Consistency w.r.t. $\mathbb{G}$

$\mathsf{MULT}([\phi], [k])$

Simplified :





No information about k

$\phi \longrightarrow$ 2P-MUL $\longleftarrow k$

$t_\phi \longleftarrow$ 2P-MUL $\longrightarrow t_k$

$$T_\phi = t_\phi \cdot G \qquad t_\phi + t_k = \phi \cdot k \qquad T_k = t_k \cdot G$$

Claim: $[k] \cdot G = R$

$$\boxed{T_k} = \phi R - T_\phi$$

Match?

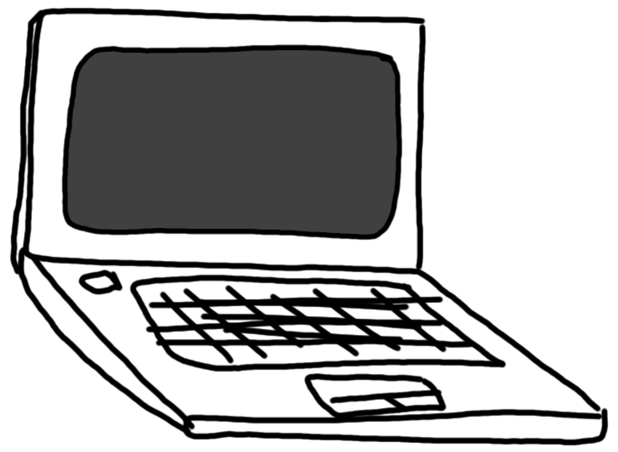
$\boxed{T_k}$

Verifying Consistency w.r.t. $\mathbb{G}$

$\mathsf{MULT}([\phi], [k])$

Simplified :

$\phi \longrightarrow$ **2P-MUL** $\longleftarrow k$



No information about k

$t_\phi \longleftarrow$ **2P-MUL** $\longrightarrow t_k$

$$T_\phi = t_\phi \cdot G \qquad t_\phi + t_k = \phi \cdot k \qquad T_k = t_k \cdot G$$

Claim: $[k] \cdot G = R$

$$\boxed{T_k} = \phi R - T_\phi$$

Match?

$\boxed{T_k}$

<u>In case of cheat:</u>
have to guess ϕ
(2^{-256} chance)

Verifying Consistency w.r.t. $\mathbb{G}$

$\mathsf{MULT}([\phi], [k])$

Simplified :

$\phi \longrightarrow$ 2P-MUL $\longleftarrow k$

$t_\phi \longleftarrow$ 2P-MUL $\longrightarrow t_k$

$$T_\phi = t_\phi \cdot G \qquad t_\phi + t_k = \phi \cdot k \qquad T_k = t_k \cdot G$$

Claimed $R^* = \Delta + k \cdot G$

Verifying Consistency w.r.t. $\mathbb{G}$

MULT([ϕ], [k])

Simplified :

$\phi \longrightarrow$ 2P-MUL $\longleftarrow k$

$t_\phi \longleftarrow$ 2P-MUL $\longrightarrow t_k$

$$t_\phi + t_k = \phi \cdot k$$

$$T_\phi = t_\phi \cdot G \qquad T_k = t_k \cdot G$$

Claimed $R^* = \Delta + k \cdot G$

$$T_k^* = \phi R^* - T_\phi$$

Verifying Consistency w.r.t. $\mathbb{G}$

$\mathsf{MULT}([\phi], [k])$

Simplified :

$\phi \longrightarrow$ 2P-MUL $\longleftarrow k$

$t_\phi \longleftarrow$ 2P-MUL $\longrightarrow t_k$

$$T_\phi = t_\phi \cdot G \qquad t_\phi + t_k = \phi \cdot k \qquad T_k = t_k \cdot G$$

Claimed $R^* = \Delta + k \cdot G$

$$T_k^* = \phi R^* - T_\phi$$
$$= T_k + \phi\Delta$$

Verifying Consistency w.r.t. $\mathbb{G}$

$\mathsf{MULT}([\phi],[k])$

Simplified :

$\phi \rightarrow$ 2P-MUL $\leftarrow k$

$t_\phi \leftarrow$ 2P-MUL $\rightarrow t_k$

$$T_\phi = t_\phi \cdot G \qquad t_\phi + t_k = \phi \cdot k \qquad T_k = t_k \cdot G$$

Claimed $R^* = \Delta + k \cdot G$

$$T_k^* = \phi R^* - T_\phi$$
$$= T_k + \phi\Delta$$

No information about ϕ
$\Rightarrow$ distributed uniformly in $\mathbb{Z}_q$

Verifying Consistency w.r.t. $\mathbb{G}$

MULT($[\phi]$, $[k]$)

Simplified :

$\phi \rightarrow$ 2P-MUL $\leftarrow k$

$t_\phi \leftarrow$ 2P-MUL $\rightarrow t_k$

$$T_\phi = t_\phi \cdot G \qquad t_\phi + t_k = \phi \cdot k \qquad T_k = t_k \cdot G$$

Claimed $R^* = \Delta + k \cdot G$

$$T_k^* = \phi R^* - T_\phi$$
$$= T_k + \phi\Delta$$

No information about ϕ
$\Rightarrow$ distributed uniformly in $\mathbb{Z}_q$

Statistically unlikely to send correct T_k^*

Notes on Consistency Check

- Case 1: Inconsistent k^*—almost certainly fails
  Case 2: Consistent k— nothing about ϕ leaked
  $\Rightarrow$ ϕ is a MAC key, but also safe to (re)use in ECDSA tuple

- Very cheap, cost superseded by 2P-MUL

- Exact same structure for $[\phi \mathsf{sk}]$ verification with pk

- Actual check: each party validates 2P-MUL inputs
  (i.e. shares of k, sk, ϕ) used by every counterparty

3 Round ECDSA Signing

[This work]

Sample $[k]$

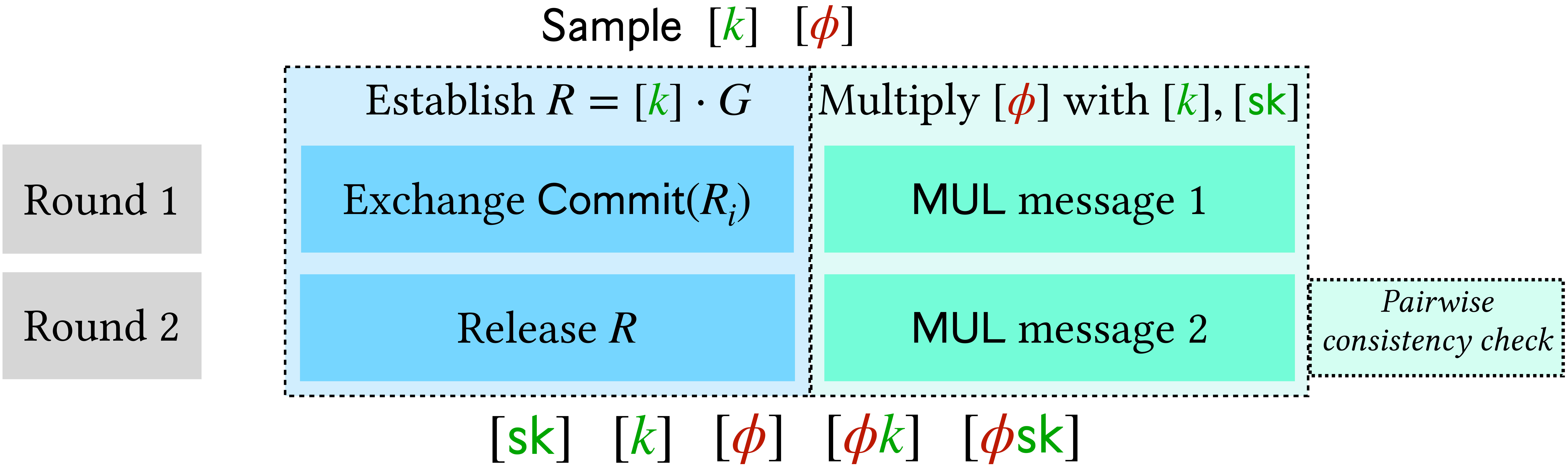
Establish $R = [k] \cdot G$

Round 1 — Exchange Commit(R_i)

Round 2 — Release R

3 Round ECDSA Signing

[This work]



3 Round ECDSA Signing

[This work]

Sample $[k]$ $[\phi]$

	Establish $R = [k] \cdot G$	Multiply $[\phi]$ with $[k], [\mathsf{sk}]$	
Round 1	Exchange $\mathsf{Commit}(R_i)$	MUL message 1	
Round 2	Release R	MUL message 2	*Pairwise consistency check*

$[\mathsf{sk}]$ $[k]$ $[\phi]$ $[\phi k]$ $[\phi \mathsf{sk}]$

Round 3	Reveal $\alpha = e[\phi] + r_x[\phi\mathsf{sk}]$ and $\beta = [\phi k]$

Output $(R, \sigma = \alpha/\beta)$

Intro		How to distribute ECDSA			Tradeoffs
MP-Schnorr is easy	but not ECDSA	Evolution of Techniques	ECDSA Tuples	Our protocol: Simple consistency check	OT vs AHE

Instantiating Multiplication

- Secure n-party mult can be reduced to $2n$ instances of 2P-MUL
- 2P-MUL inherently requires public key crypto
- Broadly two approaches:
  - Additively Homomorphic Encryption
    (low bandwidth, high computation)
  - Oblivious Transfer
    (low computation, high bandwidth)

2P-MUL from Additively Homomorphic Encryption

- Additive Homomorphism: $\alpha \cdot \mathsf{Enc}(x) + \mathsf{Enc}(\beta) = \mathsf{Enc}(\alpha x + \beta)$

  [Gilboa 99]: Conceptually simple protocol for MUL from AHE
  [CGGMP 20]: Hardened for active security through ZK proofs

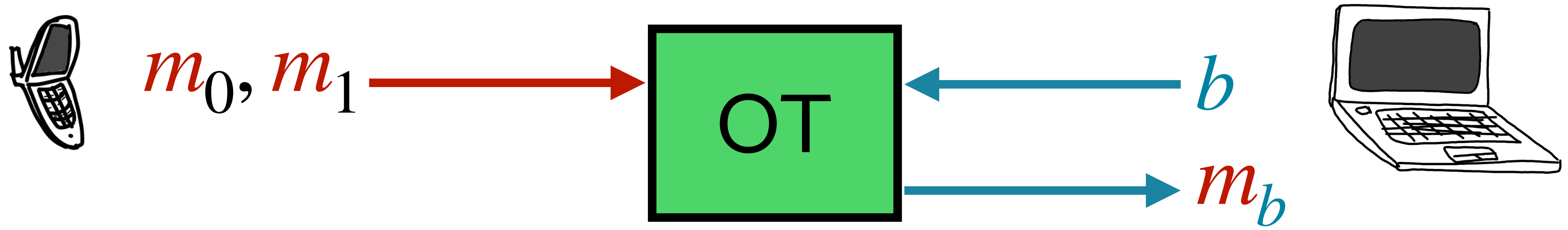
- Instantiations from factoring based cryptography (e.g. [Paillier 99]) and class groups [Castagnos Laguillaumie 15]

- <u>Advantages</u>: Parties exchange (relatively) compact ciphertexts

- <u>Downsides</u>:
  — Ciphertext operations are heavy (2 orders of magnitude slower than EC)
  — Seem to require ZK proofs to prevent misuse

2P-MUL from Oblivious Transfer

- Oblivious Transfer (OT):



  [Gilboa 99]: Elegant protocol for MUL from OT
  [DKLs 18,19, HMRT 22]: active security by randomized encoding+statistical checks

- Instantiable with ECDSA curve (think DH key exchange)

- Advantages: By OT Extension [IKNP03, Roy22] public key operations can be moved to one-time key generation phase, so only hashes when signing
  (1 order of magnitude slower than single party signing)

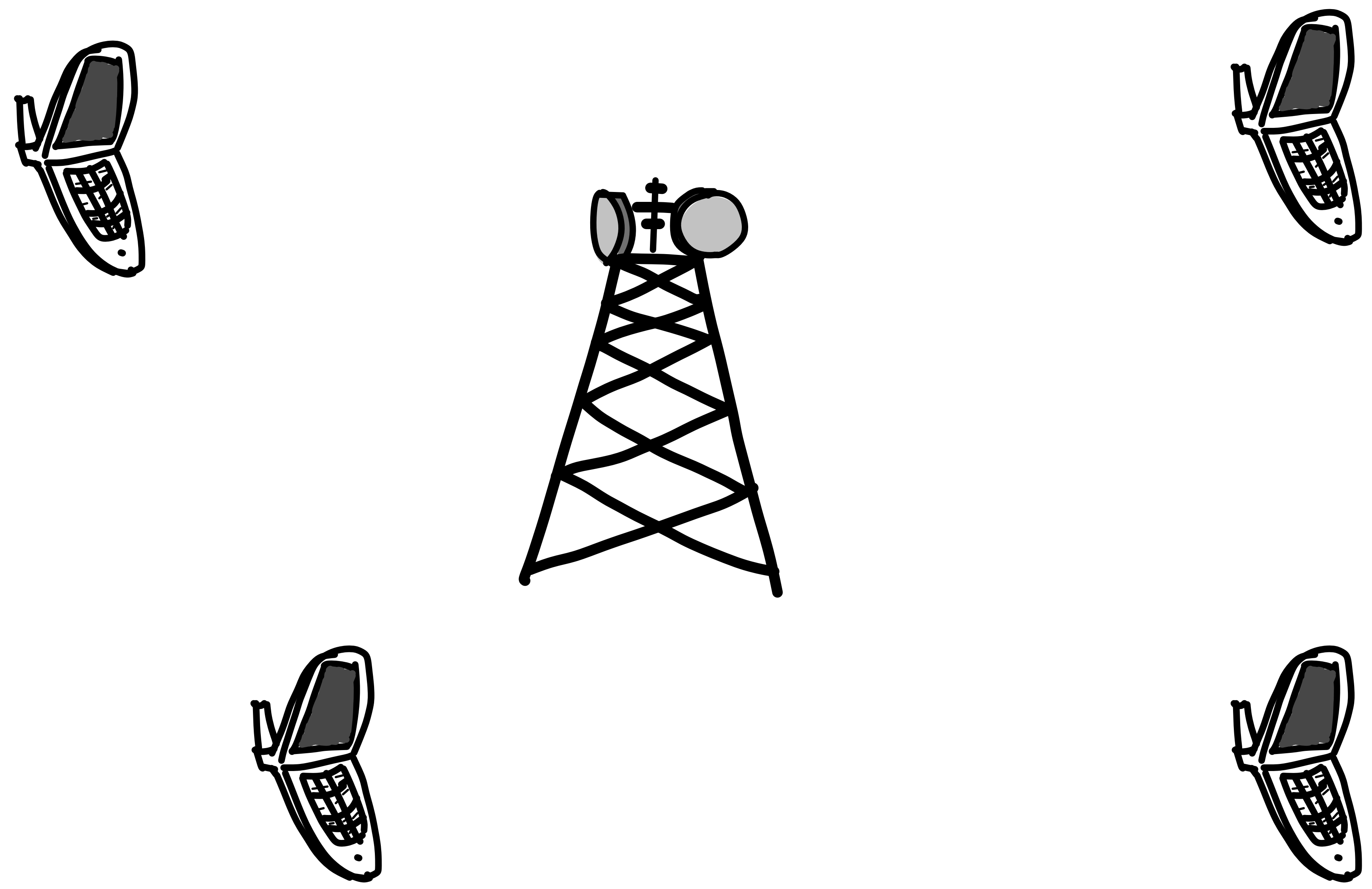
- Downsides: ~1000 OTs/sig, each transmits two $\mathbb{Z}_q$ elements

2P-MUL: AHE vs OT

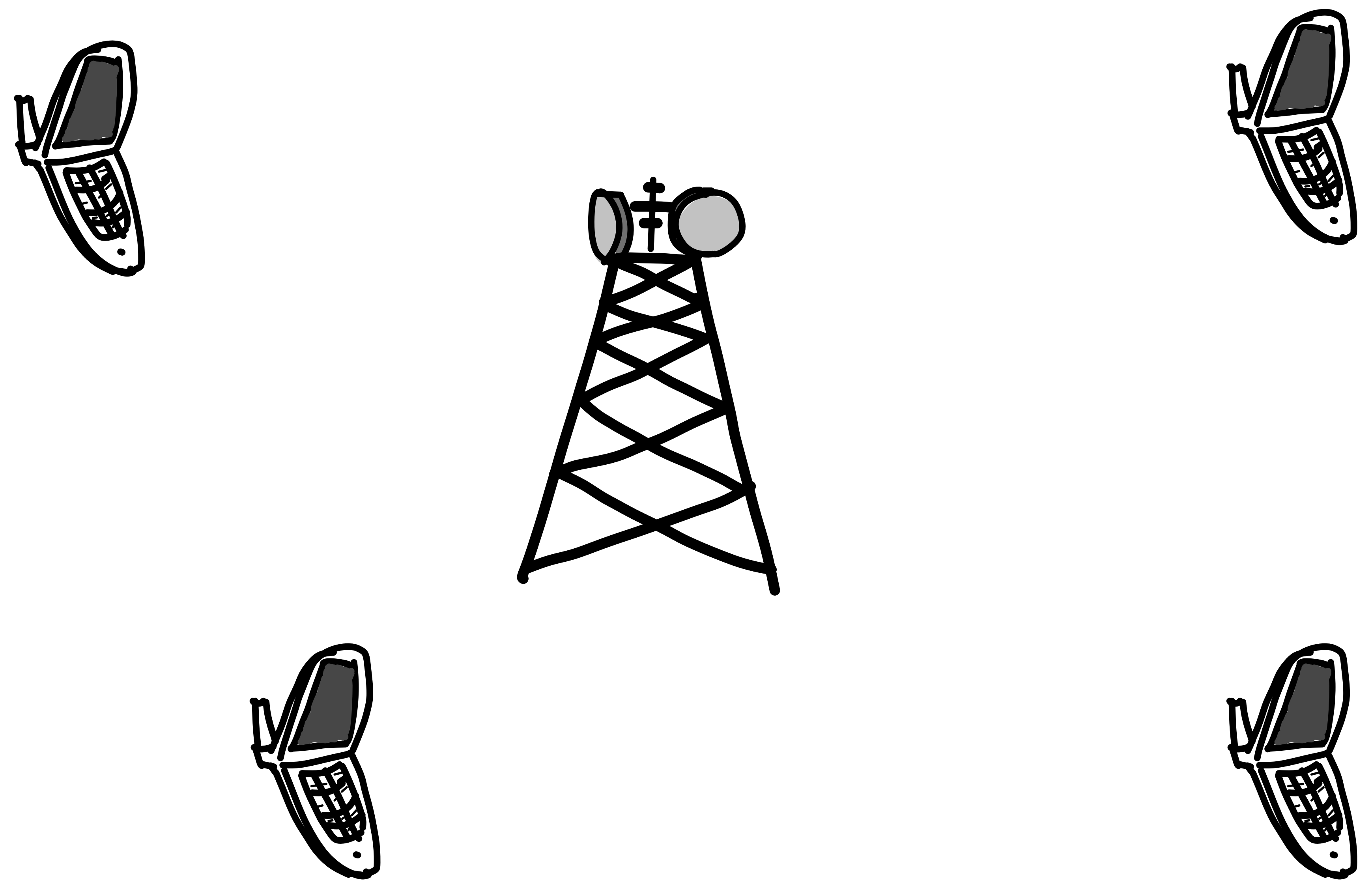
- Tradeoff to make: Computation vs. Bandwidth during signing time
- Rough costs with 256-bit curve, for each additional party (computation aggregated across [Gavenda 21, XAXYC 21, BMP 22]):

	Bandwidth	Computation
OT [DKLs 23]	60 KB	Few milliseconds
Paillier [CGGMP 20]	15 KB	Hundreds of milliseconds
Paillier [GG 18]	7 KB	Hundreds of milliseconds
Class Groups [CCLST20, YCX21]	4.5 KB	> 1 second

Is communication the bottleneck?



Is communication the bottleneck?



- **Mobile applications (human-initiated)**:

Is communication the bottleneck?

- **Mobile applications (human-initiated):**

Is communication the bottleneck?

- **Mobile applications (human-initiated)**:
  - eg. t=4, ~2Mbits transmitted per party

Is communication the bottleneck?

- **Mobile applications (human-initiated)**:
  - eg. t=4, ~2Mbits transmitted per party
  - Well within LTE envelope for responsivity

Example 1: Mobile Wallet

Multiplier: OT-based
Parties: 4
Curve: 256-bit

2 Mbits

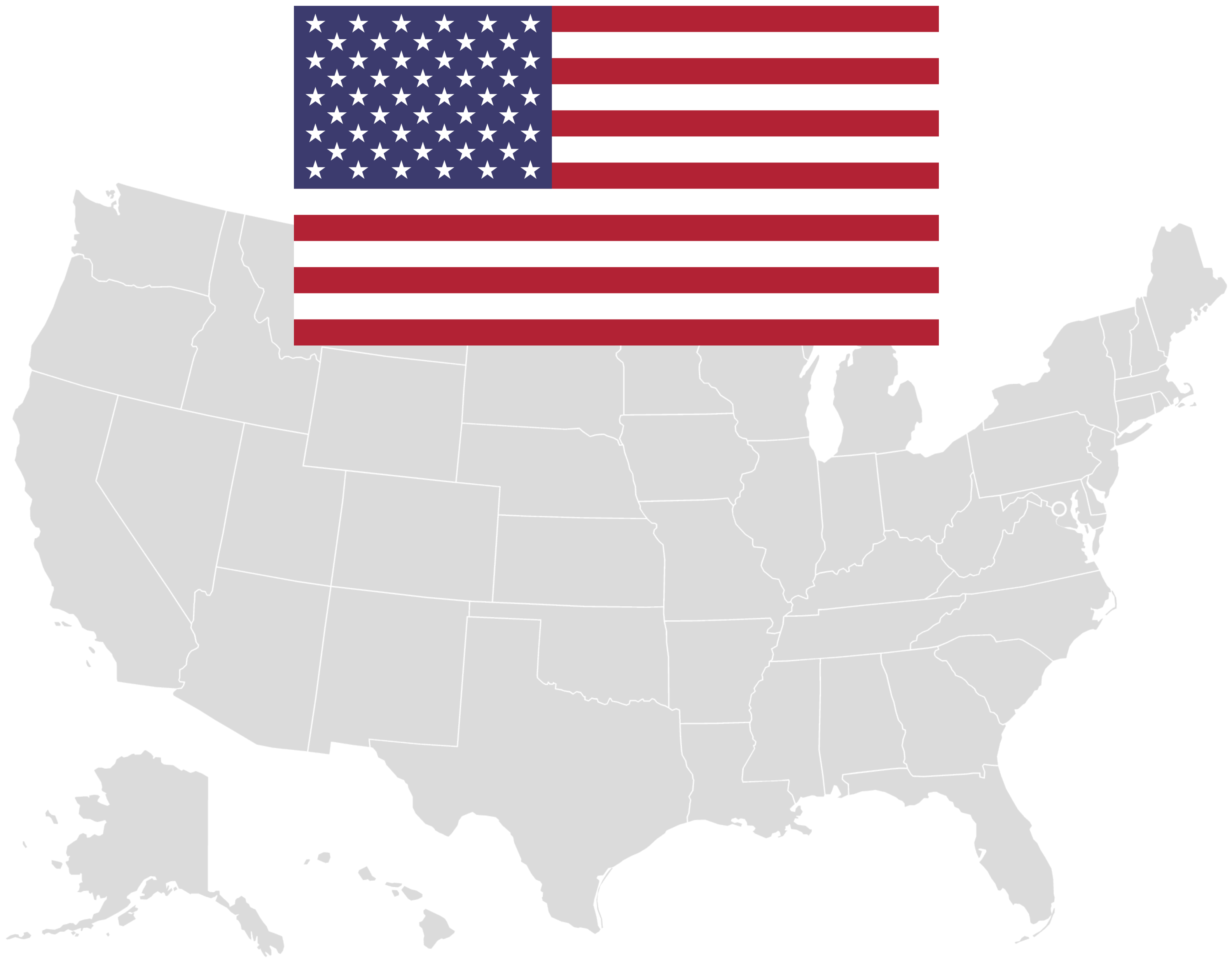
sent per party

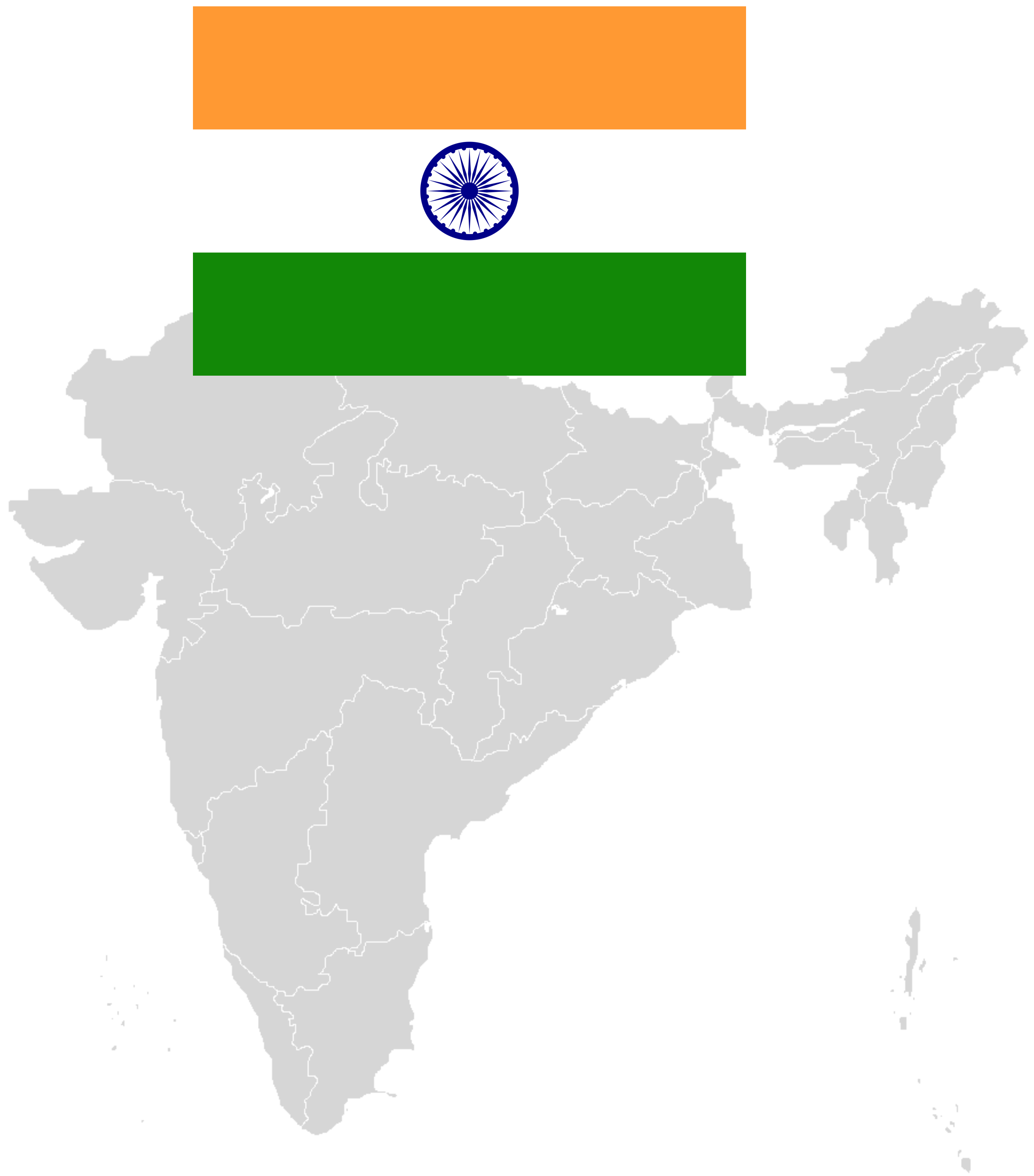
Example 1: Mobile Wallet

Multiplier: OT-based
Parties: 4
Curve: 256-bit

2 Mbits

sent per party

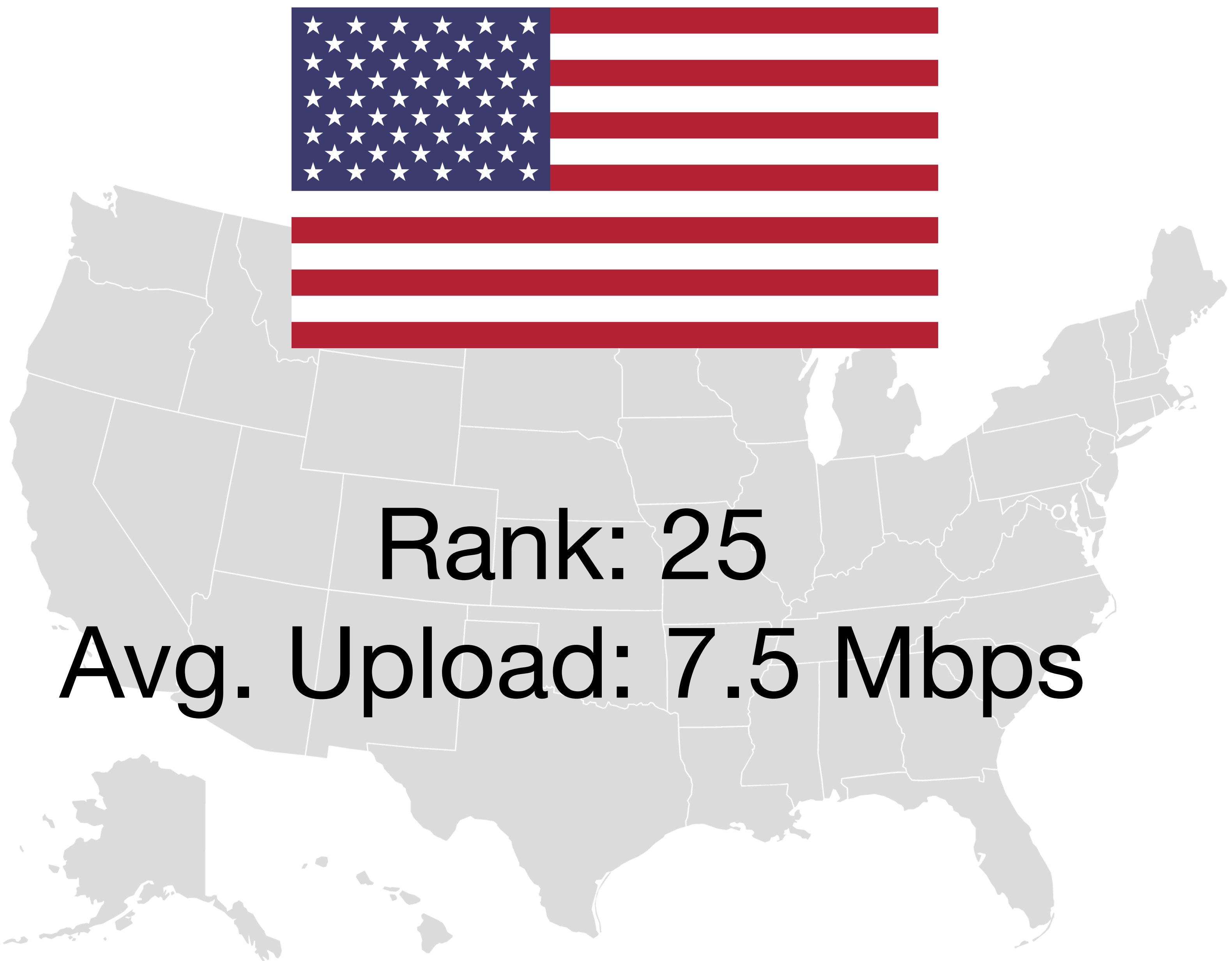


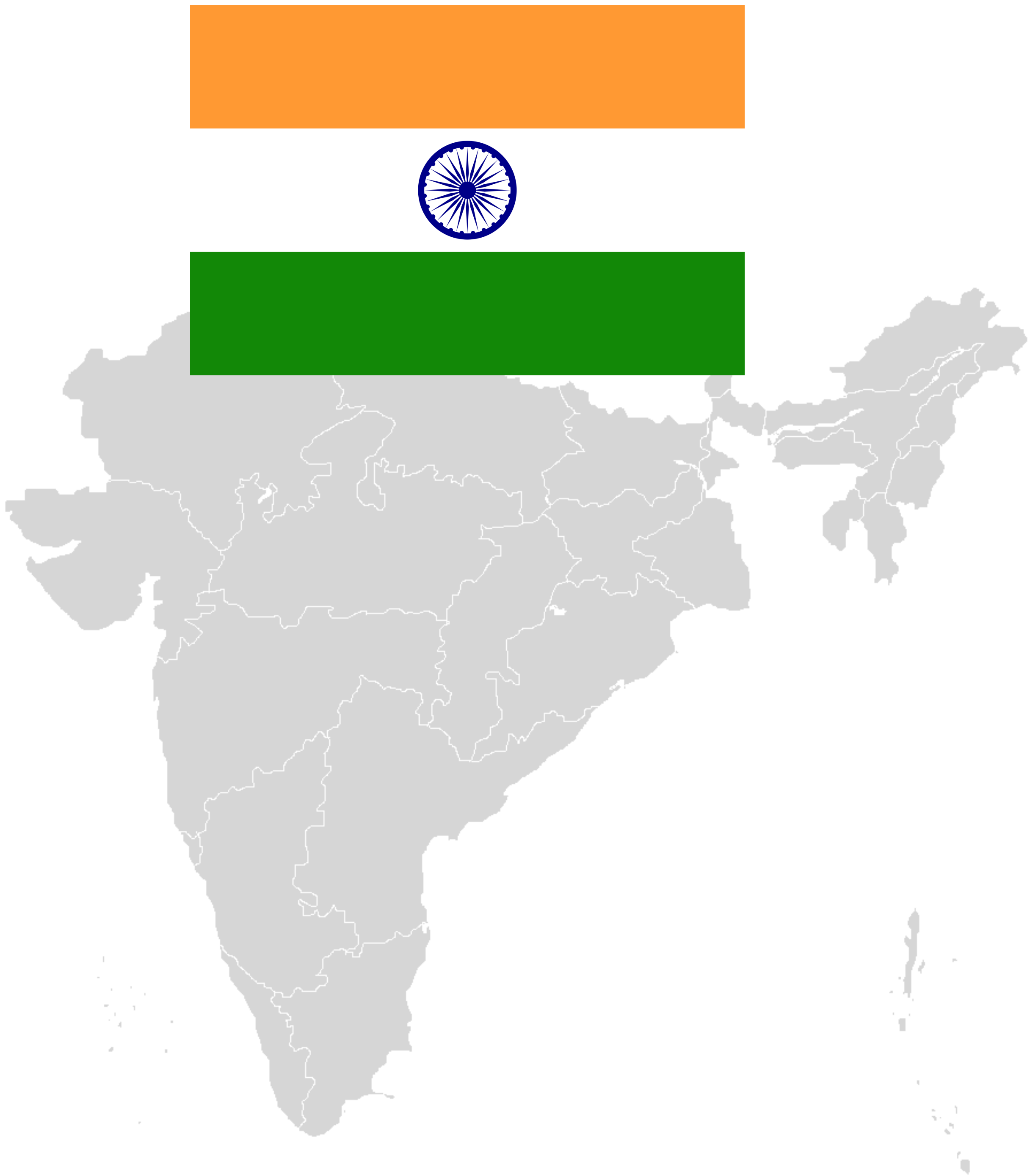


Example 1: Mobile Wallet

Multiplier: OT-based
Parties: 4
Curve: 256-bit

2 Mbits
sent per party



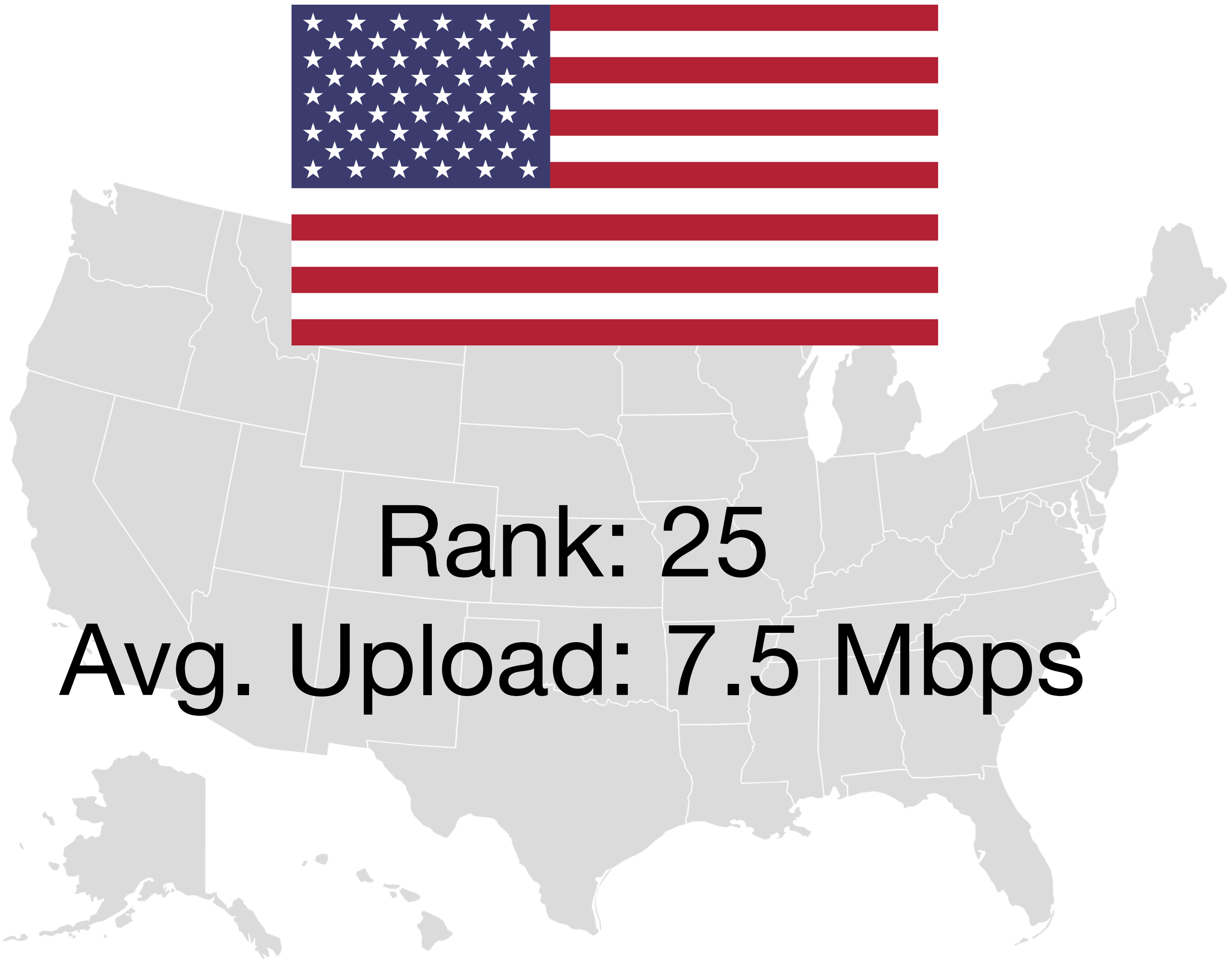


source: opensignal (2020)

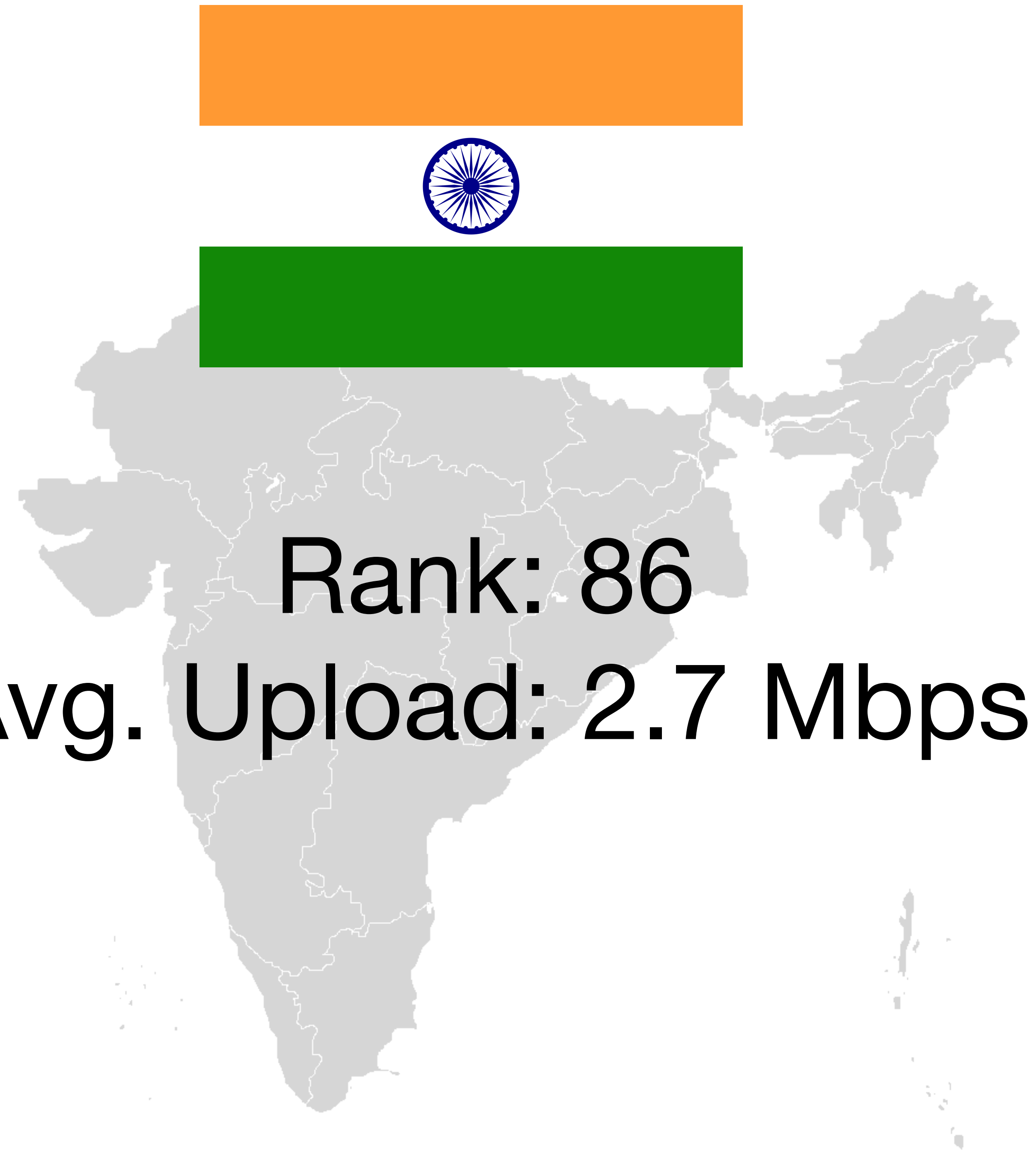
Example 1: Mobile Wallet

Multiplier: OT-based
Parties: 4
Curve: 256-bit

2 Mbits
sent per party



Rank: 25
Avg. Upload: 7.5 Mbps



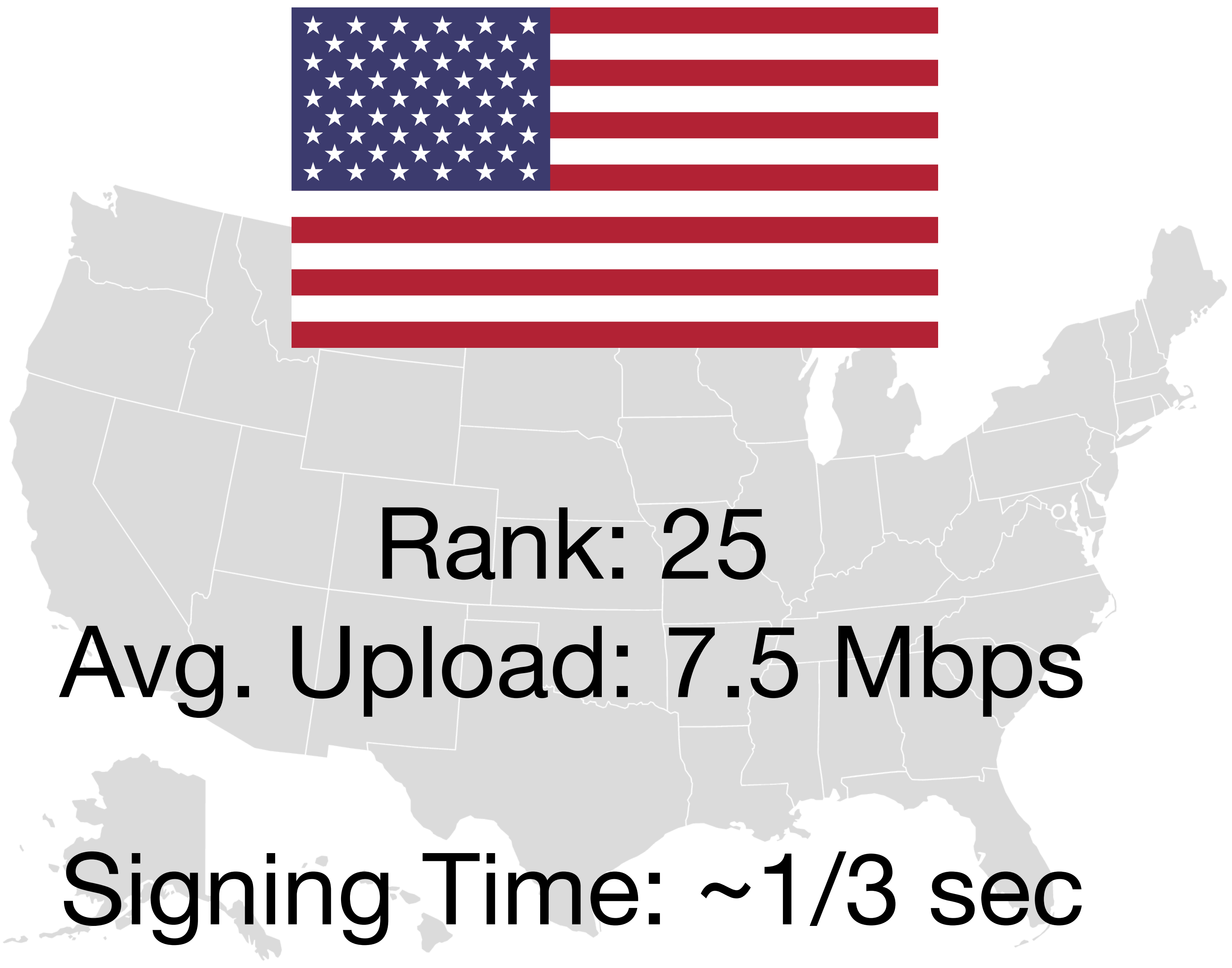
Rank: 86
Avg. Upload: 2.7 Mbps

source: opensignal (2020)

Example 1: Mobile Wallet

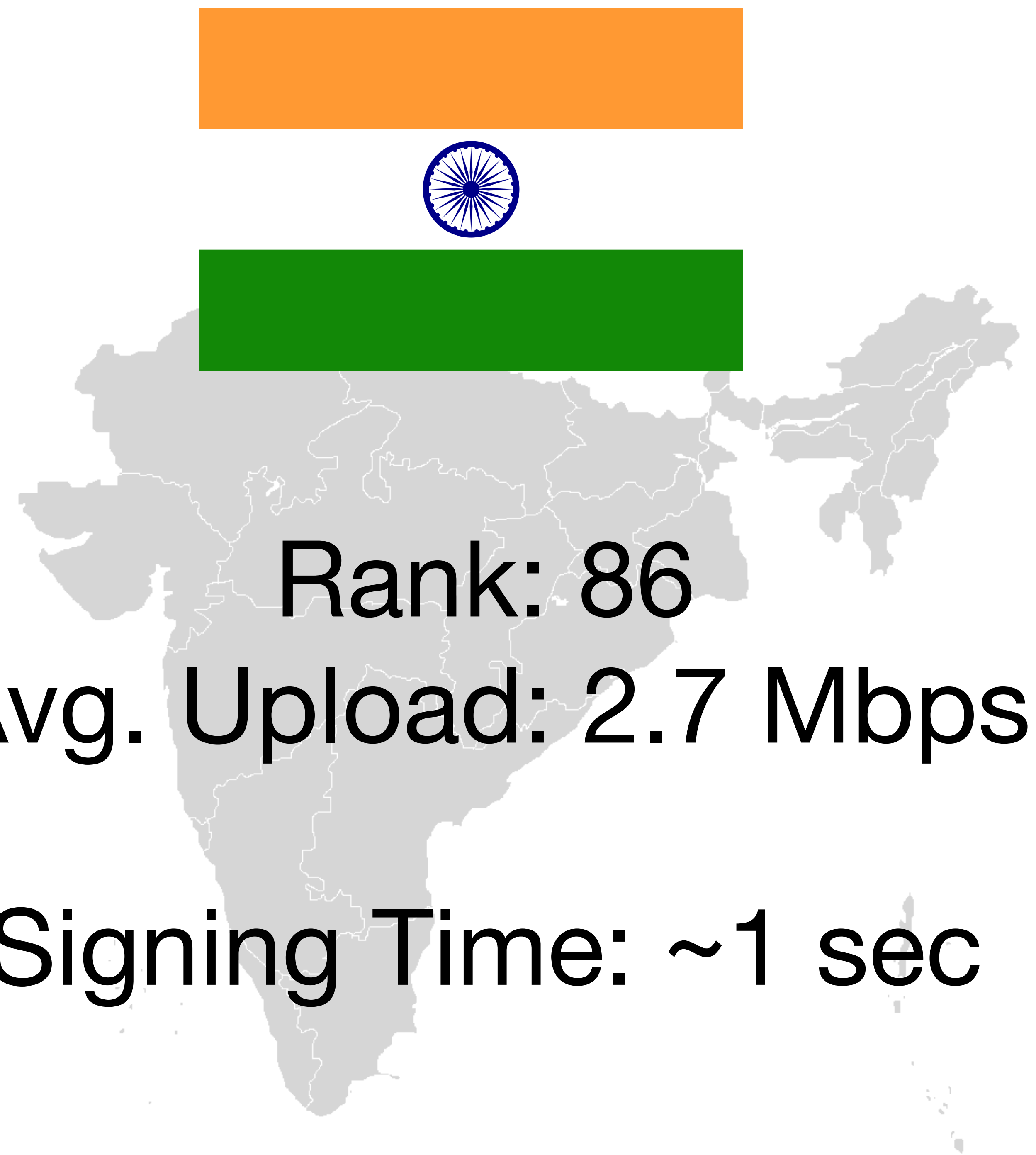
Multiplier: OT-based
Parties: 4
Curve: 256-bit

2 Mbits
sent per party



Rank: 25
Avg. Upload: 7.5 Mbps

Signing Time: ~1/3 sec



Rank: 86
Avg. Upload: 2.7 Mbps

Signing Time: ~1 sec

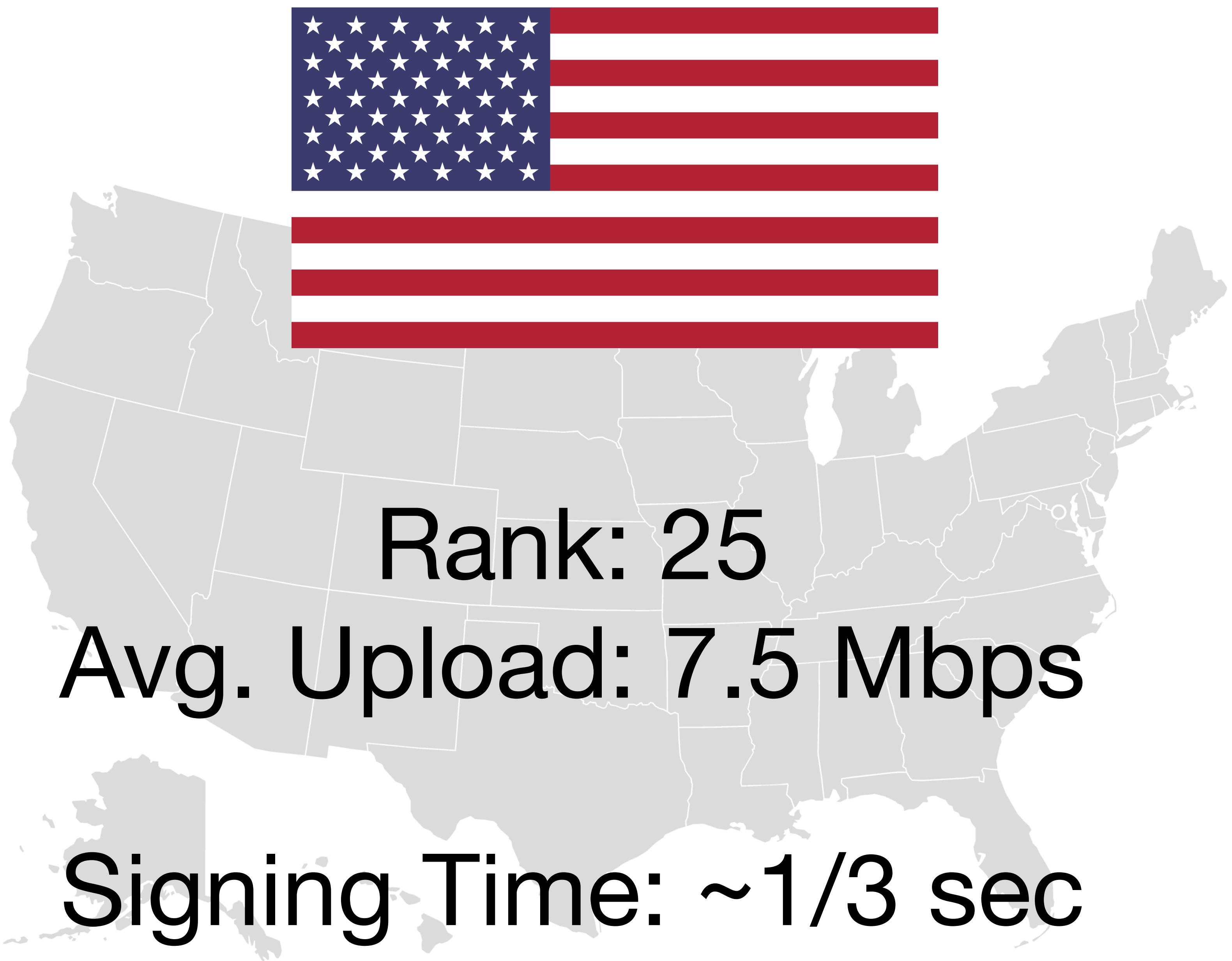
source: opensignal (2020)

Example 1: Mobile Wallet

Multiplier: OT-based
Parties: 4
Curve: 256-bit

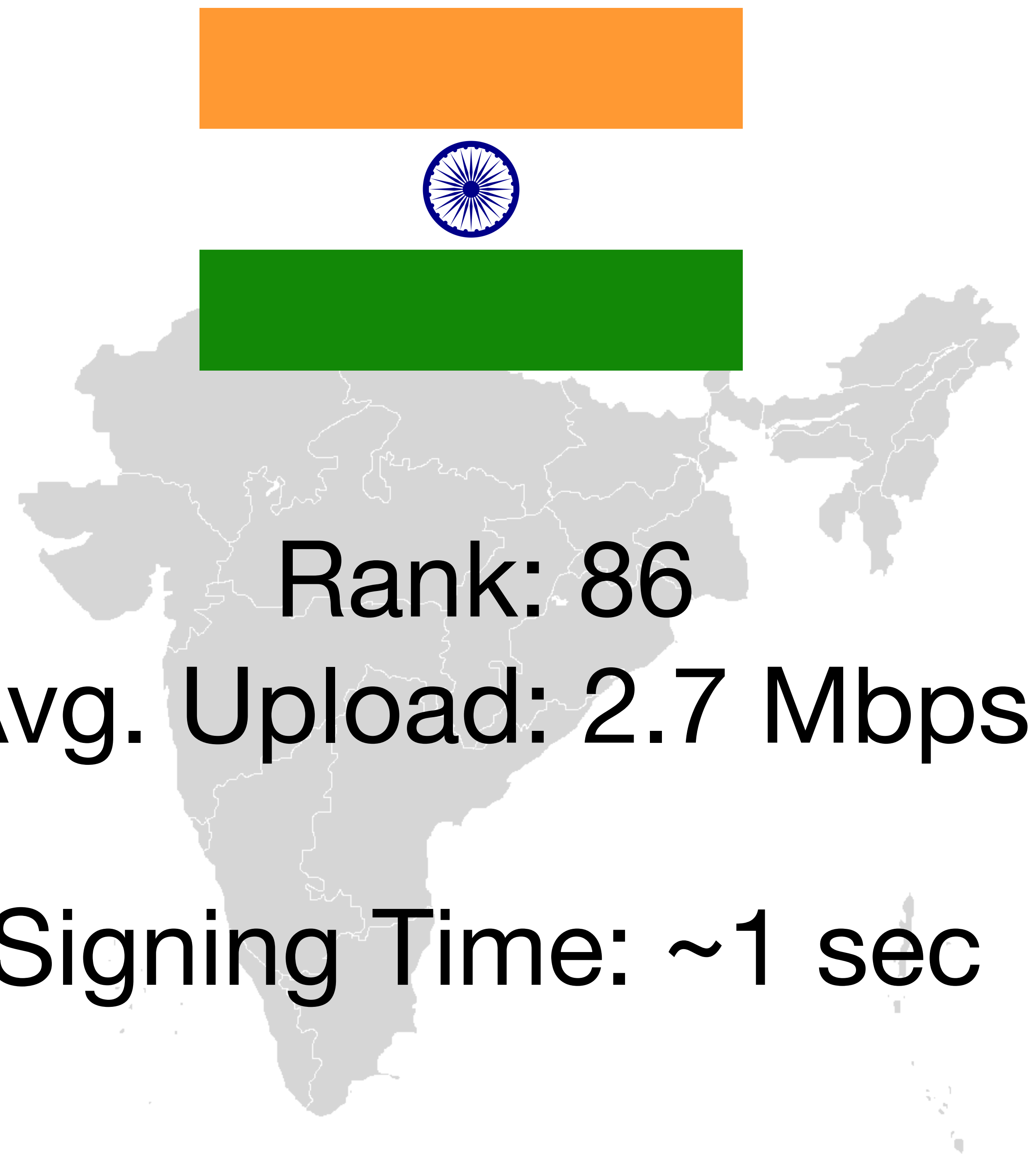
2 Mbits

sent per party



Rank: 25
Avg. Upload: 7.5 Mbps

Signing Time: ~1/3 sec



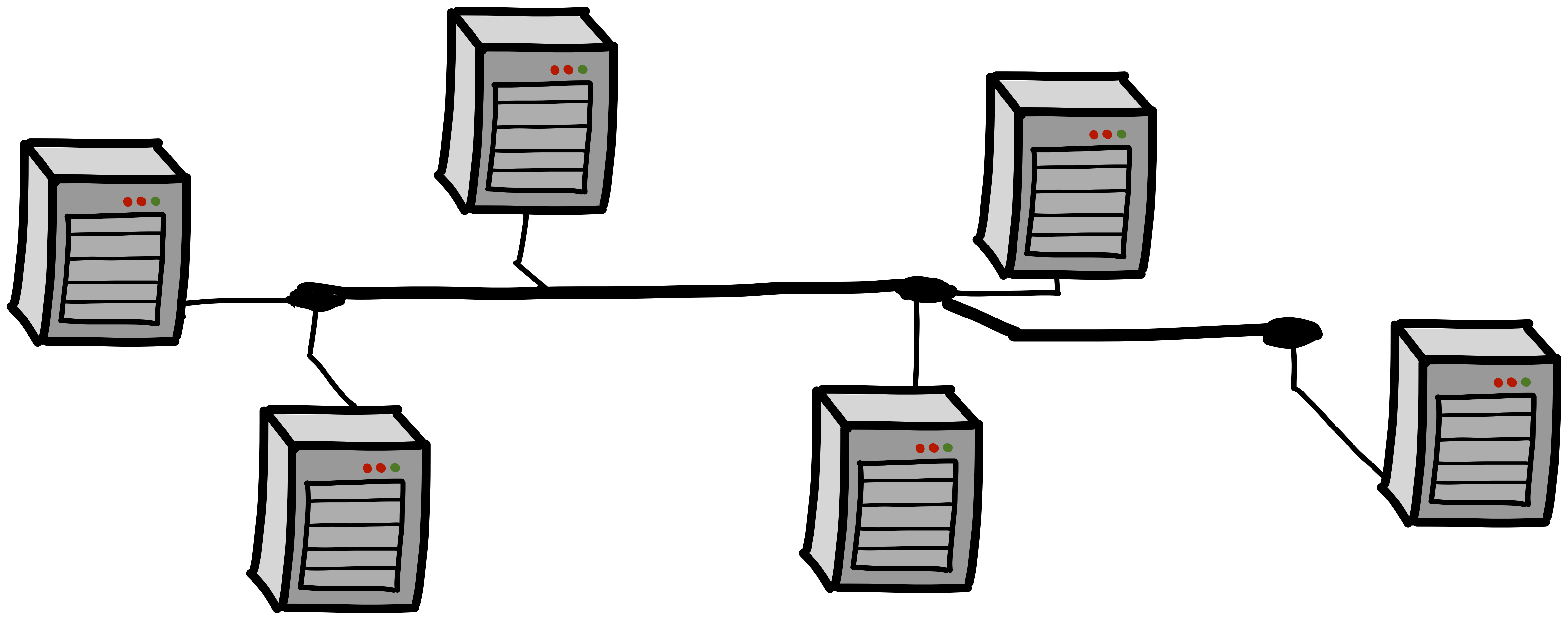
Rank: 86
Avg. Upload: 2.7 Mbps

Signing Time: ~1 sec

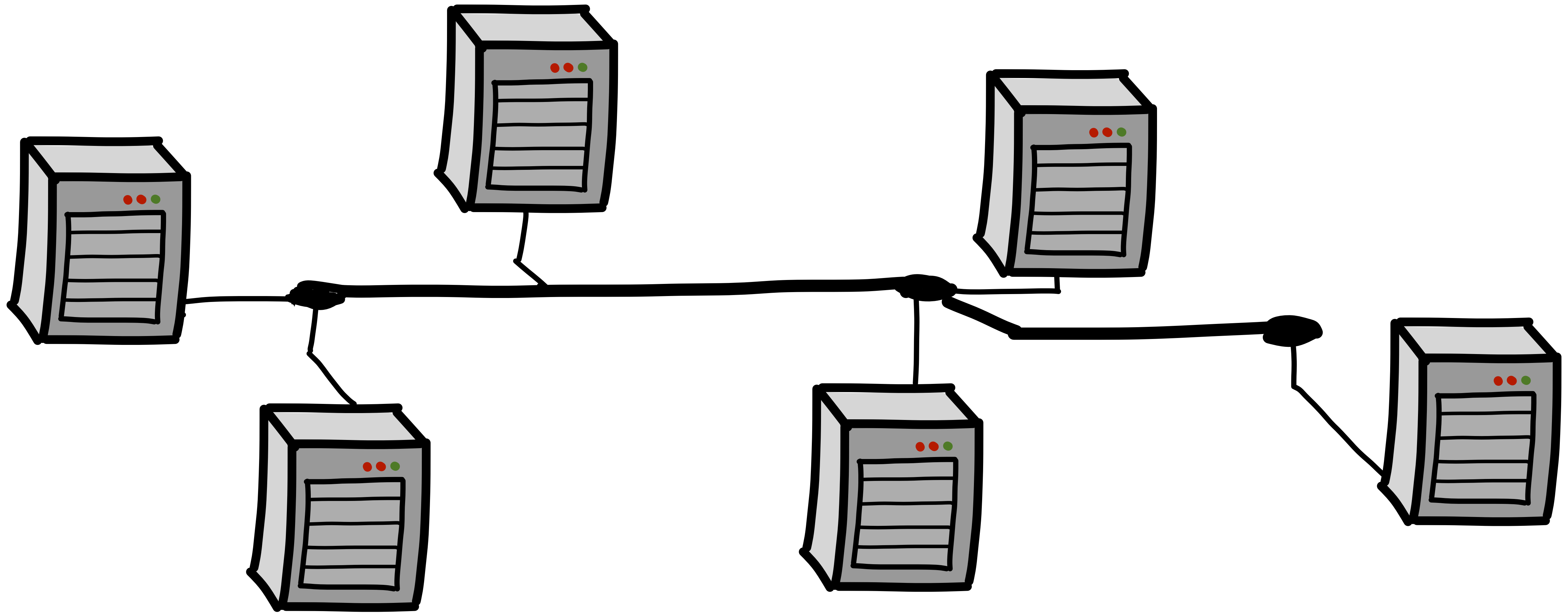
Paillier+ZK takes this long for computation alone on powerful hardware!

source: opensignal (2020)

Is communication the bottleneck?

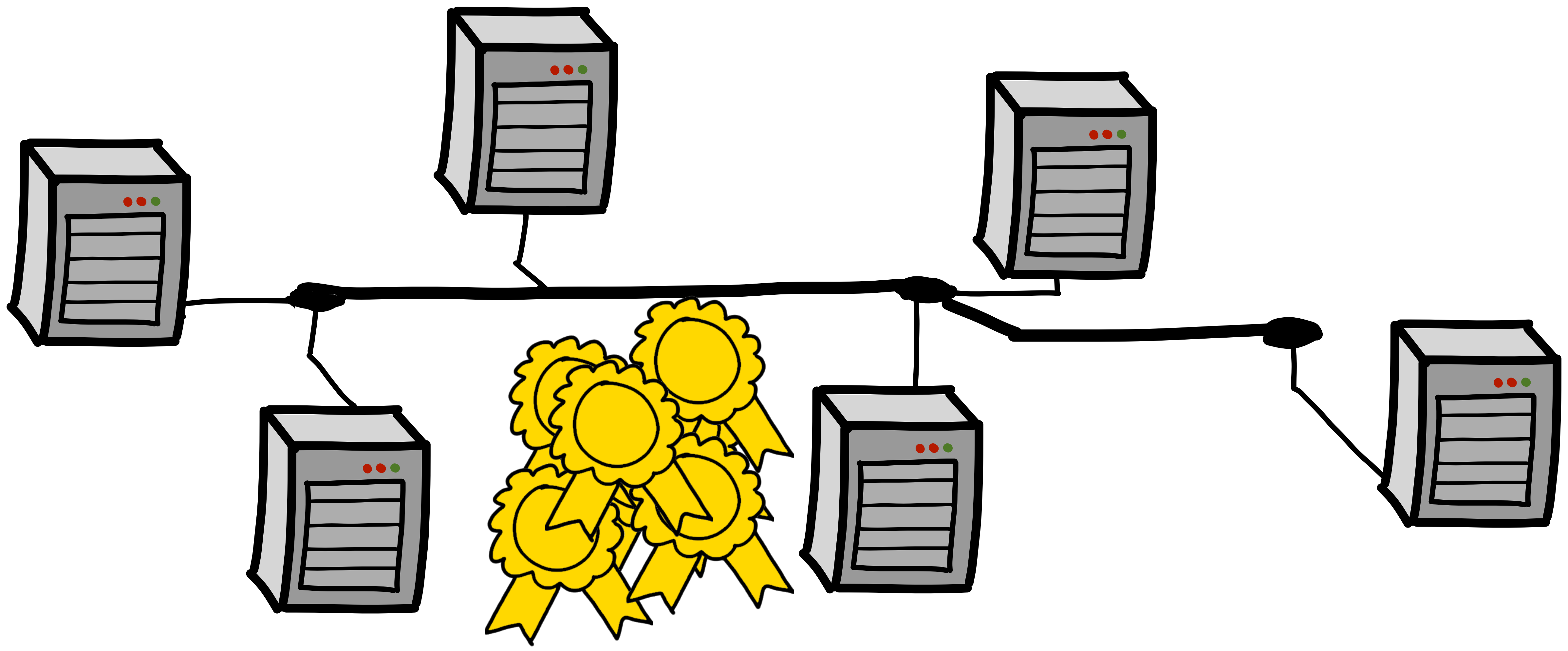


Is communication the bottleneck?



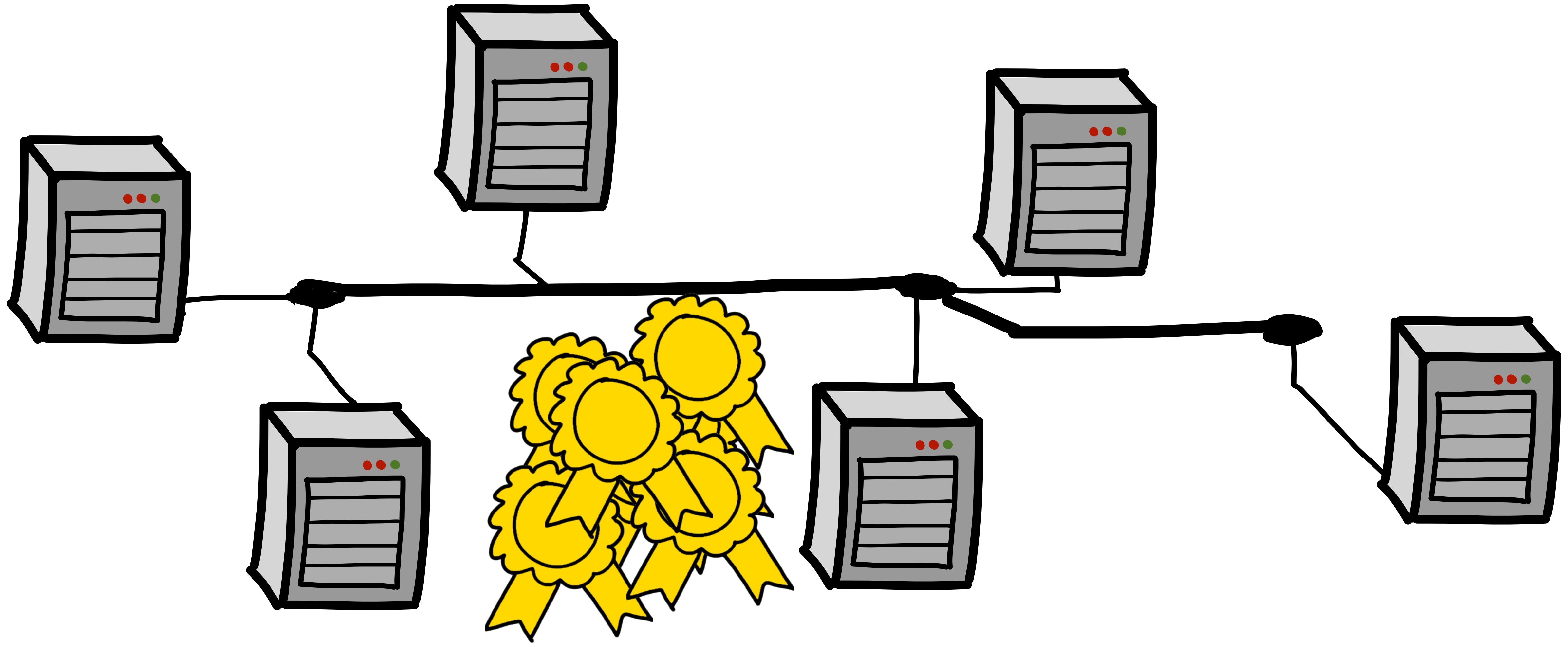
- **Large-scale automated distributed signing:**

Is communication the bottleneck?



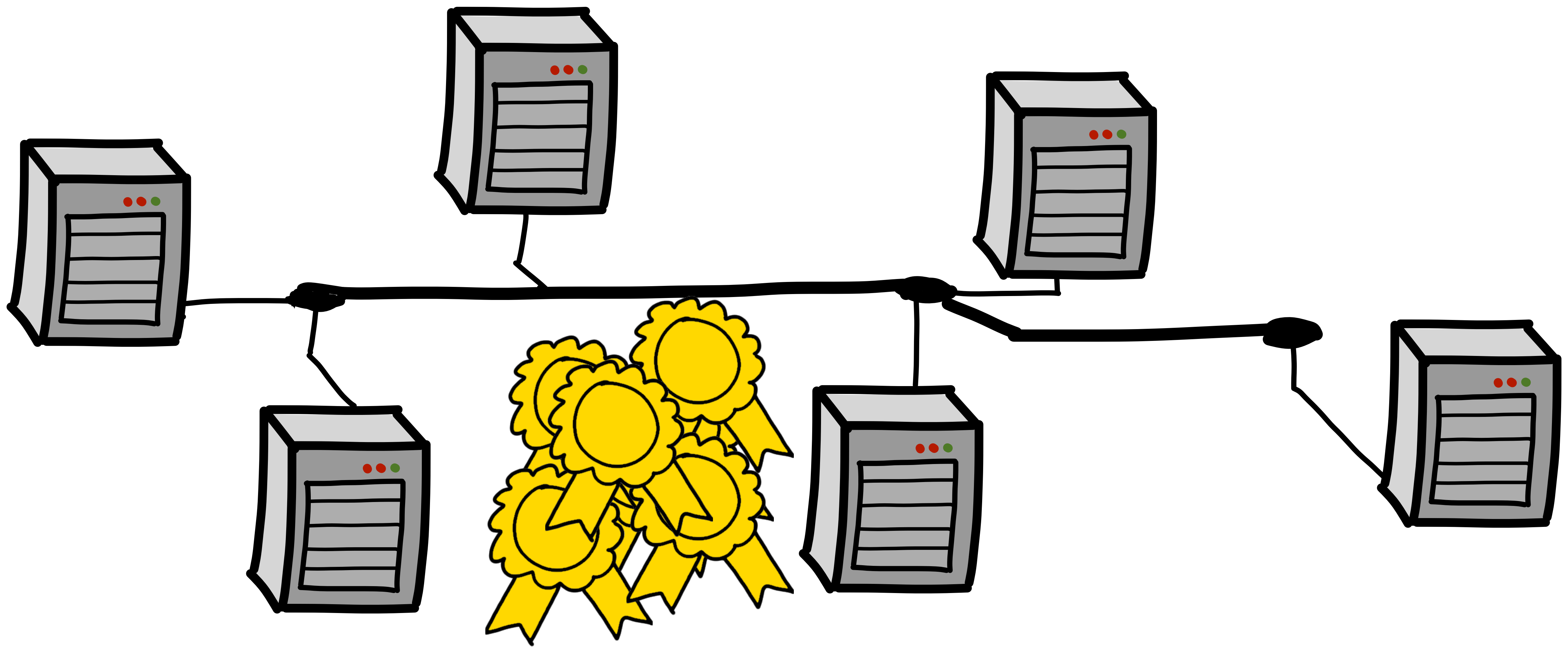
- **Large-scale automated distributed signing:**

Is communication the bottleneck?



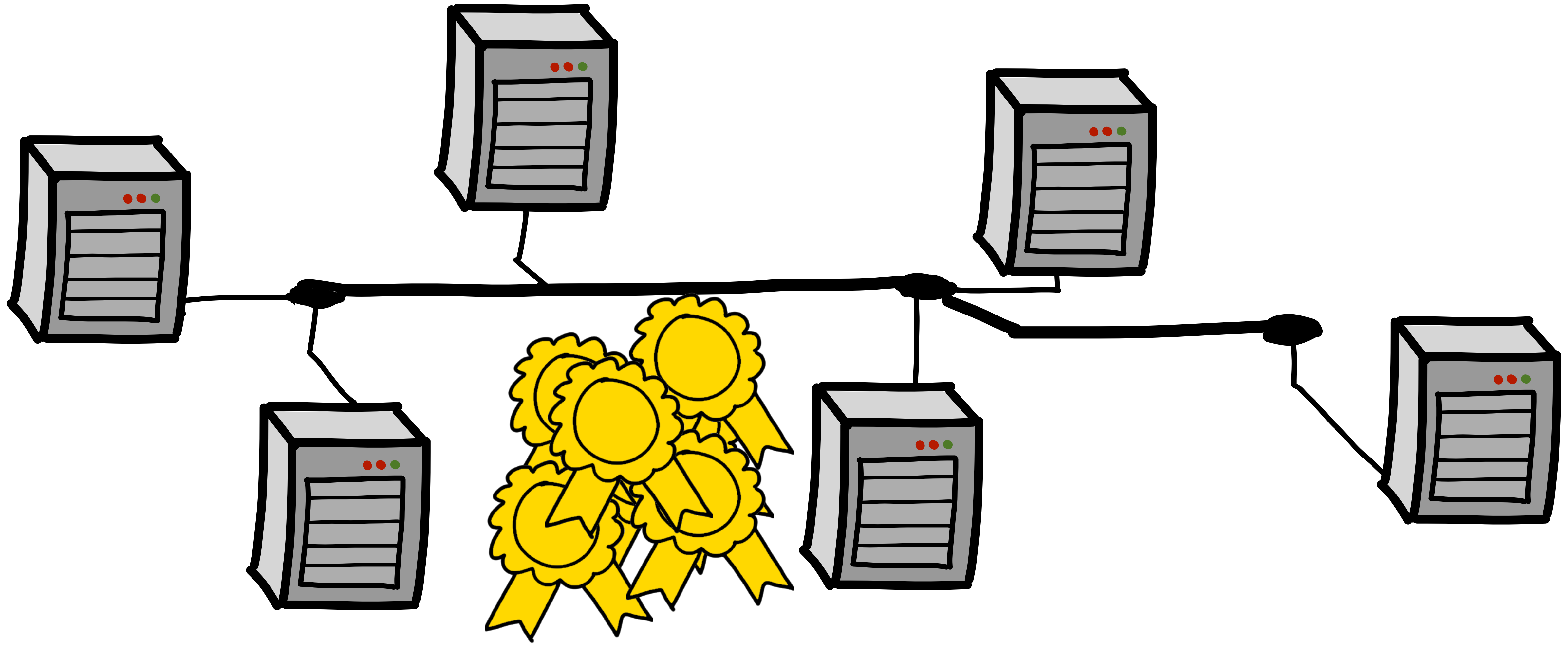
- **Large-scale automated distributed signing:**
  - Threshold 2: 3.8ms/sig <= ~263 sig/second

Is communication the bottleneck?



- **Large-scale automated distributed signing:**
  - Threshold 2: 3.8ms/sig <= ~263 sig/second
  - Threshold 20: 31.6ms/sig <= ~31 sig/second

Is communication the bottleneck?



- **Large-scale automated distributed signing:**
  - Threshold 2: 3.8ms/sig <= ~263 sig/second
  - Threshold 20: 31.6ms/sig <= ~31 sig/second
- Neither setting saturates a gigabit connection

Example 2: Datacenter Signing

How much bandwidth to be CPU bound?
(including preprocessing)

2 Parties
~250 sigs/second

256 Parties
~3 sigs/second

using GCP n1-highcpu nodes

Example 2: Datacenter Signing

How much bandwidth to be CPU bound?
(including preprocessing)

2 Parties
~250 sigs/second

Each party sends:
~700 Kbits per sig

256 Parties
~3 sigs/second

Each party sends:
~185 Mbits per sig

using GCP n1-highcpu nodes

Example 2: Datacenter Signing

How much bandwidth to be CPU bound?
(including preprocessing)

2 Parties
~250 sigs/second

Each party sends:
~700 Kbits per sig

Bandwidth required:
~180 Mbps symmetric

256 Parties
~3 sigs/second

Each party sends:
~185 Mbits per sig

Bandwidth required:
~555 Mbps symmetric

using GCP n1-highcpu nodes

Intro		How to distribute ECDSA			Tradeoffs
MP-Schnorr is easy	but not ECDSA	Evolution of Techniques	ECDSA Tuples	Our protocol: Simple consistency check	OT vs AHE

In Conclusion

- Threshold ECDSA in Three Rounds: Now matches Schnorr
- Enabled by well-chosen correlation + simple new consistency check
- Blackbox use of UC 2-round 2P-MUL
  NOTE: OT-based protocols satisfy UC, but AHE is more complicated
- No (explicit) ZK proofs during signing or DKG
  ⇒ light protocol and straightforward UC analysis

dkls.info

Thanks!

Thanks **Eysa Lee** for