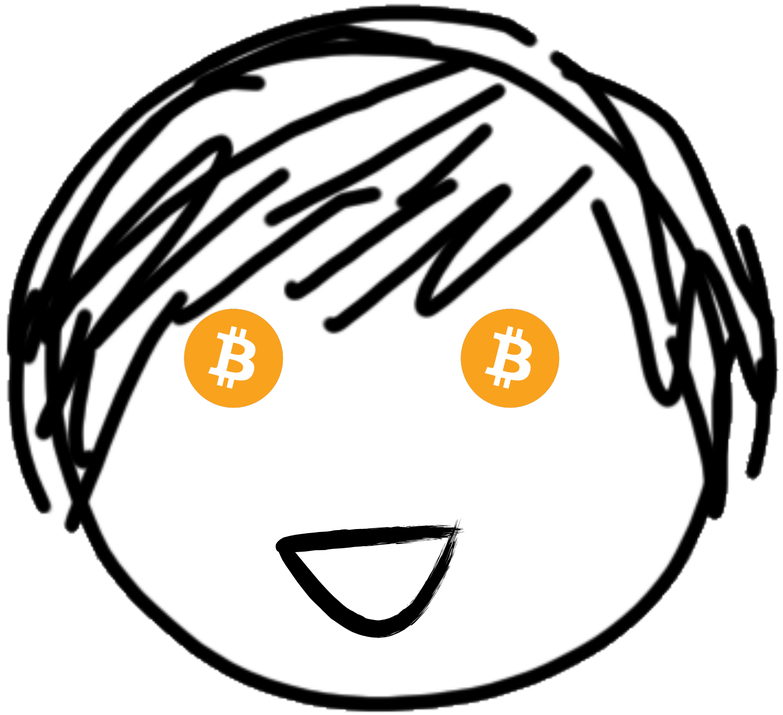
Threshold Schnorr with Stateless Deterministic Signing from Standard Assumptions

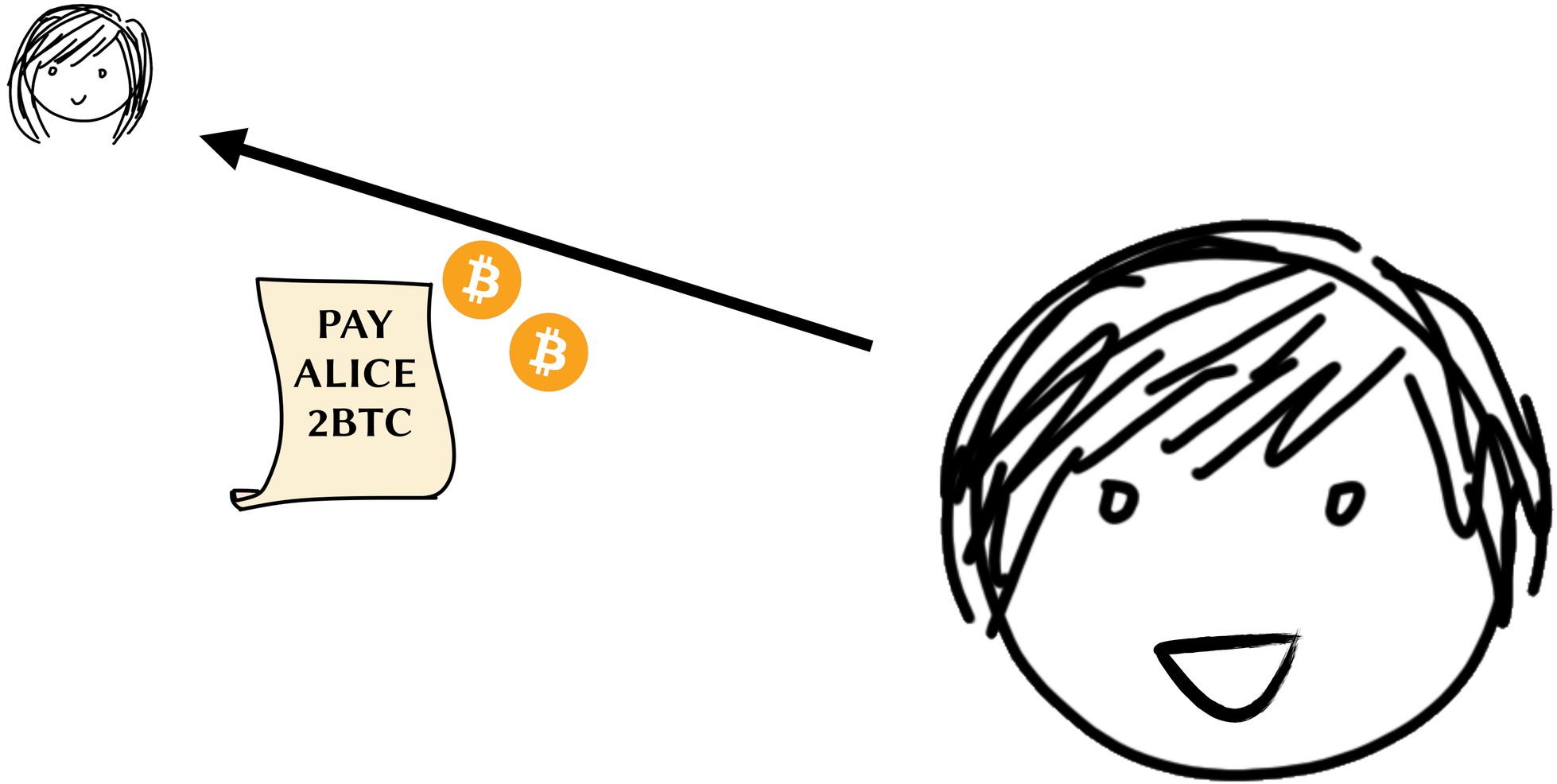
François Garillot — FACEBOOK / novi

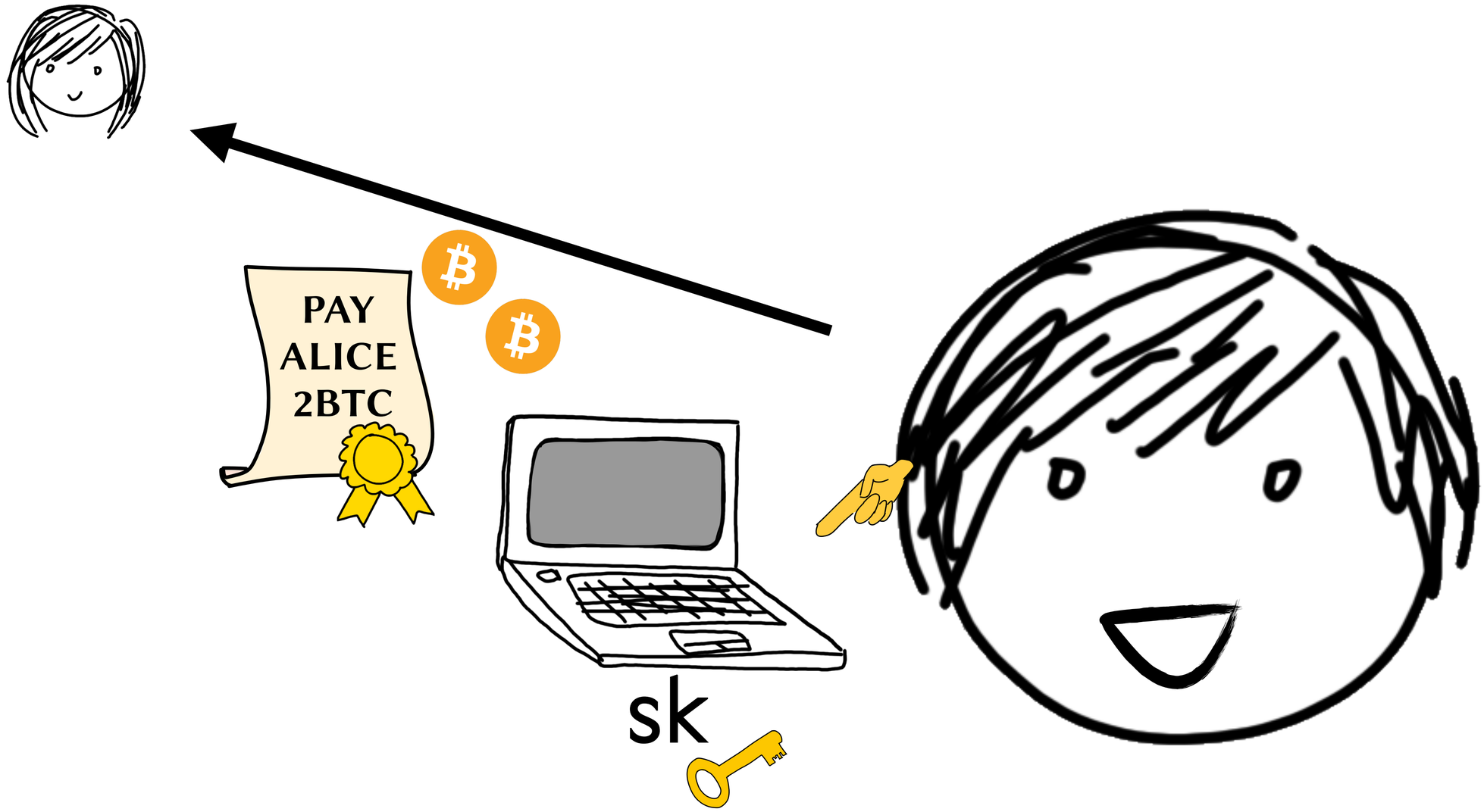
Yashvanth Kondi — Northeastern University

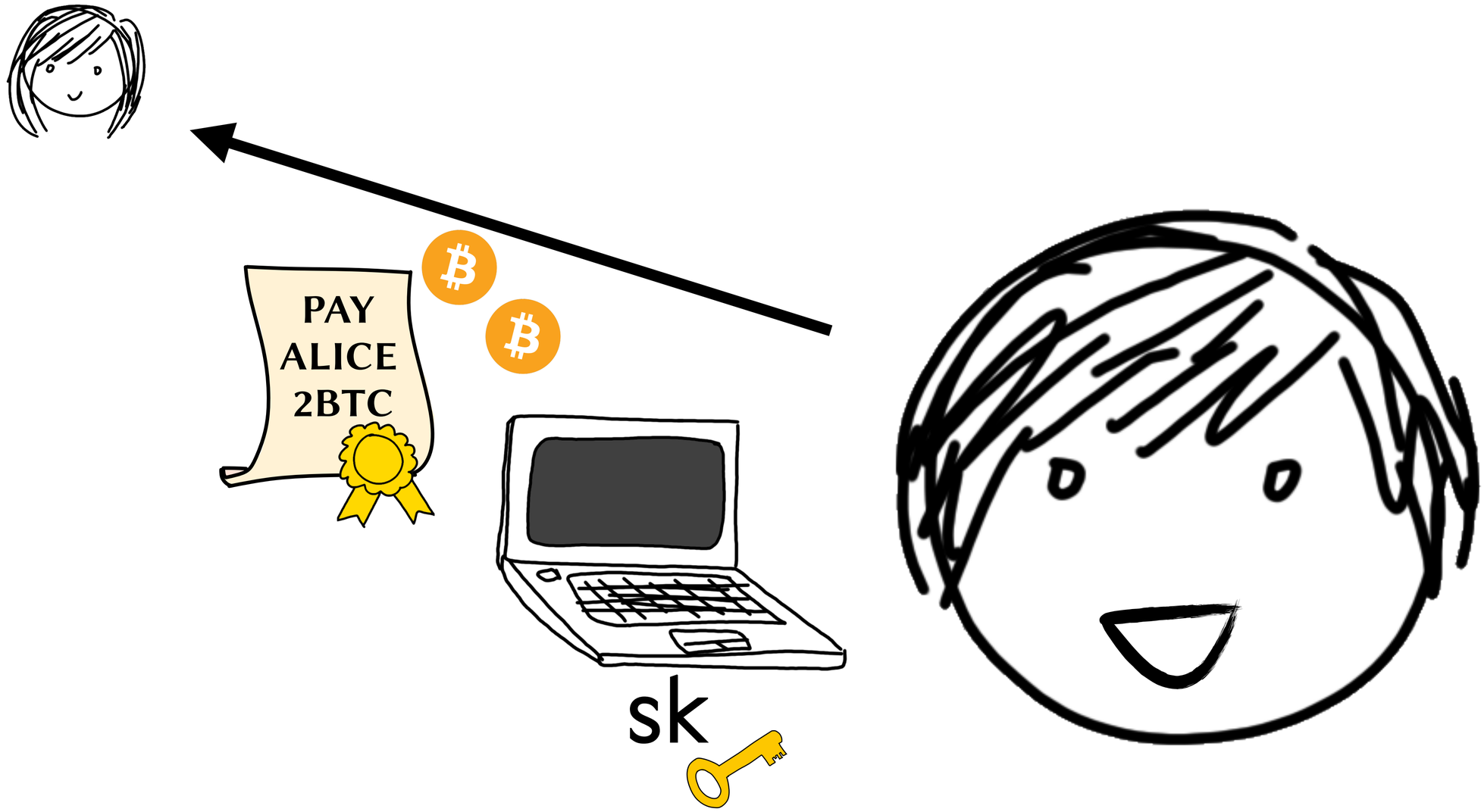
Payman Mohassel — FACEBOOK

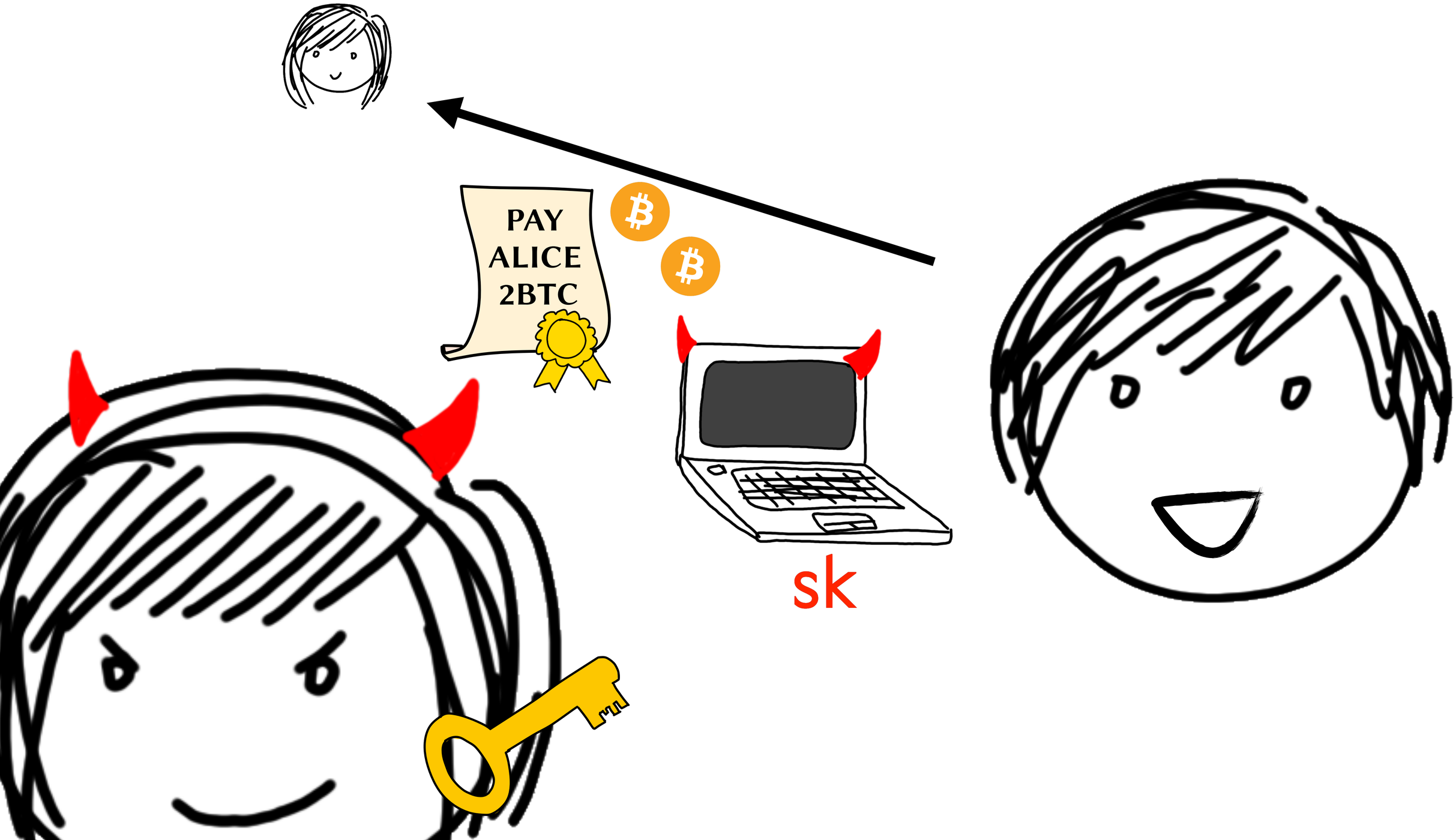
Valeria Nikolaenko — FACEBOOK / novi

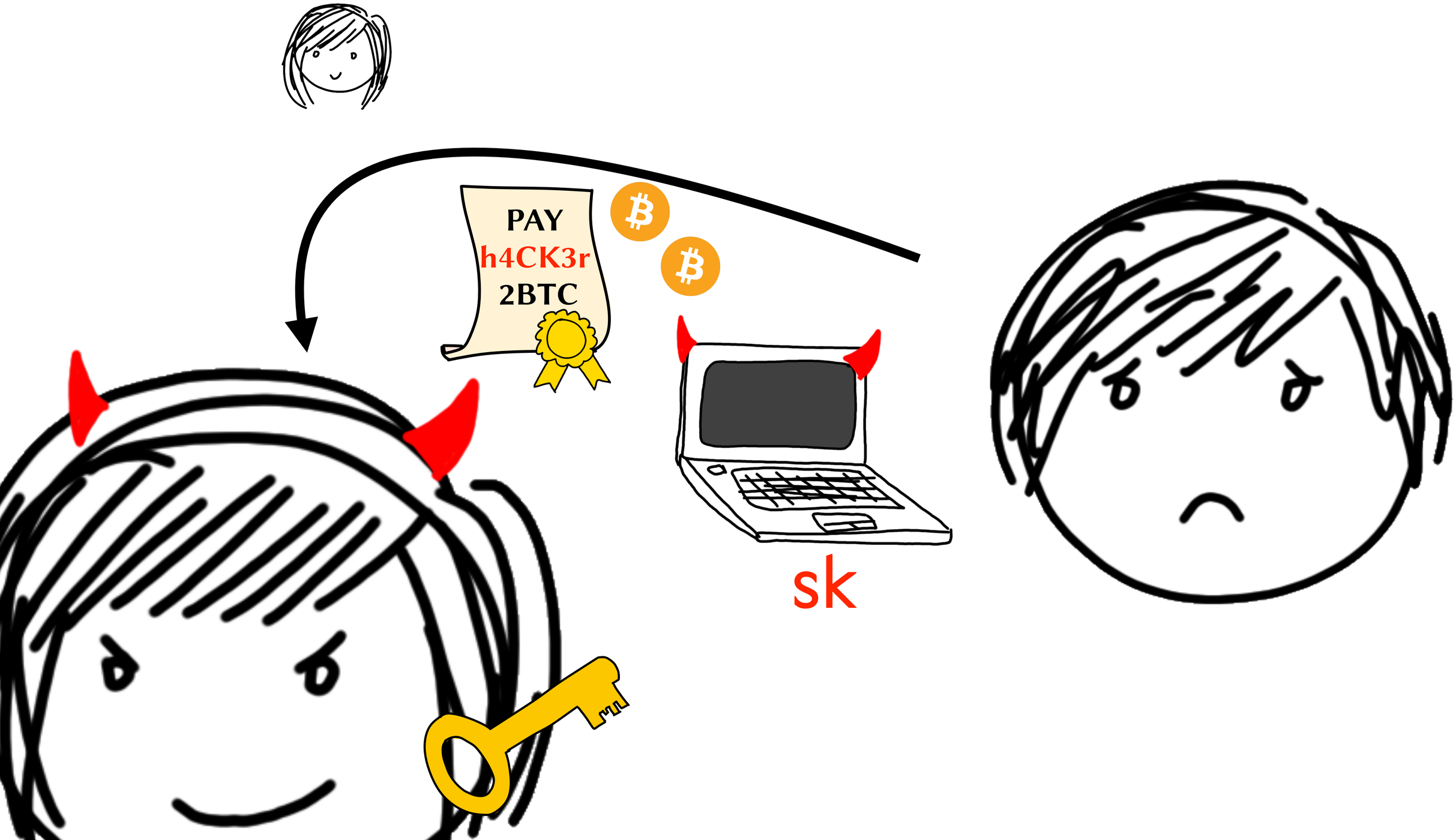


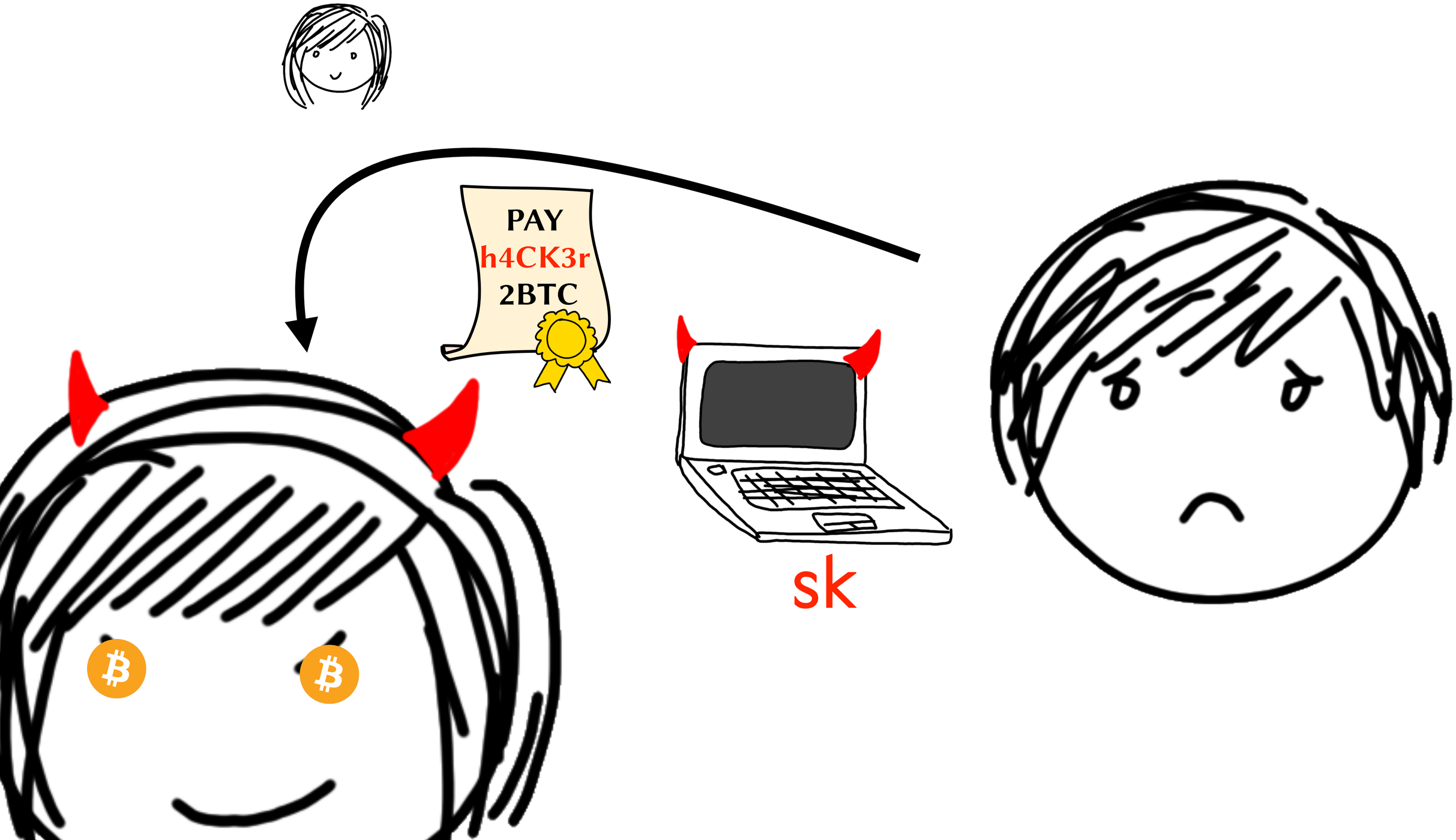

PAY
ALICE
2BTC

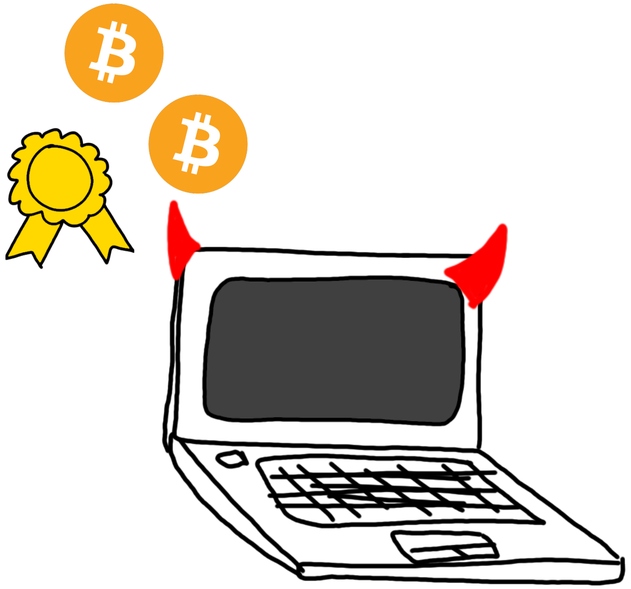

PAY
ALICE
2BTC
sk

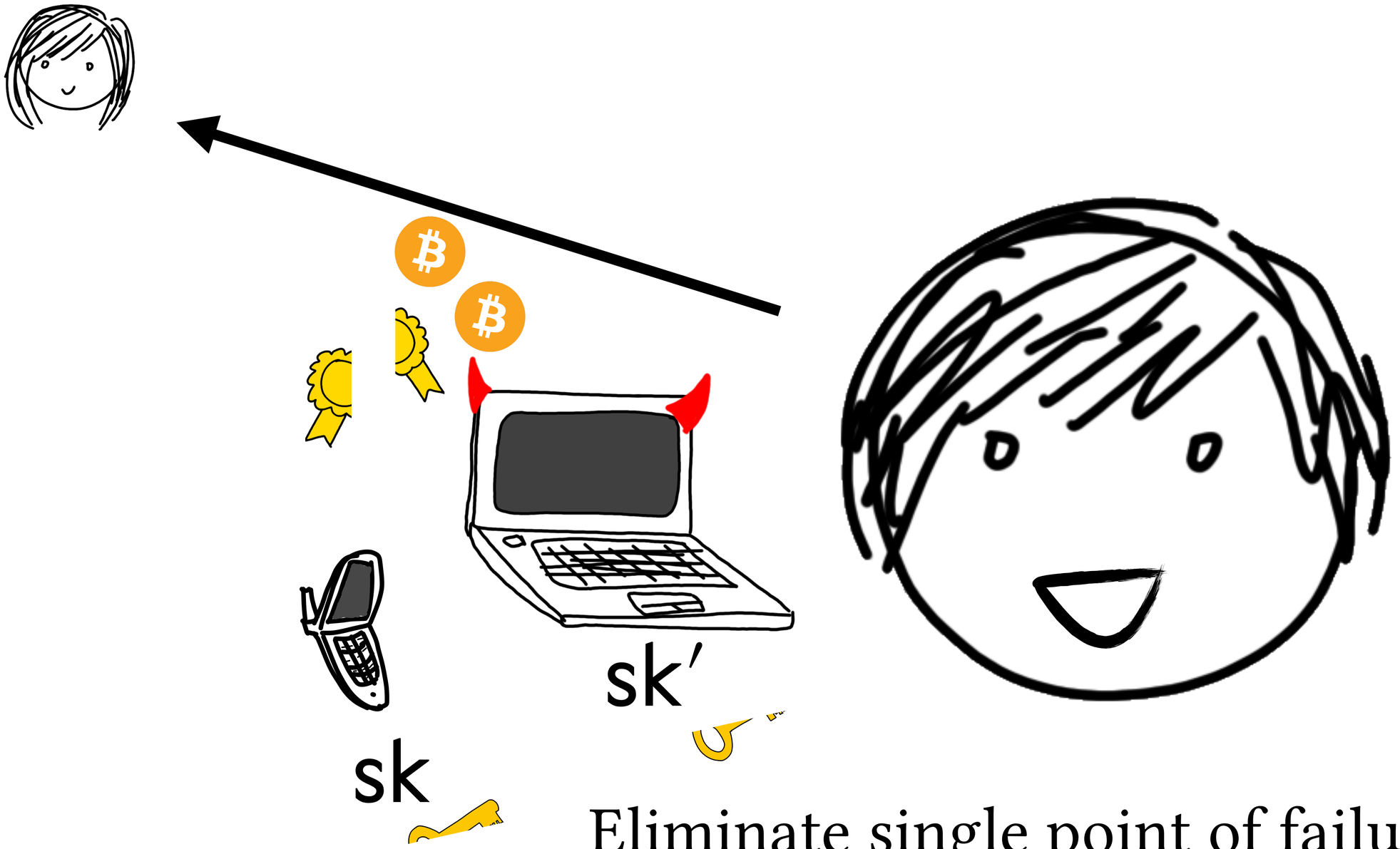

PAY
ALICE
2BTC
sk


PAY
ALICE
2BTC
sk


PAY
h4CK3r
2BTC
sk


PAY
h4CK3r
2BTC
sk

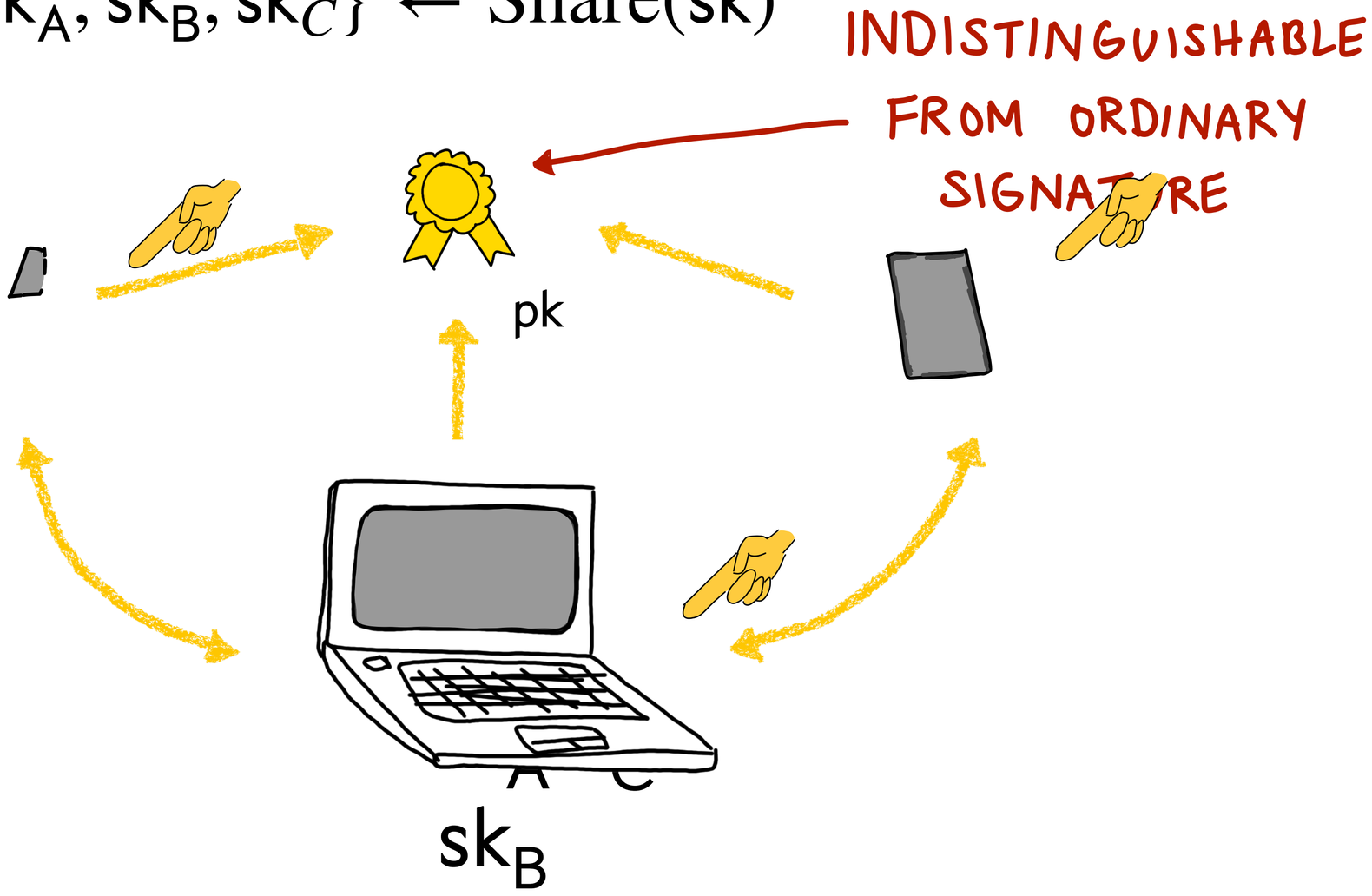




Eliminate single point of failure

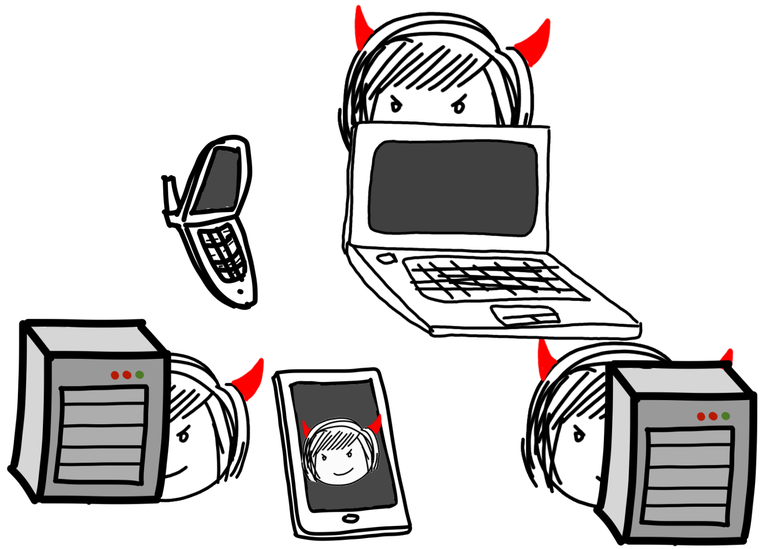
Threshold Signature

$$\{sk_A, sk_B, sk_C\} \leftarrow \text{Share}(sk)$$



Adversary Model

- Corruption threshold



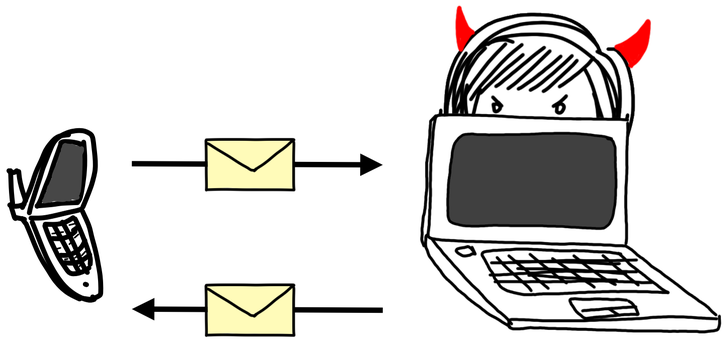
Dishonest majority
(only one device uncompromised)

Adversary Model

- Corruption threshold

Dishonest majority
(only one device uncompromised)

- Adversarial behaviour

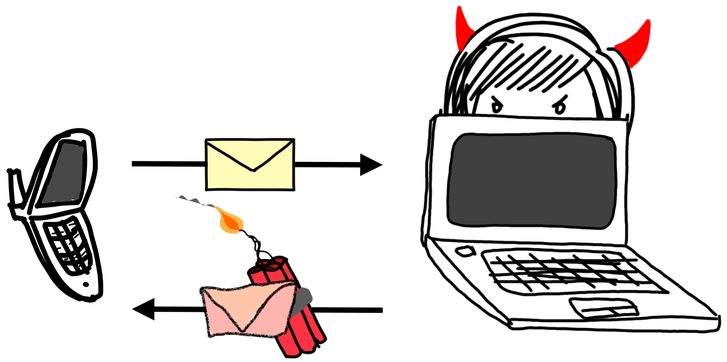


Adversary Model

- Corruption threshold

Dishonest majority
(only one device uncompromised)

- Adversarial behaviour



Malicious
(arbitrary deviations from protocol)

Schnorr's Signature Scheme

- Elegant signature scheme with security based on the hardness of computing discrete logarithms in carefully chosen group [Schnorr 89]
- Patent initially hampered widespread adoption
- Now seeing increased deployment across the internet in the form of EdDSA [Bernstein Duif Lange Schwabe Yang 11]
- Very easy to distribute with natural threshold key generation and signing protocols [Stinson Strobl 01][Gennaro Jarecki Krawczyk Rabin 03]

Schnorr Key Generation

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

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

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

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

secret **k**ey: kept private

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

Schnorr Signing

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

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

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

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

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

Schnorr Signing

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

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

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

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

SchnorrSign(sk, m) :

$k \leftarrow \mathbb{Z}_q$

Schnorr Signing

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

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

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

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

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

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

NONCE
One-time use value

Schnorr Signing

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

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

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

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

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

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

$e = H(R \| m)$

NONCE
One-time use value

Schnorr Signing

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

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

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

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

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

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

NONCE
One-time use value

$e = H(R \| m)$

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

Schnorr Signing

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

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

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

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

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

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

NONCE
One-time use value

$e = H(R \| m)$

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

$\sigma = (s, R)$

output σ

Schnorr Signing

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

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

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

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

SchnorrSign(sk, m) :

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

NONCE
One-time use value

$e = H(R \| m)$

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

$\sigma = (s, R)$

output σ

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

Schnorr Signing

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

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

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

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

SchnorrSign(sk, m) :

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

NONCE
One-time use value

$e = H(R \| m)$

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

$\sigma = (s, R)$

output σ

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

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

Schnorr Signing

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

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

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

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

SchnorrSign(sk, m) :

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

NONCE
One-time use value

$e = H(R \| m)$

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

$\sigma = (s, R)$

output σ

Verifying a signature:

$s \cdot G \stackrel{?}{=} R - e \cdot \mathsf{PK}$

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

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

Schnorr Signing

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

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

$$R = k \cdot G$$

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

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

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

$$\text{output } \sigma$$

Schnorr Signing

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

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

$e = H(R \| m)$

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

$\sigma = (s, R)$

output σ

Linear function of k, sk
Linear operations are very easy to distribute with most natural secret sharing schemes

Distributing Schnorr Signing

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

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

Distributing Schnorr Signing

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

sk_A

sk_B

$k_A + k_B = k$

Distributing Schnorr Signing

sk_A

sk_B

SchnorrSign(sk, m) :

$k \leftarrow \mathbb{Z}_q$	$k_A \leftarrow \mathbb{Z}_q$	$k_B \leftarrow \mathbb{Z}_q$
$R = k \cdot G$	$R_A = k_A \cdot G$	$R_B = k_B \cdot G$

$e = H(R \| m)$

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

$\sigma = (s, R)$

output σ

Distributing Schnorr Signing

sk_A

sk_B

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

$k_A \leftarrow \mathbb{Z}_q$

$R_A = k_A \cdot G$

$k_B \leftarrow \mathbb{Z}_q$

$R_B = k_B \cdot G$

R_B

R_A

Distributing Schnorr Signing

sk_A

sk_B

SchnorrSign(sk, m) :

$k \leftarrow \mathbb{Z}_q$	$k_A \leftarrow \mathbb{Z}_q$	$k_B \leftarrow \mathbb{Z}_q$
$R = k \cdot G$	$R = R_A + R_B$	$R = R_A + R_B$
$e = H(R \Vert m)$		
$s = k - \mathsf{sk} \cdot e$		
$\sigma = (s, R)$		
output σ		

Distributing Schnorr Signing

sk_A

sk_B

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

$k_A \leftarrow \mathbb{Z}_q$

$R = R_A + R_B$

$e = H(R \| m)$

$k_B \leftarrow \mathbb{Z}_q$

$R = R_A + R_B$

$e = H(R \| m)$

Distributing Schnorr Signing

sk_A

sk_B

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

$k_A \leftarrow \mathbb{Z}_q$

$R = R_A + R_B$

$e = H(R \| m)$

$s_A = k_A - \mathsf{sk}_A \cdot e$

$k_B \leftarrow \mathbb{Z}_q$

$R = R_A + R_B$

$e = H(R \| m)$

$s_B = k_B - \mathsf{sk}_B \cdot e$

Distributing Schnorr Signing

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

sk_A

sk_B

$k_A \leftarrow \mathbb{Z}_q$

$R = R_A + R_B$

$e = H(R \| m)$

$s_A = k_A - \mathsf{sk}_A \cdot e$

$k_B \leftarrow \mathbb{Z}_q$

$R = R_A + R_B$

$e = H(R \| m)$

$s_B = k_B - \mathsf{sk}_B \cdot e$

s_B

s_A

Distributing Schnorr Signing

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

sk_A

sk_B

$k_A \leftarrow \mathbb{Z}_q$	$k_B \leftarrow \mathbb{Z}_q$
$R = R_A + R_B$ ⟷	⟷ $R = R_A + R_B$
$e = H(R \| m)$	$e = H(R \| m)$
$s_A = k_A - \mathsf{sk}_A \cdot e$	$s_B = k_B - \mathsf{sk}_B \cdot e$
$s = s_A + s_B$ ⟷	⟷ $s = s_A + s_B$

Distributing Schnorr Signing

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

sk_A

sk_B

$k_A \leftarrow \mathbb{Z}_q$	$k_B \leftarrow \mathbb{Z}_q$
$R = R_A + R_B$ ⟷	⟷ $R = R_A + R_B$
$e = H(R \| m)$	$e = H(R \| m)$
$s_A = k_A - \mathsf{sk}_A \cdot e$	$s_B = k_B - \mathsf{sk}_B \cdot e$
$s = s_A + s_B$ ⟷	⟷ $s = s_A + s_B$

Generalizes to n parties easily

Practical Issue

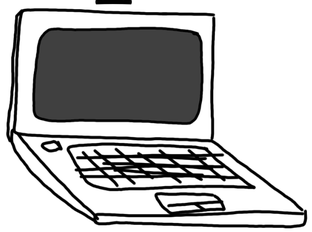
- Each Schnorr signature requires a fresh, one-time nonce (k, R)
- Security is extremely sensitive to the distribution of k — even a tiny amount of non-uniformity across signatures can be leveraged to retrieve sk in its entirety [Boneh Venkatesan 96][Howgrave-Graham Smart 01][Bleichenbacher 00] [Aranha Novaes Takahashi Tibouchi Yarom 20][Albrecht Heninger 21]
- Major concern in practice: "true" randomness is a scarce resource
  - Errors in implementation
  - Poorly seeded Random Number Generators

Cryptographic Solution

- This systems-level problem can be avoided with a simple cryptographic trick
  - During (one-time) secret key generation: sample a seed $\mathsf{sd} \leftarrow \{0,1\}^\kappa$
  - When signing m: instead of sampling fresh k, compute $k = F(\mathsf{sd}, m)$
  - Assuming $F : \{0,1\}^\kappa \to \mathbb{Z}_q$ is a pseudorandom function, the distribution of $F(\mathsf{sd}, m)$ is negligibly different from that of uniform choice of k per message
- This is a classic idea [Barwood 97][M'Raïhi Naccache Pointcheval Vaudenay 98] [Wigley 97] that is employed by the modern **EdDSA** variant of Schnorr

A Simple Attempt

sk_A



sk_B

$k_A \leftarrow \mathbb{Z}_q$

$k_B \leftarrow \mathbb{Z}_q$

$R_A = k_A \cdot G$

$R_B = k_B \cdot G$

$R = R_A + R_B$ ⟷ $R = R_A + R_B$

$e = H(R \| m)$

$e = H(R \| m)$

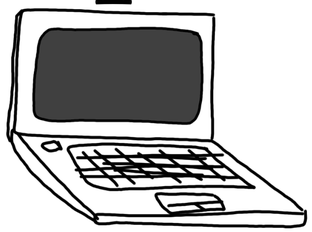
$s_A = k_A - \mathsf{sk}_A \cdot e$

$s_B = k_B - \mathsf{sk}_B \cdot e$

$s = s_A + s_B$ ⟷ $s = s_A + s_B$

A Simple Attempt

sk_A sd_A sd_B sk_B



$k_A \leftarrow \mathbb{Z}_q$

$R_A = k_A \cdot G$

$R = R_A + R_B$ ⟷ $R = R_A + R_B$

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

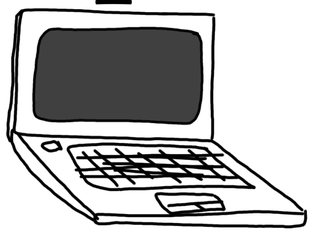
$s_A = k_A - \mathsf{sk}_A \cdot e$ $\quad$ $s_B = k_B - \mathsf{sk}_B \cdot e$

$s = s_A + s_B$ ⟷ $s = s_A + s_B$

A Simple Attempt

sk_A sd_A

sd_B sk_B



has collected

$k_A = F(\mathsf{sd}_A, m)$ | $k_B = F(\mathsf{sd}_B, m)$

$R_A = k_A \cdot G$ | $R_B = k_B \cdot G$

$R = R_A + R_B$ ⟷ $R = R_A + R_B$

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

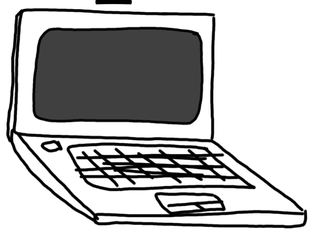
$s_A = k_A - \mathsf{sk}_A \cdot e$ | $s_B = k_B - \mathsf{sk}_B \cdot e$

$s = s_A + s_B$ ⟷ $s = s_A + s_B$

A Simple Attempt

sk_A sd_A

sd_B sk_B



has collected

$s_B = k_B - \mathsf{sk}_B \cdot e$

$k_A = F(\mathsf{sd}_A, m)$ | $k_B = F(\mathsf{sd}_B, m)$

$R_A = k_A \cdot G$ | $R_B = k_B \cdot G$

$R = R_A + R_B$ ⟷ $R = R_A + R_B$

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

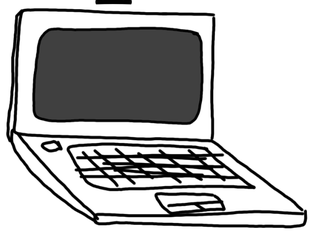
$s_A = k_A - \mathsf{sk}_A \cdot e$ | $s_B = k_B - \mathsf{sk}_B \cdot e$

$s = s_A + s_B$ ⟷ $s = s_A + s_B$

A Simple Attempt

sk_A sd_A

sd_B sk_B



has collected

$s_B = k_B - \mathsf{sk}_B \cdot e$

$k_A^* = F(\mathsf{sd}_A, m)+1$

$R_A = k_A \cdot G$

$R = R_A + R_B$ ⟷ $R = R_A + R_B$

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

$s_A = k_A - \mathsf{sk}_A \cdot e$

$k_B = F(\mathsf{sd}_B, m)$

$R_B = k_B \cdot G$

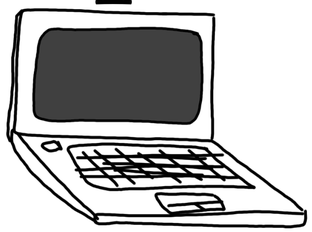
$s_B = k_B - \mathsf{sk}_B \cdot e$

$s = s_A + s_B$ ⟷ $s = s_A + s_B$

A Simple Attempt

sk_A sd_A

sd_B sk_B



has collected

$s_B = k_B - \mathsf{sk}_B \cdot e$

$k_A^* = F(\mathsf{sd}_A, m) + 1$

$R_A^* = k_A^* \cdot G$

$R = R_A + R_B$ ⟷ $R = R_A + R_B$

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

$s_A = k_A - \mathsf{sk}_A \cdot e$

$k_B = F(\mathsf{sd}_B, m)$

$R_B = k_B \cdot G$

$s_B = k_B - \mathsf{sk}_B \cdot e$

$s = s_A + s_B$ ⟷ $s = s_A + s_B$

A Simple Attempt

sk_A sd_A

sd_B sk_B

has collected

$s_B = k_B - \mathsf{sk}_B \cdot e$

$k_A^* = F(\mathsf{sd}_A, m) + 1$

$R_A^* = k_A^* \cdot G$

$k_B = F(\mathsf{sd}_B, m)$

$R_B = k_B \cdot G$

$R^* = R_A^* + R_B$ ⟷ $R^* = R_A^* + R_B$

$e = H(R \| m)$

$e = H(R \| m)$

$s_A = k_A - \mathsf{sk}_A \cdot e$

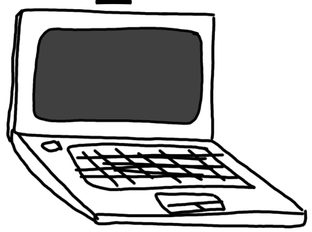
$s_B = k_B - \mathsf{sk}_B \cdot e$

$s = s_A + s_B$ ⟷ $s = s_A + s_B$

A Simple Attempt

$\mathsf{sk}_A \quad \mathsf{sd}_A$

$\mathsf{sd}_B \quad \mathsf{sk}_B$



has collected

$s_B = k_B - \mathsf{sk}_B \cdot e$

$k_A^* = F(\mathsf{sd}_A, m) + 1$

$R_A^* = k_A^* \cdot G$

$k_B = F(\mathsf{sd}_B, m)$

$R_B = k_B \cdot G$

$R^* = R_A^* + R_B$ ⟷ $R^* = R_A^* + R_B$

$e^* = H(R^* \| m)$ $\qquad$ $e^* = H(R^* \| m)$

$s_A = k_A - \mathsf{sk}_A \cdot e$

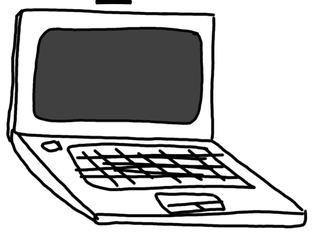
$s_B = k_B - \mathsf{sk}_B \cdot e$

$s = s_A + s_B$ ⟷ $s = s_A + s_B$

A Simple Attempt

$\mathsf{sk}_A \quad \mathsf{sd}_A$

$\mathsf{sd}_B \quad \mathsf{sk}_B$



has collected

$s_B = k_B - \mathsf{sk}_B \cdot e$

$k_A^* = F(\mathsf{sd}_A, m) + 1$

$R_A^* = k_A^* \cdot G$

$R^* = R_A^* + R_B$

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

$s_A^* = k_A^* - \mathsf{sk}_A \cdot e^*$

$s = s_A + s_B$

$k_B = F(\mathsf{sd}_B, m)$

$R_B = k_B \cdot G$

$R^* = R_A^* + R_B$

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

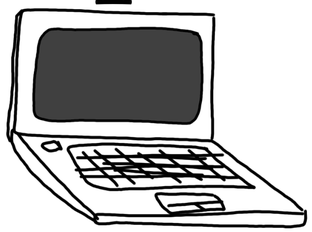
$s_B^* = k_B - \mathsf{sk}_B \cdot e^*$

$s = s_A + s_B$

A Simple Attempt

sk_A sd_A

sd_B sk_B



has collected

$s_B = k_B - \mathsf{sk}_B \cdot e$

$s_B^* = k_B - \mathsf{sk}_B \cdot e^*$

$k_A^* = F(\mathsf{sd}_A, m) + 1$

$R_A^* = k_A^* \cdot G$

$R^* = R_A^* + R_B$

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

$k_B = F(\mathsf{sd}_B, m)$

$R_B = k_B \cdot G$

$R^* = R_A^* + R_B$

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

$s_A^* = k_A^* - \mathsf{sk}_A \cdot e^*$

$s_B^* = k_B - \mathsf{sk}_B \cdot e^*$

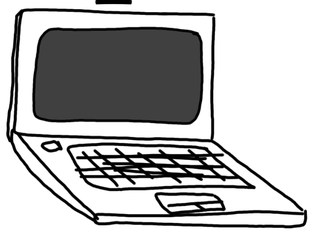
$s = s_A + s_B$

$s = s_A + s_B$

A Simple Attempt

sk_A sd_A

sd_B sk_B



$k_A^* = F(\mathsf{sd}_A, m)+1$

$R_A^* = k_A^* \cdot G$

$R^* = R_A^* + R_B$ ⟷ $R^* = R_A^* + R_B$

$e^* = H(R^* \| m)$ $\quad$ $e^* = H(R^* \| m)$

$s_A^* = k_A^* - \mathsf{sk}_A \cdot e^*$ $\quad$ $s_B^* = k_B - \mathsf{sk}_B \cdot e^*$

$s = s_A + s_B$ ⟷ $s = s_A + s_B$

has collected

$s_B = k_B - \mathsf{sk}_B \cdot e$

$s_B^* = k_B - \mathsf{sk}_B \cdot e^*$

Two linear equations in two unknowns

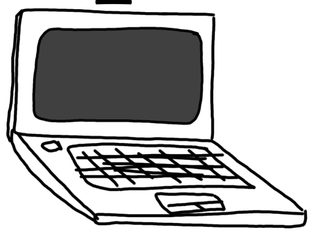
Simply solve for sk_B and reconstruct sk

[Maxwell Poelstra Seurin Wuille 19]

A Simple Attempt

sk_A sd_A

sd_B sk_B



has collected

$s_B = k_B - \mathsf{sk}_B \cdot e$

$s_B^* = k_B - \mathsf{sk}_B \cdot e^*$

Two linear equations in two unknowns

Simply solve for sk_B and reconstruct sk

[Maxwell Poelstra Seurin Wuille 19]

$k_A^* = F(\mathsf{sd}_A, m) + 1$

$R_A^* = k_A^* \cdot G$

$R^* = R_A^* + R_B$ ⟷ $R^* = R_A^* + R_B$

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

$s_A^* = k_A^* - \mathsf{sk}_A \cdot e^*$

$s = s_A + s_B$ ⟷ $s = s_A + s_B$

$k_B = F(\mathsf{sd}_B, m)$

$R_B = k_B \cdot G$

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

$s_B^* = k_B - \mathsf{sk}_B \cdot e^*$

General flavour of problem previously encountered in Resettable ZK/MPC

[Canetti Goldreich Goldwasser Micali 00]

Maintain a Counter?

- **Simple proposal**: each device maintains a counter (CTR), and derives for eg. $k_A = F(sd_A, CTR)$. Each time CTR is accessed it is incremented, k_A is always fresh

- However this introduces a new attack surface: CTR reuse would be catastrophic

- Undetectable reuse of stale state is a significant concern in practice, due to:
  - Power supply interruptions
  - Restoring from backups
  - Virtual Machines loaded with old snapshots

Can be adversarially induced, or even due to careless mistakes

Isn’t this a Systems problem?

- Maintaining persistent state (‘state continuity’) is difficult even on dedicated hardened and isolated devices [Parno Lorch Douceur Mickens McCune 11]

- Broadly two flavours of general systems solutions:
  1. **Use ‘helper’ nodes**. Inapplicable to our dishonest majority setting.
  2. **Special Purpose Hardware** [PLDMM 11][Strackx Piessens 16]
     - Qualitatively, this induces extra physical assumptions
     - Quantitatively, trusted hardware is slow (60-100ms to increment an Intel SGX Trusted Monotonic Counter), expensive, and has a limited lifespan as non-volatile memory can wear out in a few days of continuous use [Matetic Ahmed Kostiainen Dhar Sommer Gervais Juels Capkun 17]

Can it be a Crypto problem?

- In this work we study whether solving this as a cryptographic protocol design problem can be of benefit:

How can we design a threshold Schnorr protocol that enjoys stateless deterministic signing?

…no device need sample fresh randomness, **or rely on updating state** after (Distributed) KeyGeneration

Can safely restore crashed devices with long-term secrets

- We develop new techniques to construct Schnorr threshold signing that is **stateless and deterministic by design** while relying on **native cryptographic tools** that we estimate to be significantly **faster than trusted hardware**

Intuition

$\mathsf{sk}_A \quad \mathsf{sd}_A$

$\mathsf{sd}_B \quad \mathsf{sk}_B$

$k_A = F(\mathsf{sd}_A, m)$

$R_A = k_A \cdot G$

$R = R_A + R_B$ ⟷ $R = R_A + R_B$

$e = H(R \| m)$

$s_A = k_A - \mathsf{sk}_A \cdot e$

$k_B = F(\mathsf{sd}_B, m)$

$R_B = k_B \cdot G$

$e = H(R \| m)$

$s_B = k_B - \mathsf{sk}_B \cdot e$

$s = s_A + s_B$ ⟷ $s = s_A + s_B$

Intuition

$\mathsf{sk}_A \quad \mathsf{sd}_A$

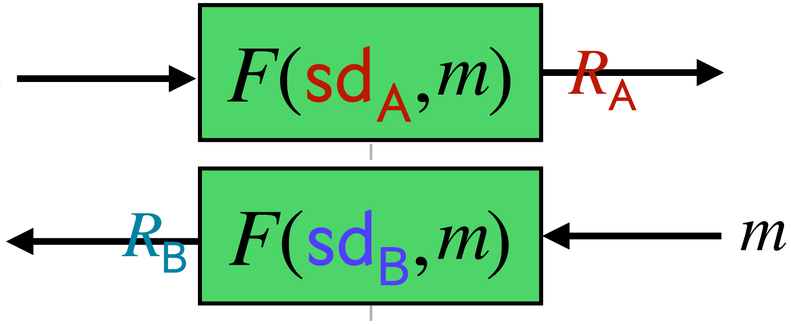
$\mathsf{sd}_B \quad \mathsf{sk}_B$

$k_A = F(\mathsf{sd}_A, m)$

$R_A = k_A \cdot G$

$k_B = F(\mathsf{sd}_B, m)$

$R_B = k_B \cdot G$



$R = R_A + R_B$

$e = H(R \| m)$

$s_A = k_A - \mathsf{sk}_A \cdot e$

$s = s_A + s_B$

$R = R_A + R_B$

$e = H(R \| m)$

$s_B = k_B - \mathsf{sk}_B \cdot e$

$s = s_A + s_B$

Instantiating $F(\mathsf{sd}, m)$

- **Canonical instantiation**:
  - Produce $\mathsf{Com} = \mathsf{Commit}(\mathsf{sd})$ during Distributed KeyGeneration
  - When signing m, prove in ZK that public parameters (Com, R, F, m) satisfy the relation $R = F(\mathsf{sd}, m) \cdot G$ where $\mathsf{Decommit}(\mathsf{Com}, \mathsf{sd}) = 1$

- There is a myriad of ways to construct such a proof system. We prioritise:
  - **Conservative assumptions**: F must be a standard PRF since security of the signature (which is exposed to the outside world) strongly depends on F
  - **Lightweight computation**: friendly to weaker devices (eg. mobile cryptocurrency wallet) as well as institutional high-throughput settings
  - **Round efficiency**: match regular threshold Schnorr (3 rounds)

Our Approach

- Variety of candidate cryptographic tools to instantiate $F(\mathsf{sd}_A, \cdot)$:
  - Succinct Non-interactive Arguments of Knowledge (SNARKs)
  - ~~Generic Multiparty Computation~~
  - MPC-in-the-head
  - Garbled Circuits

- Recall our constraints:
  - Standardised choice for F (eg. AES, SHA)
  - Lightweight computation
  - Match round efficiency of Threshold Schnorr

Our Approach

- Variety of candidate cryptographic tools to instantiate $F(\mathsf{sd}_A, \cdot)$:
  - ~~Succinct Non-interactive Arguments of Knowledge (SNARKs)~~
  - ~~Generic Multiparty Computation~~
  - MPC-in-the-head
  - Garbled Circuits

[Nick Ruffing Seurin Wuille 20]

Custom PRF+Bulletproofs very bandwidth efficient (~1KB), heavy to execute (~1 second)

- Recall our constraints:
  - Standardised choice for F (eg. AES, SHA)
  - Lightweight computation
  - Match round efficiency of Threshold Schnorr

Our Approach

- Variety of candidate cryptographic tools to instantiate $F(\mathsf{sd}_A, \cdot)$:
  - ~~Succinct Non-interactive Arguments of Knowledge (SNARKs)~~
  - ~~Generic Multiparty Computation~~
  - MPC-in-the-head
  - Garbled Circuits

Zero-knowledge for "composite statements"

[Nick Ruffing Seurin Wuille 20]

Custom PRF+Bulletproofs very bandwidth efficient (~1KB), heavy to execute (~1 second)

- Recall our constraints:
  - Standardised choice for F (eg. AES, SHA)
  - Lightweight computation
  - Match round efficiency of Threshold Schnorr

ZK for Composite Statements

- **Garbled Circuits** and **MPC-in-the-head**: lightweight proof systems that are traditionally efficient for Boolean Circuits, but not algebraic operations
- Framework of [Chase Ganesh Mohassel 16] for Garbled Circuits, extended to MPC-in-the-head by [Backes Hanzlik Herzberg Kate Pryvalov 19] bridges these respective techniques with algebraic operations with good concrete efficiency
- This is a great direction, since standardised tools (AES, SHA) have compact Boolean Circuit representations, and we require support for Elliptic Curve algebra
- Our target: only cheap symmetric key operations per proof, but existing techniques do not achieve this out of the box

This Work

- In existing works [CGM16, BHHKP19] applied to our setting the dominant cost (computation and bandwidth*) lies in the logistics of the Boolean-algebraic bridge, and in encoding the witness sd
- In particular they require $\Theta(\kappa)$ exponentiations, due to homomorphic commitments and Committed Oblivious Transfer (C-OT) per bit of the witness
- [CGM16] show how to replace the commitments by garbling $\tilde{O}(\kappa^2)$ extra gates, which concretely is more expensive than standard PRFs ($8 \times$ AES)
- We focus on the garbled circuit approach [CGM16] and develop new techniques so that the **dominant cost of a proof is garbling and evaluating the PRF F** which is quite efficient for traditional block ciphers like AES

Our Approach

- We make use of the Zero-Knowledge from Garbled Circuits (ZKGC) paradigm [Jawurek Kerschbaum Orlandi 13] augmented by a conditional-disclosure round compression technique [Ganesh **K** Patra Sarkar 18]

- We develop the following new techniques to tailor and improve this paradigm:
  - Garbling gadget to output the exponentiation of an encoded input
    - Improves $\tilde{O}(\kappa^2)$ Boolean gate gadget [CGM16] to $O(\kappa)$ Boolean gates
    - Also applies to anonymous credentials based on ECDSA [CGM16]
  - Custom C-OT protocol to preprocess all public-key operations
    - Input encoding now cheaper than garbling F
    - Also applies to Distributed Symmetric Encryption [Agrawal Mohassel Mukherjee Rindal 18]

Garbling Gadget

$$\varphi(y_1 y_2 \ldots y_\eta) = y \cdot G$$

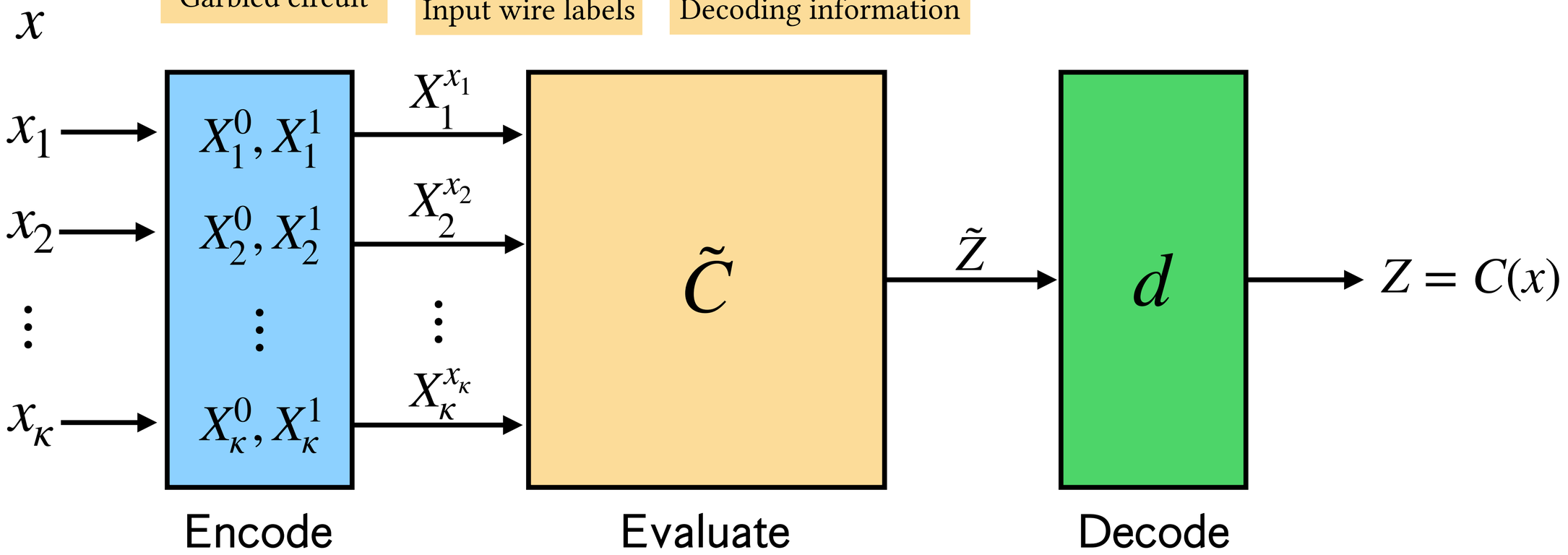
- **Task**: garble $C(x) = \varphi(F(x))$ where $F : \{0,1\}^\kappa \to \{0,1\}^\eta$, and $\varphi : \{0,1\}^\eta \to \mathbb{G}$

  Boolean circuit — Exponentiation / Curve multiplication

- **Our approach**:

$$\tilde{C}, \{X_i^0, X_i^1\}_{i\in[\kappa]}, d \leftarrow \mathsf{Garble}(C)$$

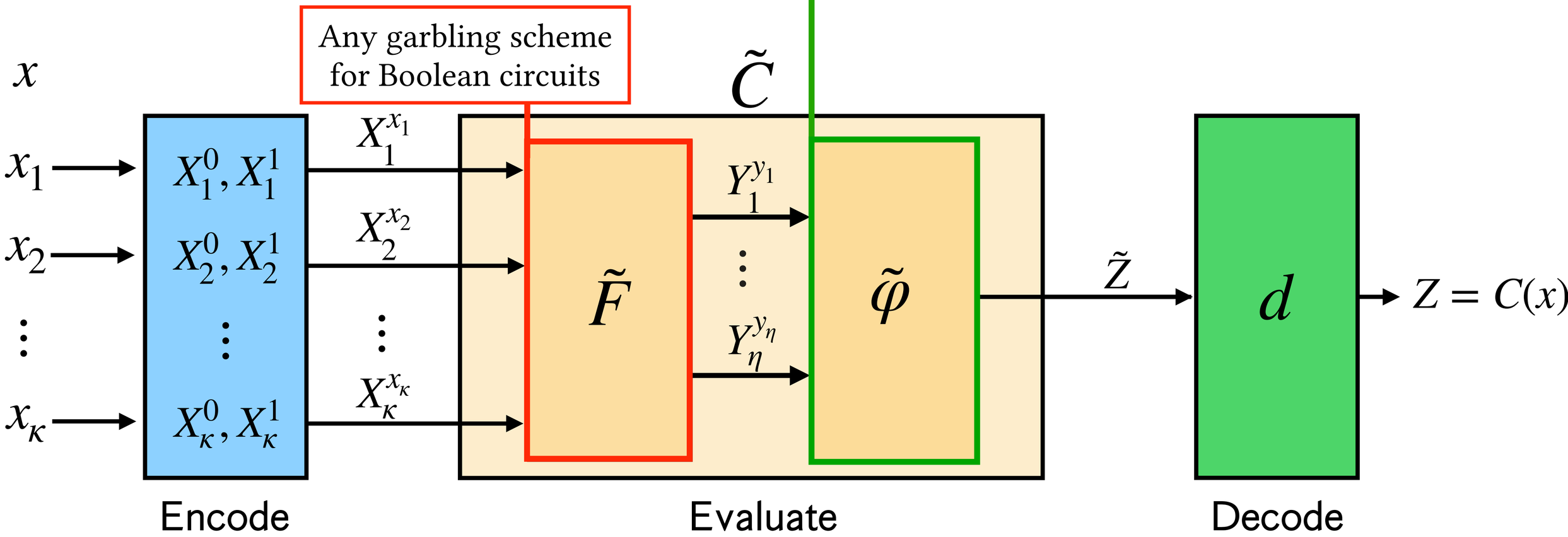
Garbled circuit — Input wire labels — Decoding information



Garbling Gadget

$\varphi(y_1 y_2 \ldots y_\eta) = y \cdot G$

- **Task**: garble $C(x) = \varphi(F(x))$ where $F : \{0,1\}^\kappa \to \{0,1\}^\eta$, and $\varphi : \{0,1\}^\eta \to \mathbb{G}$
  - F: Boolean circuit
  - φ: Exponentiation / Curve multiplication
- **Our approach**: New gadget inspired by Oblivious Linear Evaluation of [Gilboa 99]

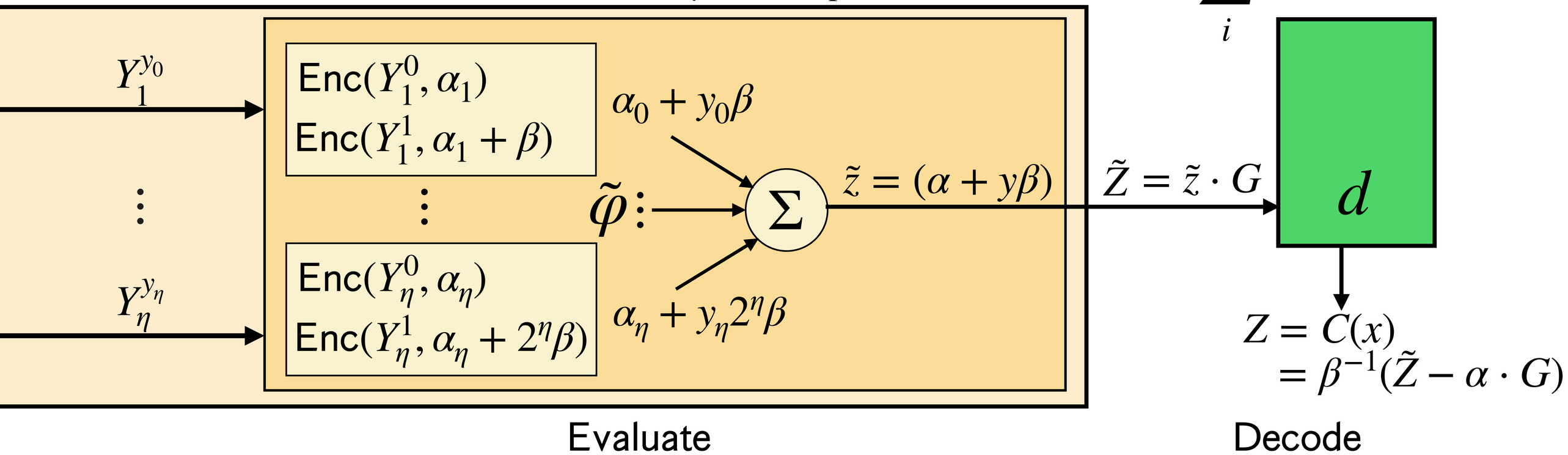


Garbling Gadget

$\varphi(y_1 y_2 \ldots y_\eta) = y \cdot G$

- **Task**: garble $C(x) = \varphi(F(x))$ where $F : \{0,1\}^\kappa \to \{0,1\}^\eta$, and $\varphi : \{0,1\}^\eta \to \mathbb{G}$
  - F: Boolean circuit
  - φ: Exponentiation / Curve multiplication
- **Our approach**: New gadget inspired by Oblivious Linear Evaluation of [Gilboa 99]

$$(\beta, \alpha_1, \alpha_2, \cdots, \alpha_\eta) \leftarrow \mathbb{Z}_q^{\eta+1}, \text{ and set } \alpha = \sum_i \alpha_i$$



Garbling Gadget

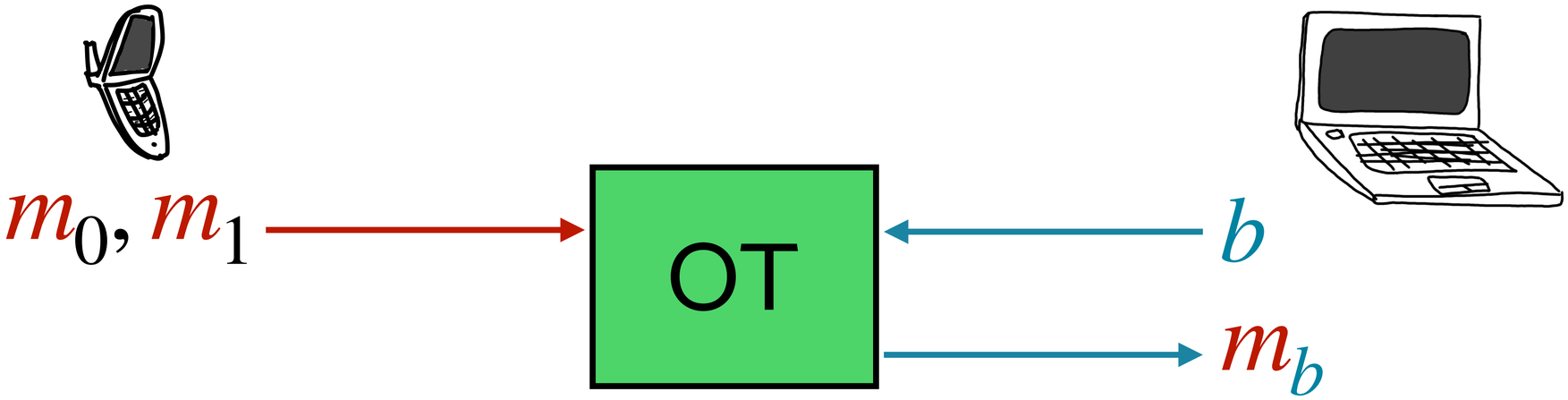
- Security intuition:
  - **Soundness/Authenticity**: (α, β) act as information-theoretic MAC key — $\tilde{Z}^* = \alpha + \beta y^*$ is hard to forge
  - **Zero-knowledge/Uniqueness**: Once $(\alpha, \beta), Z$ are fixed, one can simulate $\tilde{Z}$ perfectly; $\tilde{Z} = \beta^{-1} \cdot (Z - \alpha \cdot G)$ resembles simulation of a Schnorr signature
- **Efficiency**: η cipher texts of size $\log_2(q)$ bits each — equivalent to garbling $2\log_2(q) \in O(\kappa)$ Boolean AND gates (compare with garbling $\tilde{O}(\kappa^2)$ [CGM16])

Concrete improvement: $60\times$ in computation and bandwidth

Cost is now insignificant compared to garbling PRF circuit

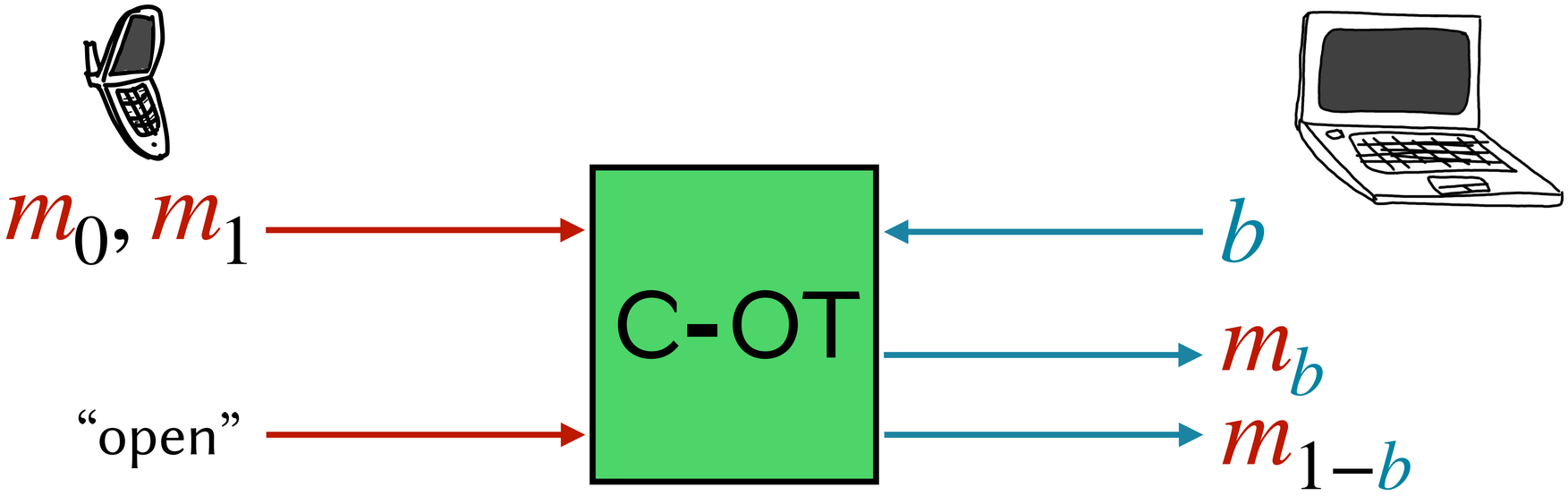
Committed OT

- Executed to encode each bit of input to the garbled circuit [JKO13]



Committed OT

- Executed to encode each bit of input to the garbled circuit [JKO13]



- Unclear how to preprocess public key operations for C-OT; standard OT Extension does not natively support decommitting messages without sacrificing setup
- **Relaxation**: input b once and transmit an unbounded number of message pairs

C-OT: Naive Attempt

$\mathsf{sd}_0, \mathsf{sd}_1$ → OT ← b

OT → sd_b

x

$p_0 = \mathsf{PRF}(\mathsf{sd}_0, x)$
$p_1 = \mathsf{PRF}(\mathsf{sd}_1, x)$

$\mathsf{ct}_0 = p_0 \oplus m_0$
$\mathsf{ct}_1 = p_1 \oplus m_1$ →

Retrieve
$m_b = \mathsf{ct}_b \oplus \mathsf{PRF}(\mathsf{sd}_b, x)$

"open"
p_0, p_1 →

$m_{1-b} = \mathsf{ct}_{1-b} \oplus p_{1-b}$

C-OT: Naive Attempt

$\mathsf{sd}_0, \mathsf{sd}_1$

OT

0

sd_0

$b = 0$

x

$p_0 = \mathsf{PRF}(\mathsf{sd}_0, x)$
$p_1 = \mathsf{PRF}(\mathsf{sd}_1, x)$

$\mathsf{ct}_0 = p_0 \oplus m_0$
$\mathsf{ct}_1 = p_1 \oplus m_1$

Retrieve
$m_0 = \mathsf{ct}_0 \oplus \mathsf{PRF}(\mathsf{sd}_0, x)$

"open"
p_0, p_1

$m_1 = \mathsf{ct}_1 \oplus p_1$

C-OT: Naive Attempt

$\mathsf{sd}_0, \mathsf{sd}_1$

OT

0

sd_0

x

$p_0 = \mathsf{PRF}(\mathsf{sd}_0, x)$

$p_1 = \mathsf{PRF}(\mathsf{sd}_1, x)$

$\mathsf{ct}_0 = p_0 \oplus m_0$

$\mathsf{ct}_1 = p_1 \oplus m_1$

Retrieve

$m_0 = \mathsf{ct}_0 \oplus \mathsf{PRF}(\mathsf{sd}_0, x)$

"open"

p_0, p_1^*

$m_1 = \mathsf{ct}_1 \oplus p_1$

C-OT: Naive Attempt

$\mathsf{sd}_0, \mathsf{sd}_1$ → OT ← 0

OT → sd_0

x

$p_0 = \mathsf{PRF}(\mathsf{sd}_0, x)$
$p_1 = \mathsf{PRF}(\mathsf{sd}_1, x)$

$\mathsf{ct}_0 = p_0 \oplus m_0$
$\mathsf{ct}_1 = p_1 \oplus m_1$ →

Retrieve
$m_0 = \mathsf{ct}_0 \oplus \mathsf{PRF}(\mathsf{sd}_0, x)$

"open"
p_0, p_1^* →

$m_1^* = \mathsf{ct}_1 \oplus p_1^*$

Tool: UC Commitments

- We make use of Universally Composable commitments [Canetti 01] to construct our C-OT scheme

- Specifically, UC commitments that permit the following algorithms:
  - $\mathsf{Setup}(1^\kappa) : (\mathsf{td}, \mathsf{ck}, \mathsf{vk})$
    - Trapdoor
    - Verification key
    - Commitment key
  - $\mathsf{Commit}(\mathsf{ck}, m) : \mathsf{Com}, d$
    - Decommitment information
  - $\mathsf{Verify}(\mathsf{vk}, \mathsf{Com}, m, d) : \{0,1\}$
  - $\mathsf{Extract}(\mathsf{td}, \mathsf{Com}) : m$
    - **Straight-line algorithm** that extracts the committed message
    - Conventionally a “proof artefact” that we actually execute in our construction

C-OT from UC Commitments

Setup

$\mathsf{ck}_0, \mathsf{ck}_1$

b

$\mathsf{td}_b, \mathsf{vk}_0, \mathsf{vk}_1$

$\mathsf{ct}_0, d_0 \leftarrow \mathsf{Commit}(\mathsf{ck}_0, m_0)$
$\mathsf{ct}_1, d_1 \leftarrow \mathsf{Commit}(\mathsf{ck}_1, m_1)$

$\mathsf{ct}_0, \mathsf{ct}_1$

Retrieve
$m_b = \mathsf{Extract}(\mathsf{td}_b, \mathsf{ct}_b)$

"open"

$(m_0, d_0), (m_1, d_1)$

$\mathsf{Verify}(\mathsf{vk}_0, \mathsf{ct}_0, m_0, d_0)$
$\wedge\ \mathsf{Verify}(\mathsf{vk}_1, \mathsf{ct}_1, m_1, d_1)$

C-OT from UC Commitments

- Security **follows directly from straight-line extraction** and equivocability of the UC commitment scheme

- Instantiation of the commitment scheme borrows ideas from literature on UC Commitments from Error Correcting Codes
  [Cascudo Damgård David Giacomelli Nielsen Trifiletti 15]

- Committing, Verification, and Extraction require only PRF evaluations (and some hashing) which translates to the online complexity of C-OT

- **Concretely** for 128-bit computational and 60-bit statistical security, all C-OT instances combined cost ~16k PRF invocations (+some hashing) and ~82KB in bandwidth, which is roughly the **same cost as garbling and evaluating a single AES circuit**

In Summary

- The ZKGC paradigm [JKO13] is well suited to enabling stateless determinism in Threshold Schnorr when prioritising computational efficiency and standard assumptions
- The dominant cost of previous techniques [CGM16] applied to this setting were in input encoding (C-OT) and garbling exponentiations
- Our new techniques for these tasks improve their efficiencies to the point where the **dominant cost is now garbling and evaluating the PRF circuit**
- Our cost analysis estimates computation to be faster than using trusted hardware
- See the paper for: concrete cost analysis, optimisations and tricks, details

Thanks!

ccs.neu.edu/~ykondi

ykondi@ccs.neu.edu

Thanks **Eysa Lee** for