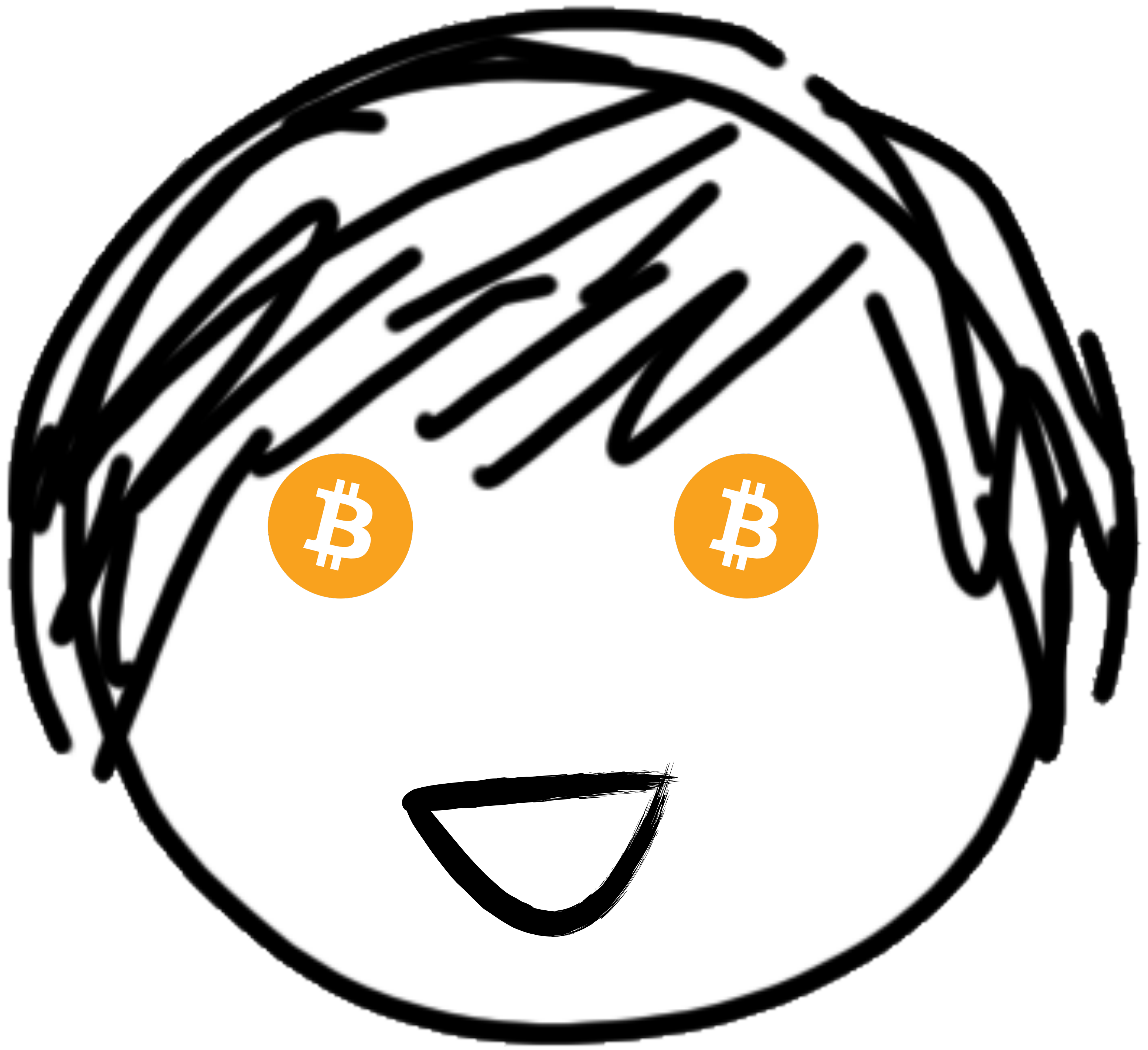
Threshold ECDSA from ECDSA assumptions

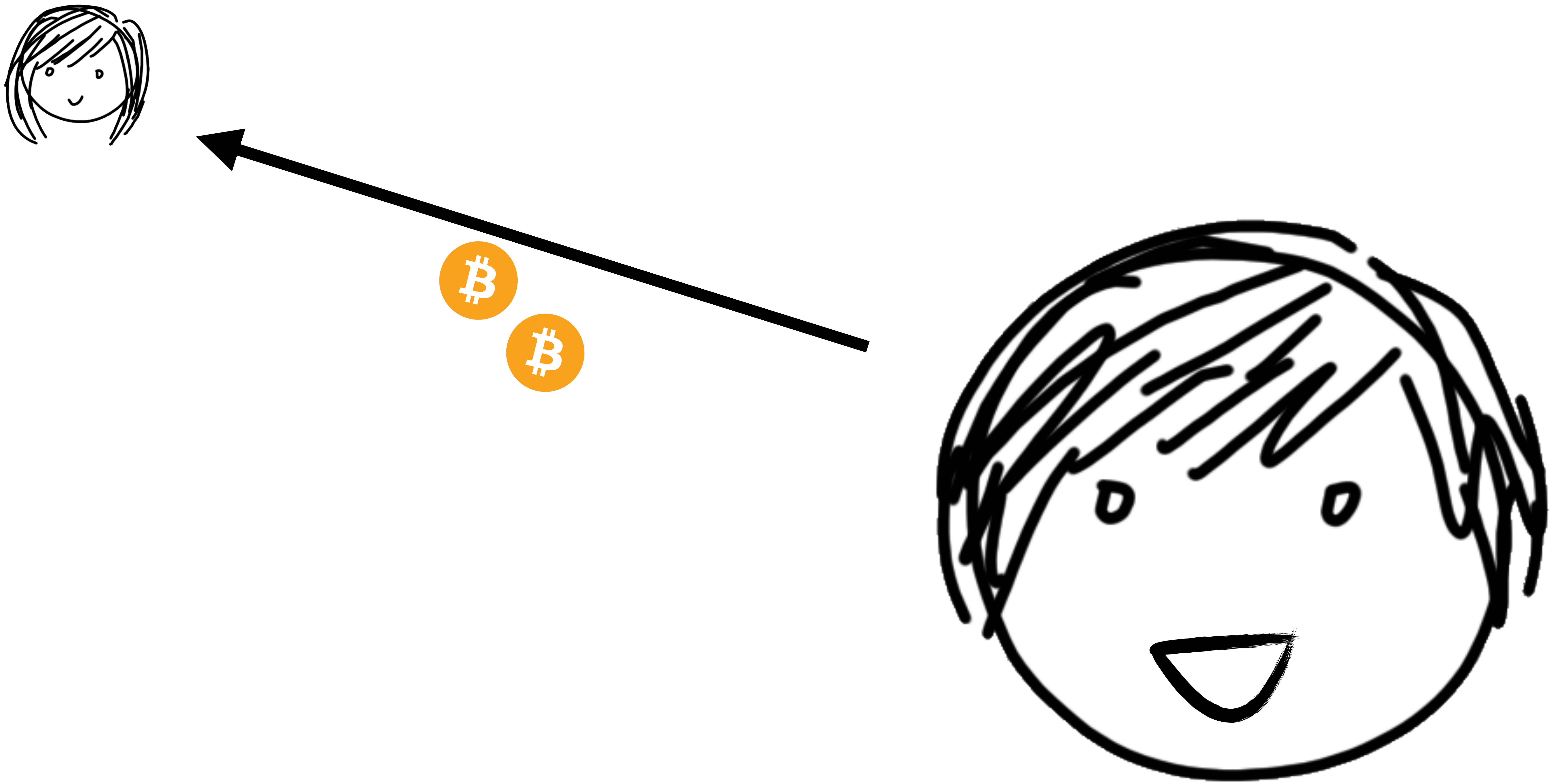
Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat

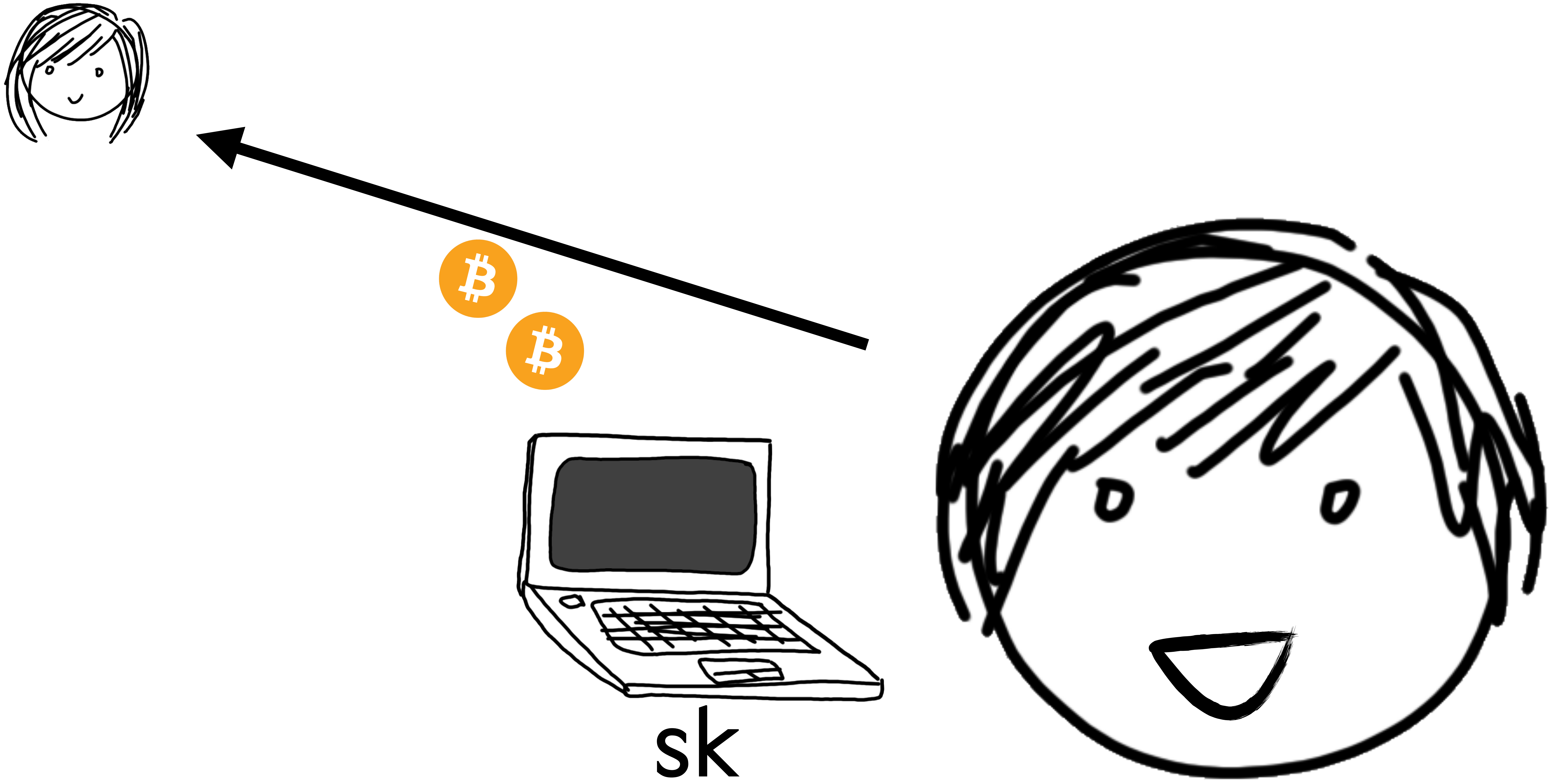
j@ckdoerner.net ykondi@ccs.neu.edu eysa@ccs.neu.edu abhi@neu.edu

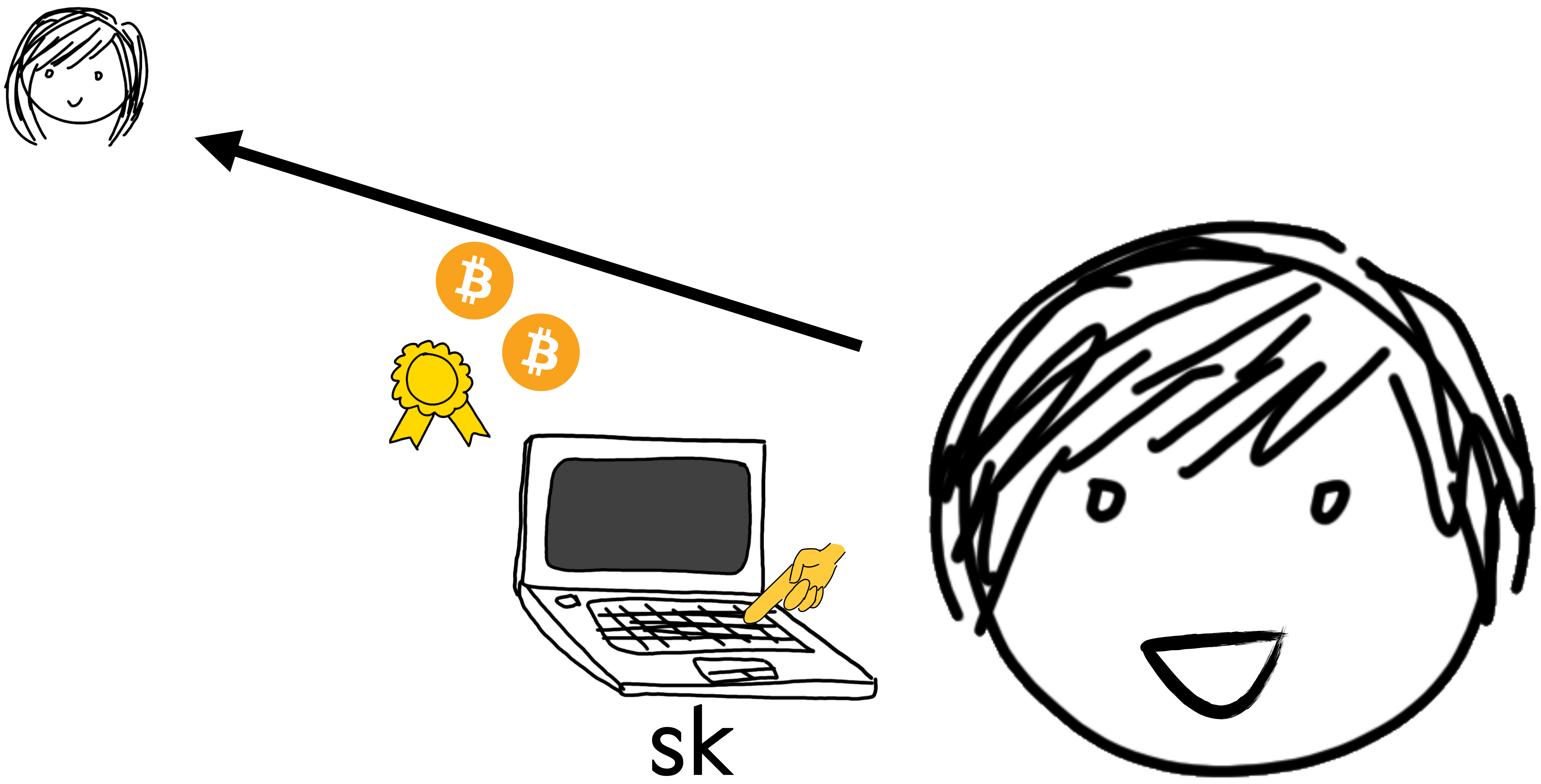
Northeastern University

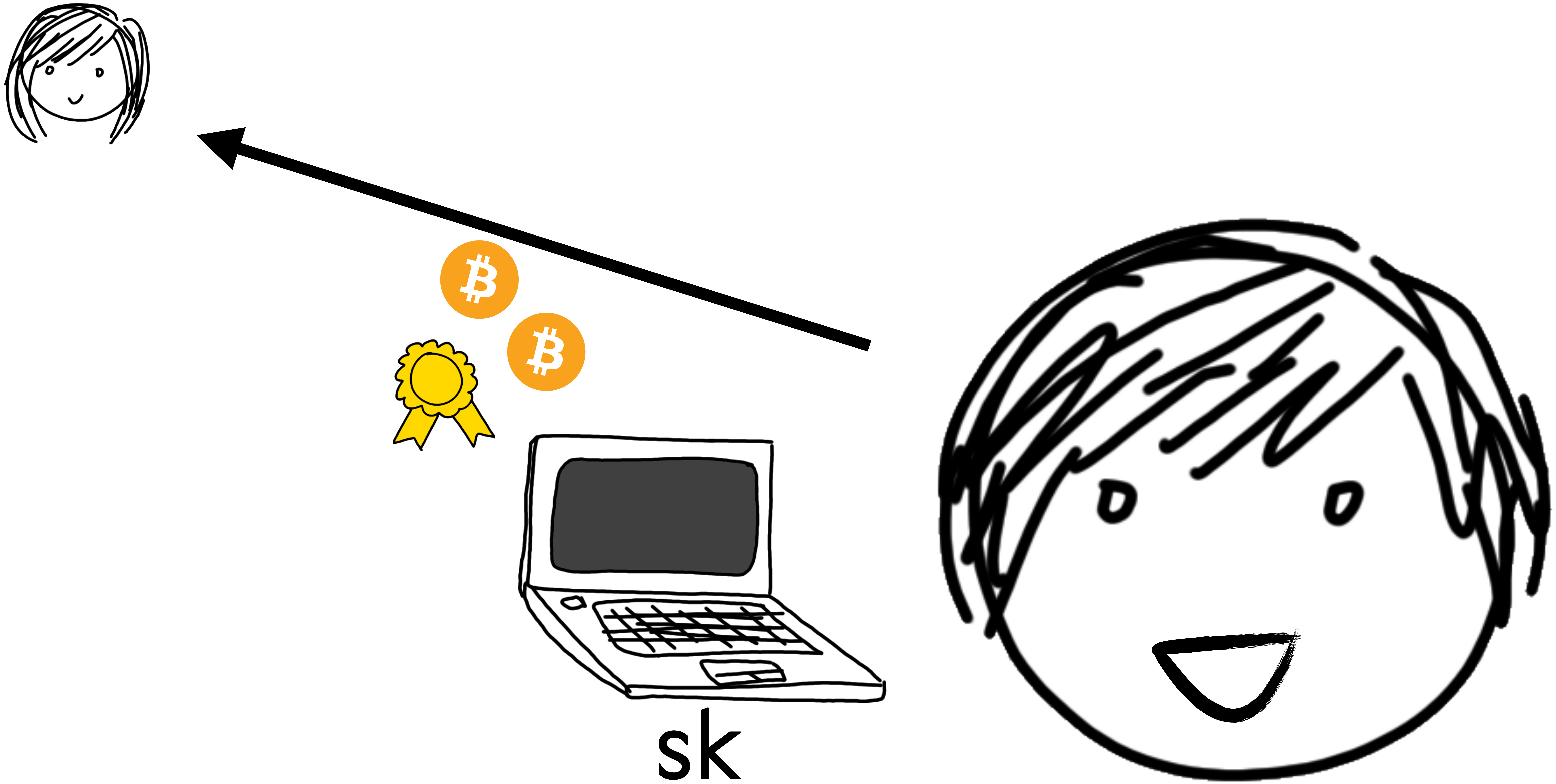
Appeared at IEEE S&P ’18 and ’19



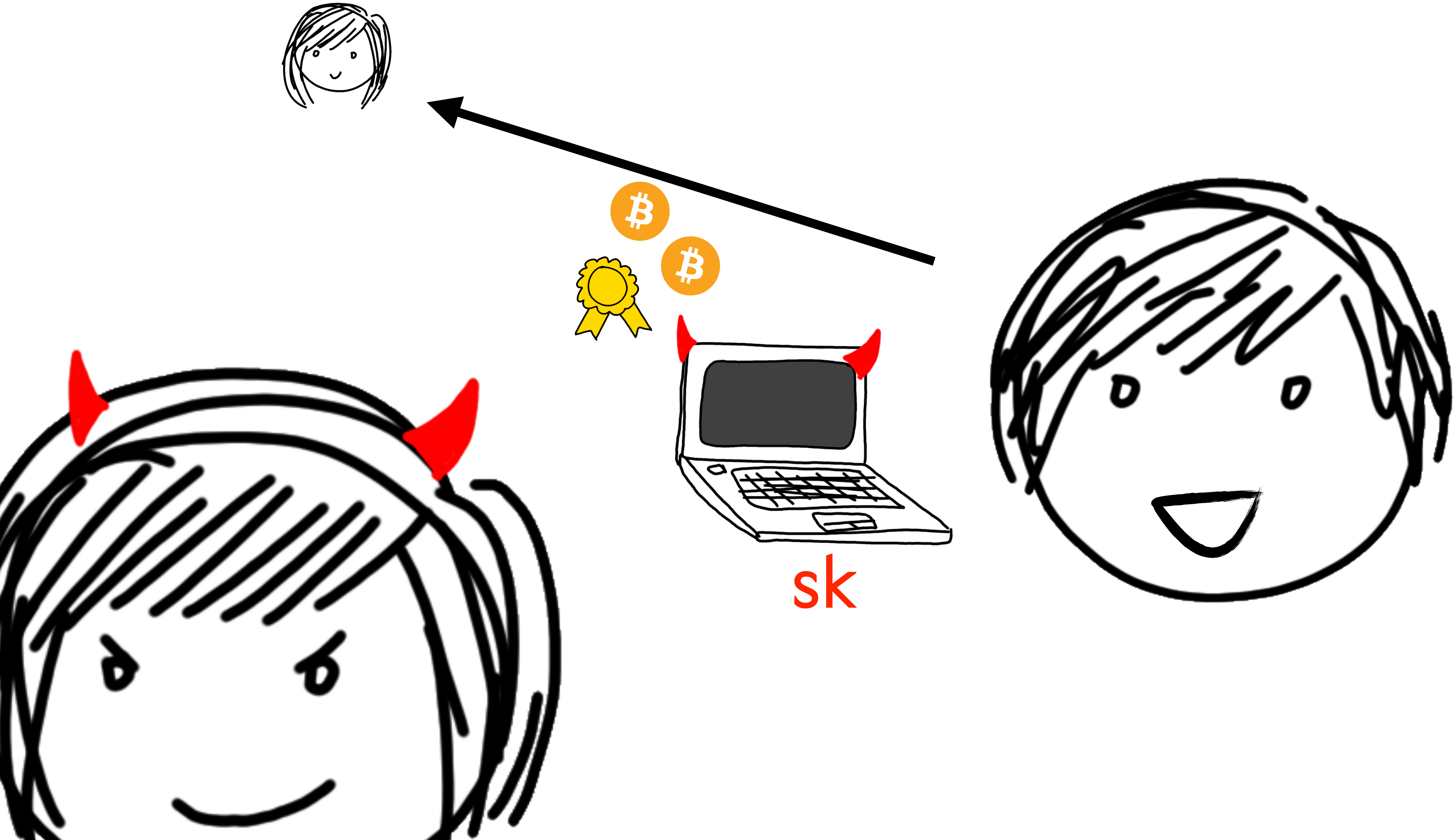



sk

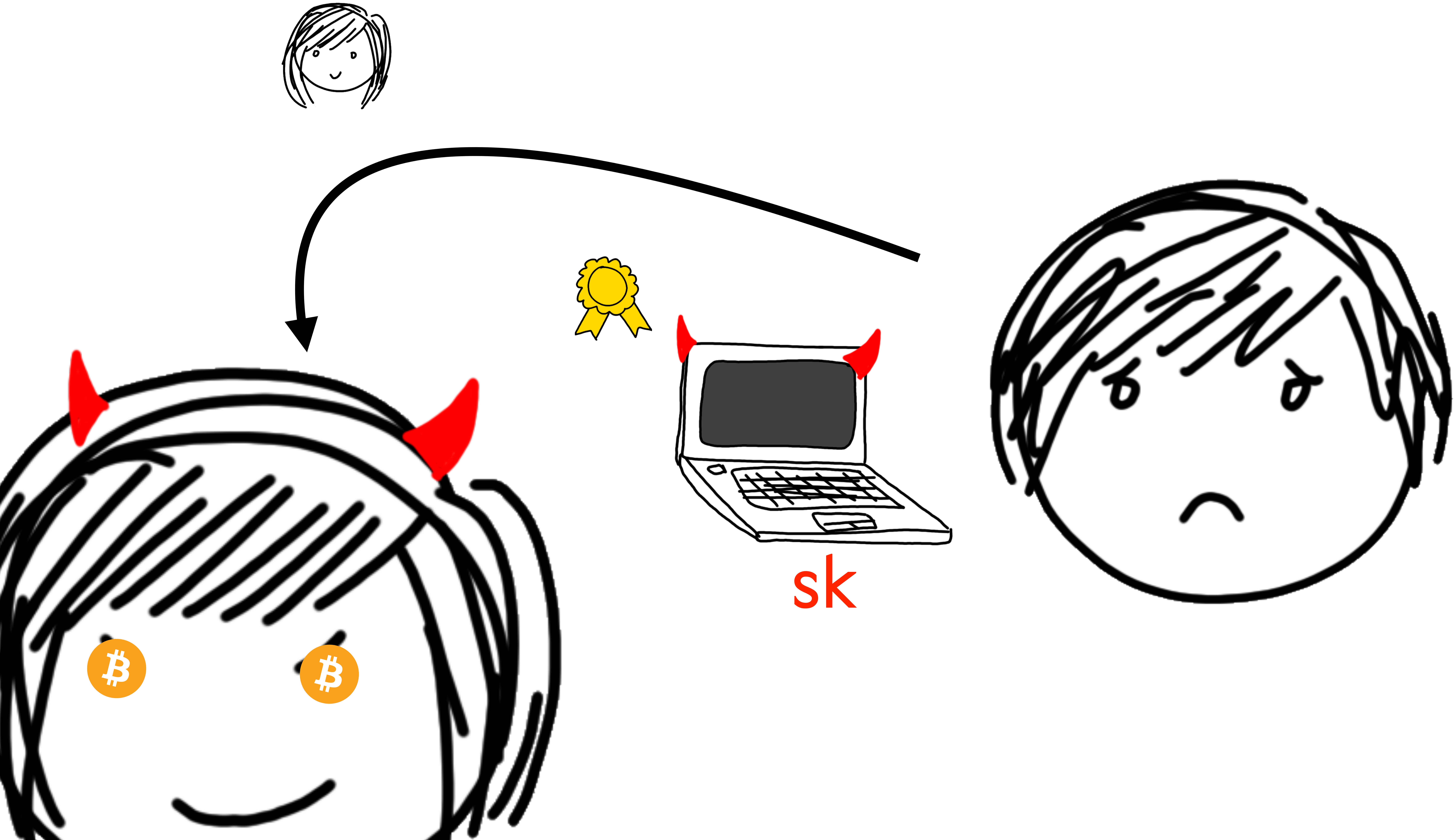

sk

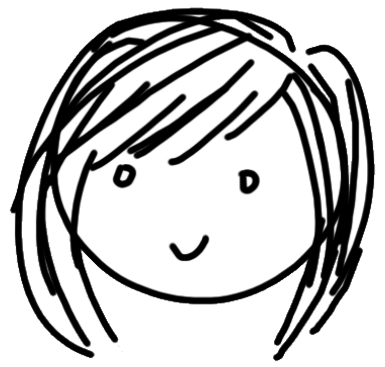

sk

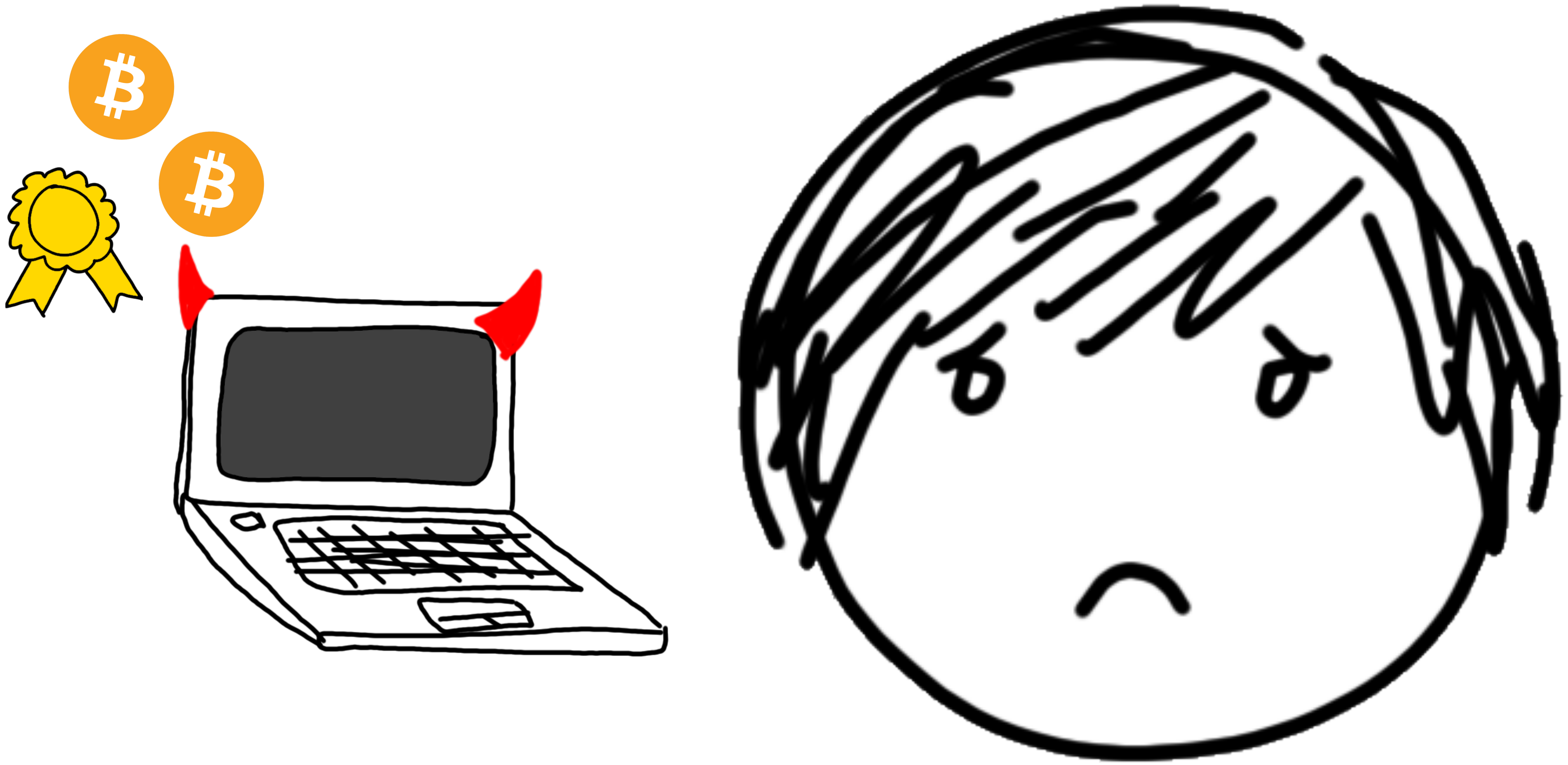
sk

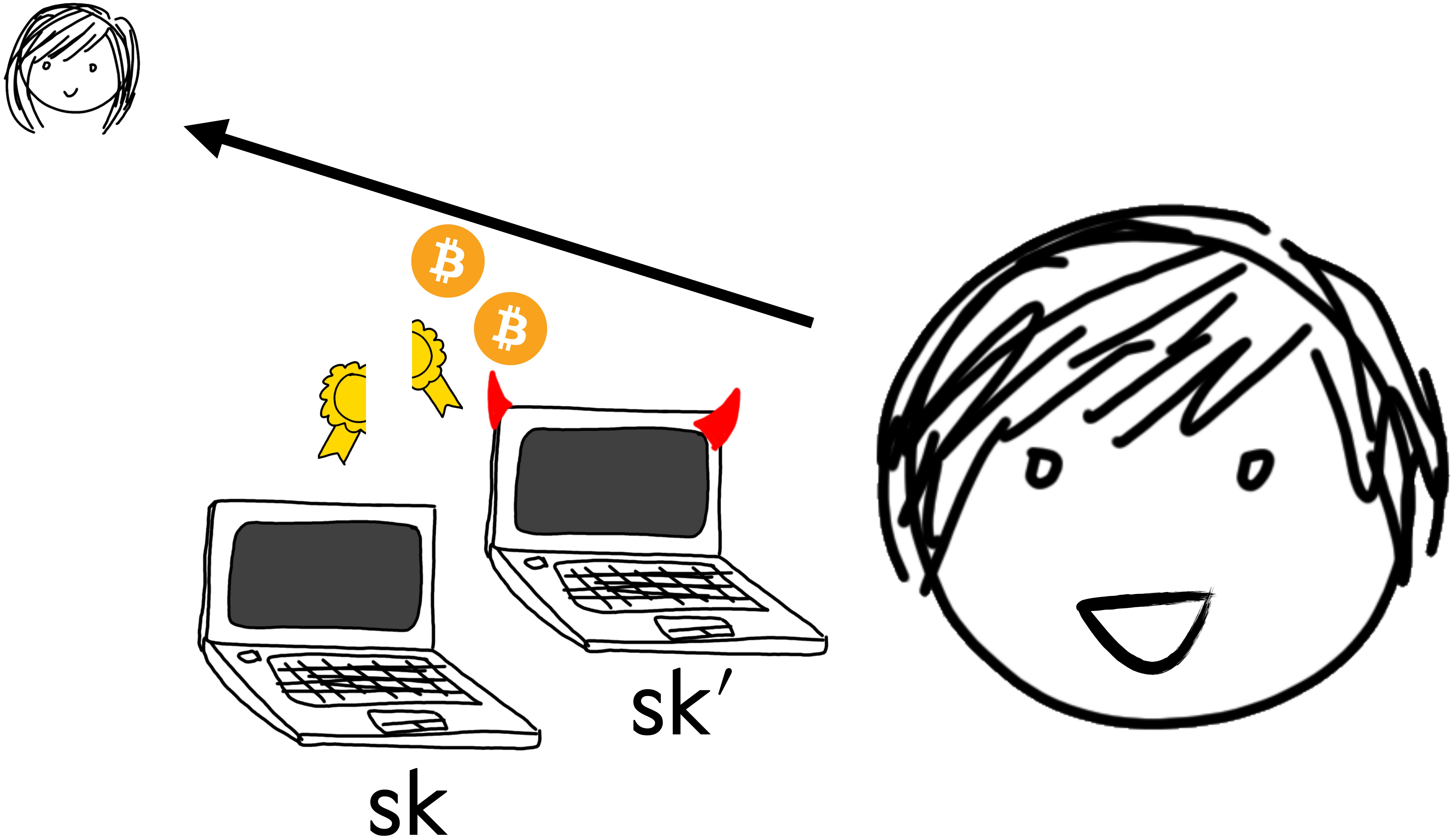

sk

sk

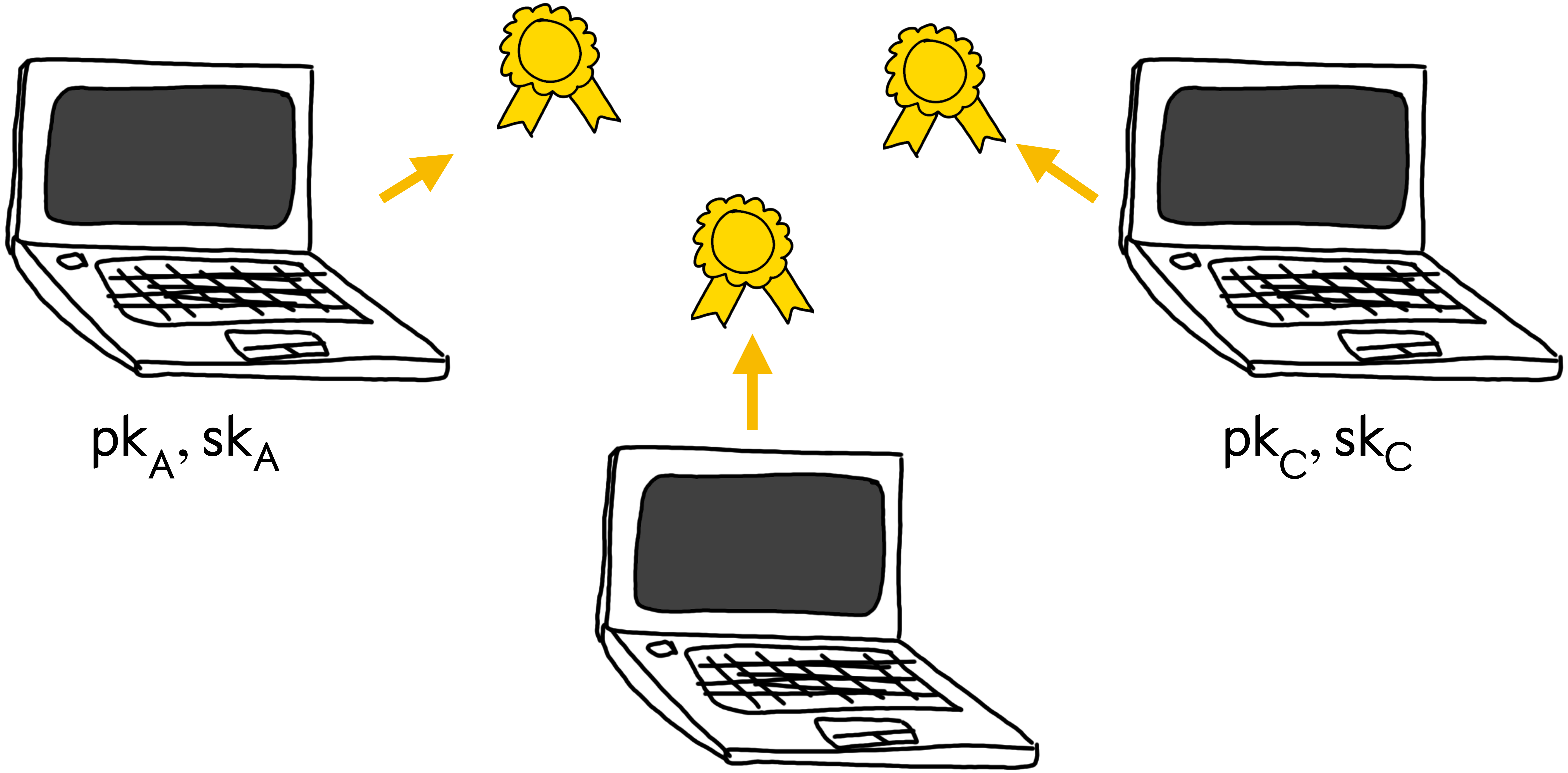

sk






sk′
sk

Multi-Sig



pk_A, sk_A

pk_B, sk_B

pk_C, sk_C

Disadvantages

No Anonymity

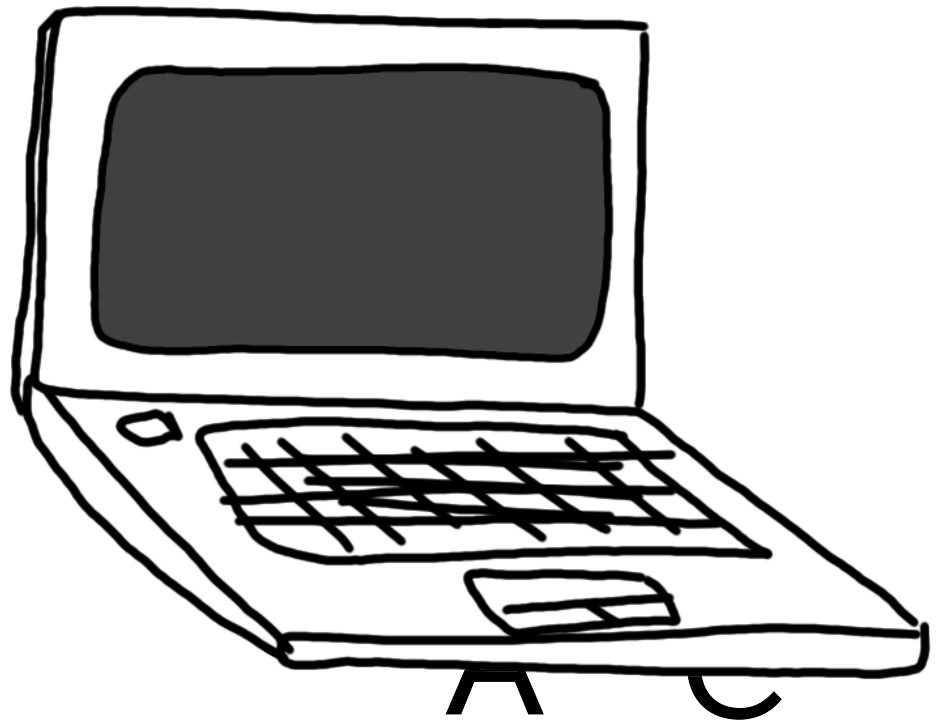
Size is linear in party count

Not compatible with other useful protocols
(e.g. web protocols, binary authentication)

Threshold Signature

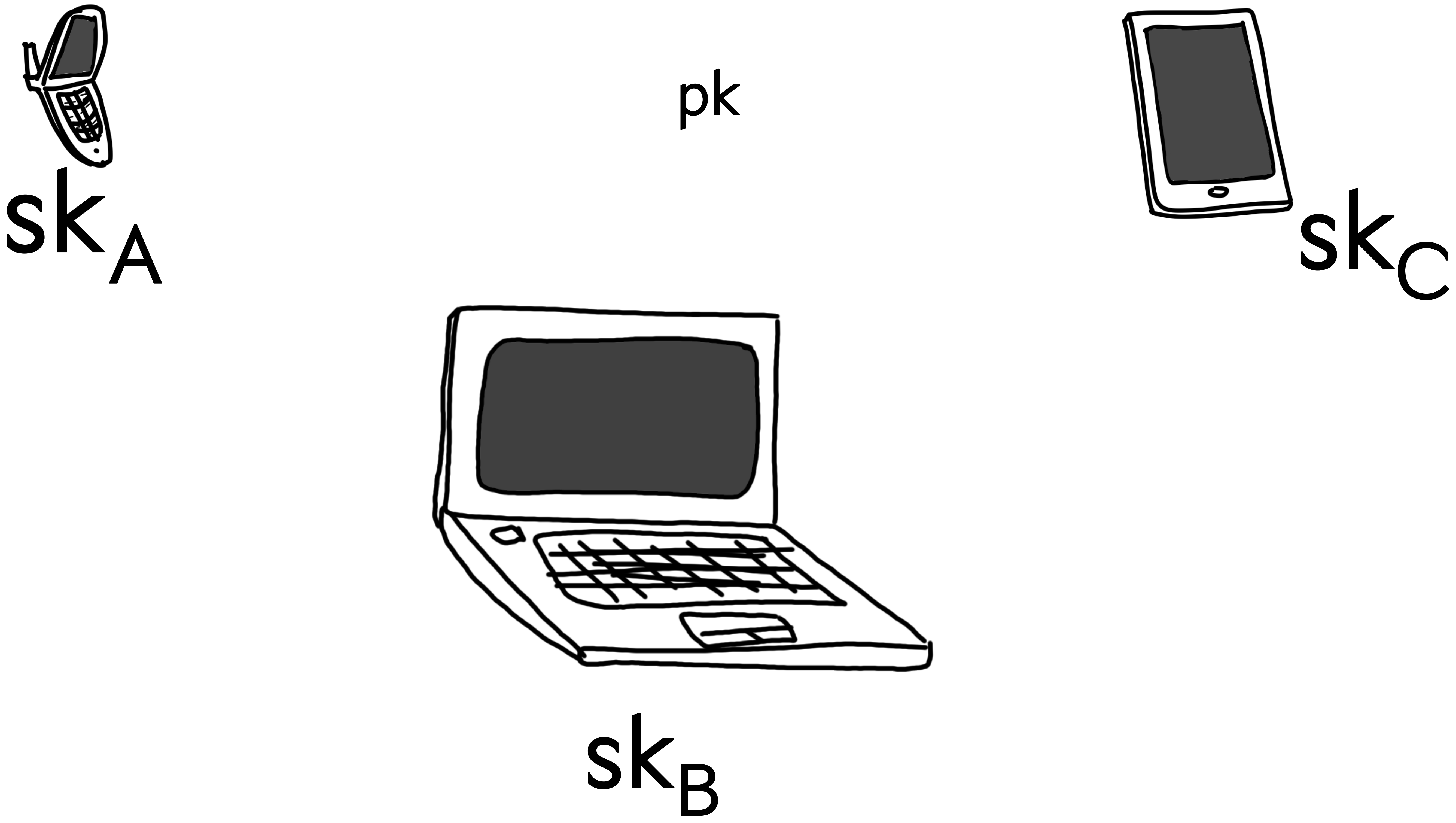
$$\{\mathsf{sk}_{\mathsf{A}}, \mathsf{sk}_{\mathsf{B}}, \mathsf{sk}_{C}\} \leftarrow \mathrm{Share}(\mathsf{sk})$$

pk



Threshold Signature

$$\{\mathsf{sk}_\mathsf{A}, \mathsf{sk}_\mathsf{B}, \mathsf{sk}_C\} \leftarrow \mathrm{Share}(\mathsf{sk})$$



pk

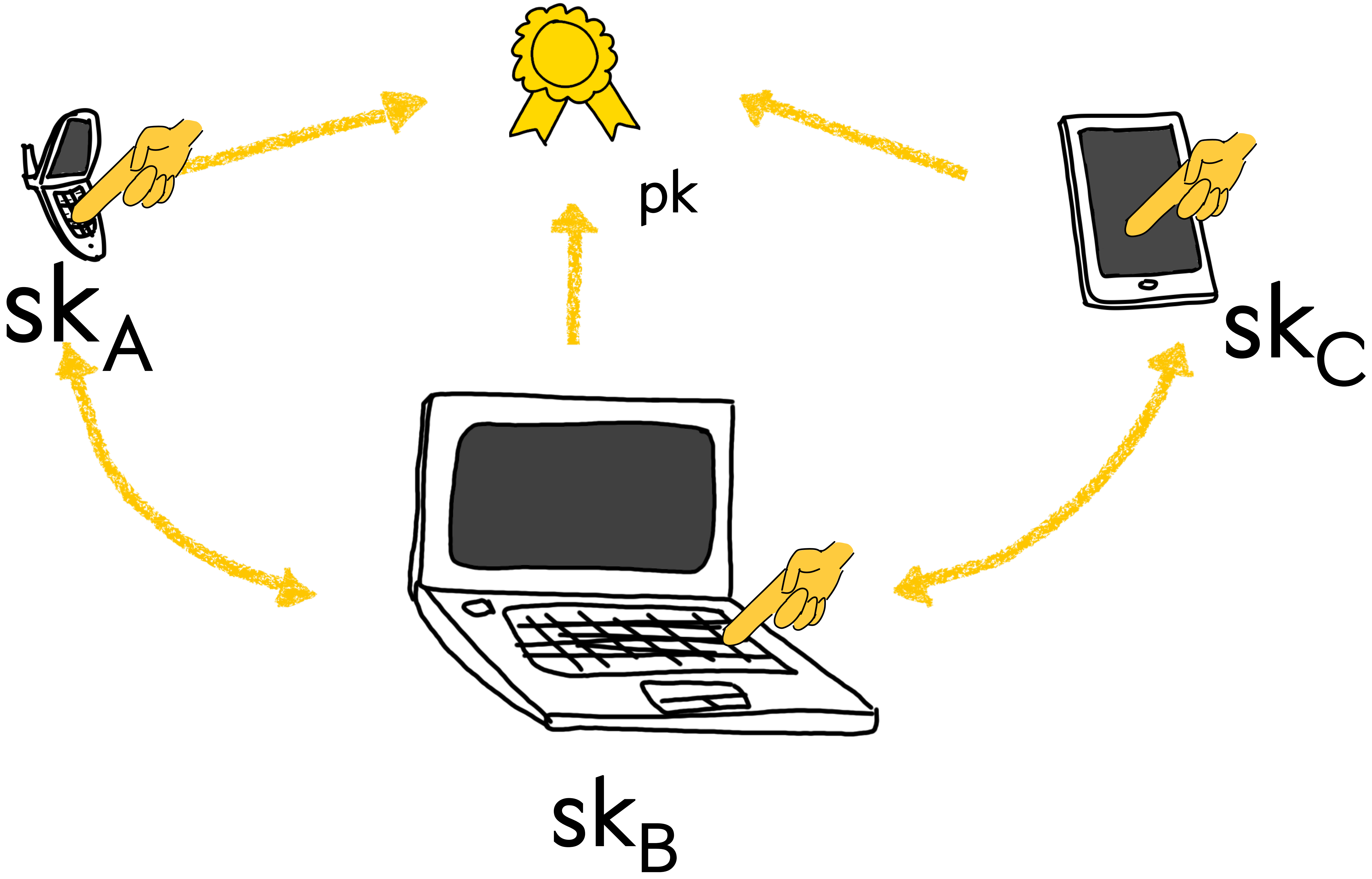
sk_A

sk_C

sk_B

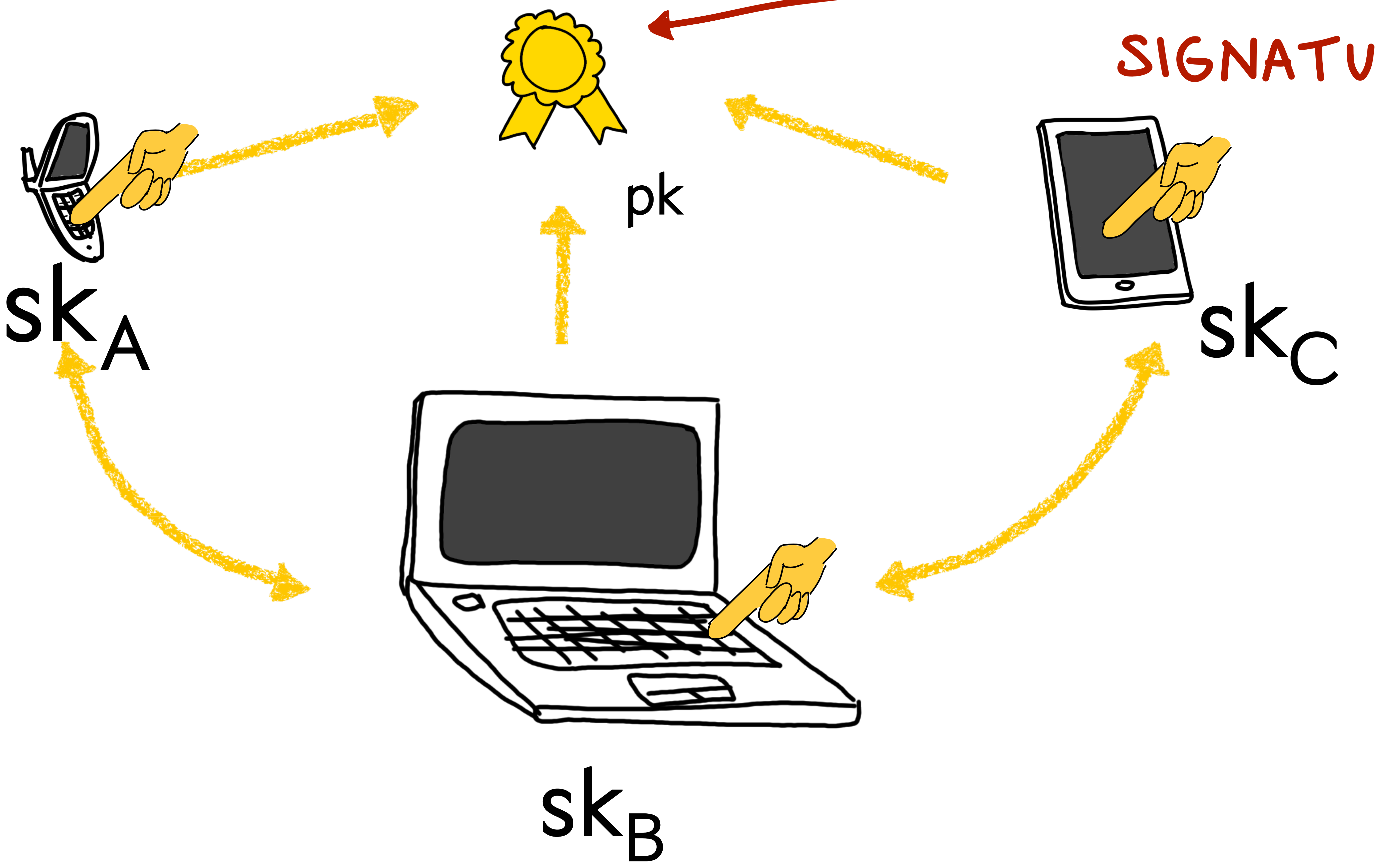
Threshold Signature

$$\{\mathsf{sk}_\mathsf{A}, \mathsf{sk}_\mathsf{B}, \mathsf{sk}_C\} \leftarrow \mathrm{Share}(\mathsf{sk})$$



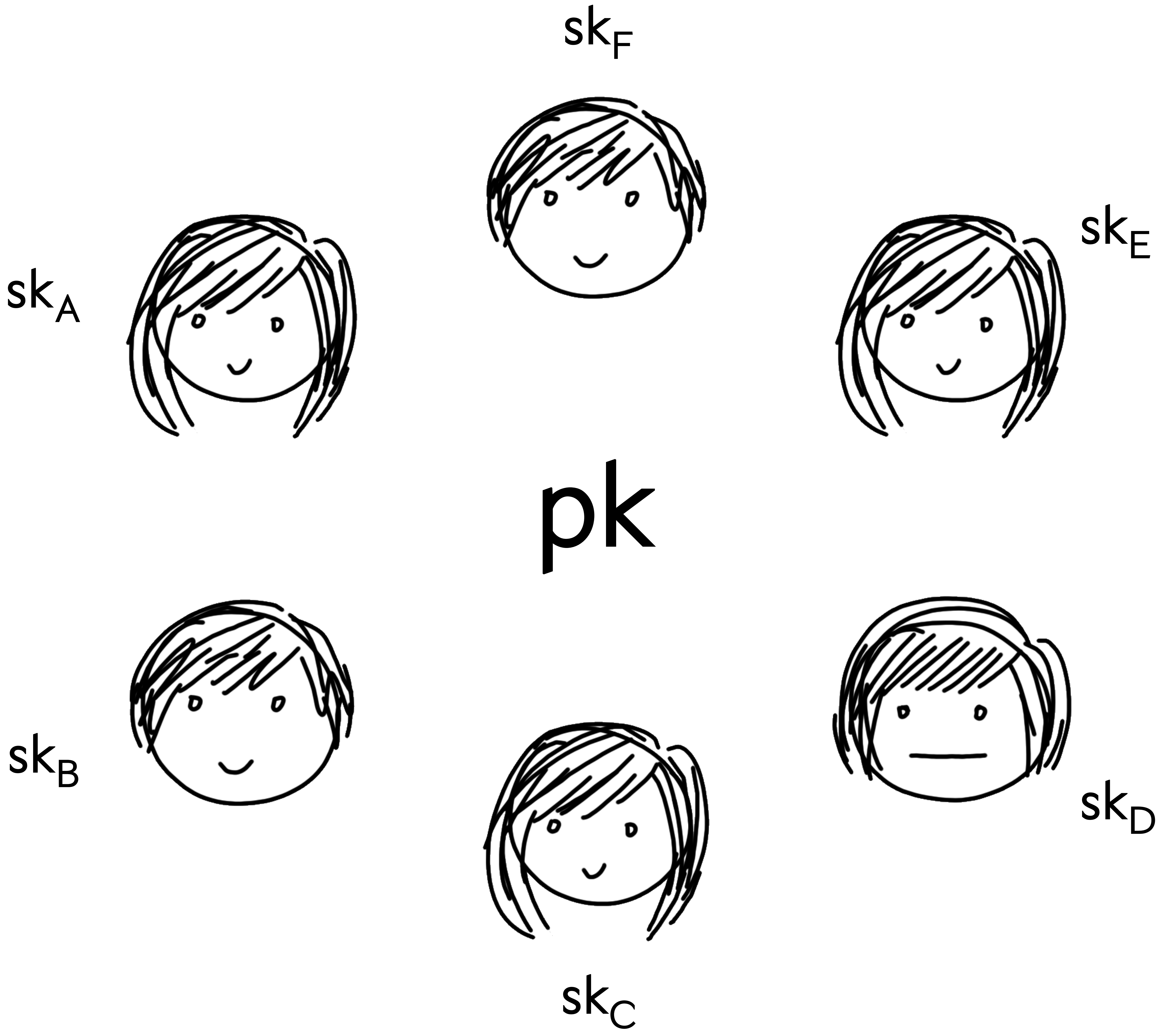
Threshold Signature

$\{\mathsf{sk}_{\mathsf{A}}, \mathsf{sk}_{\mathsf{B}}, \mathsf{sk}_{C}\} \leftarrow \text{Share}(\mathsf{sk})$



INDISTINGUISHABLE FROM ORDINARY SIGNATURE

3-of-n Signature Scheme



sk_F

sk_A

sk_E

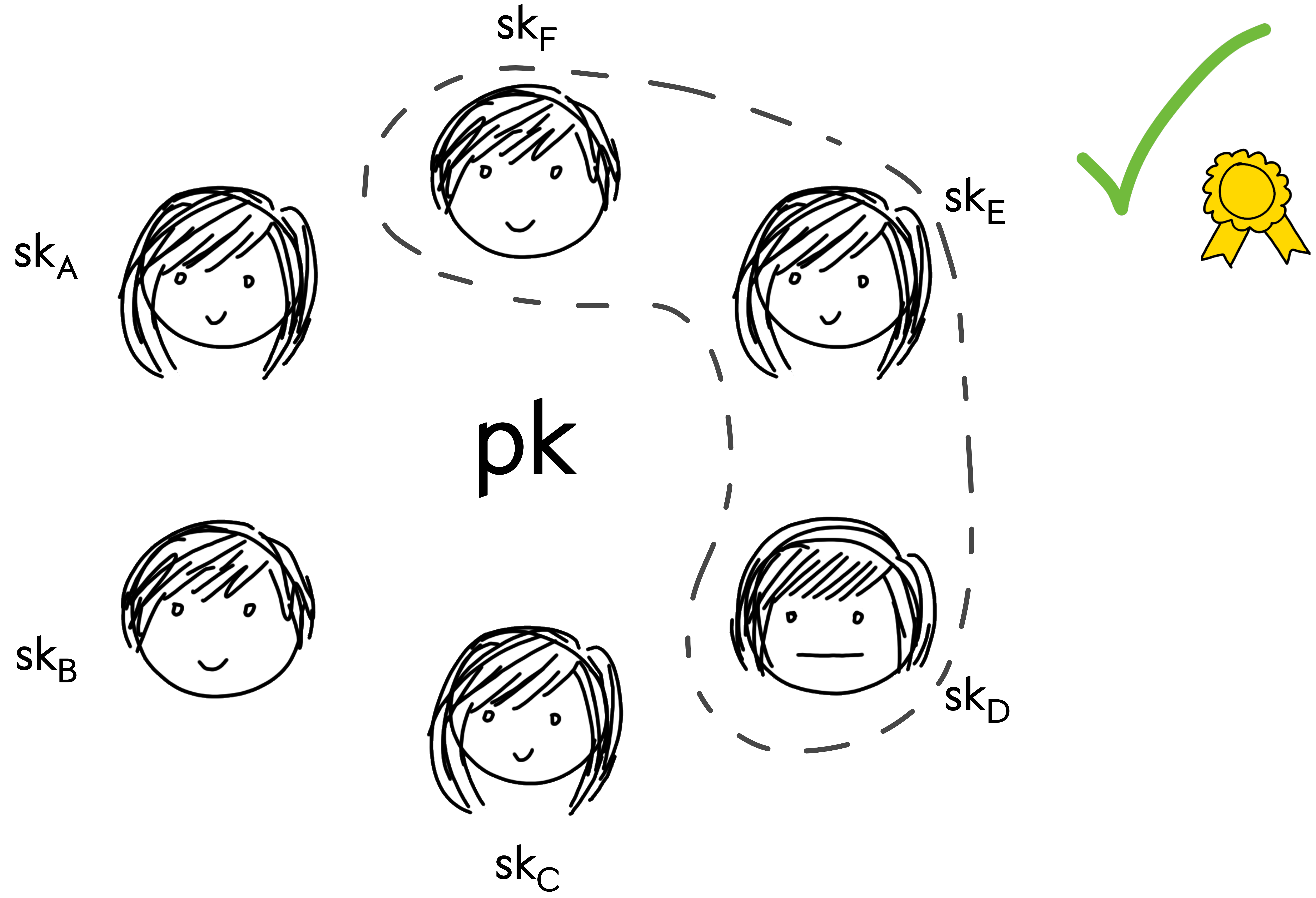
pk

sk_B

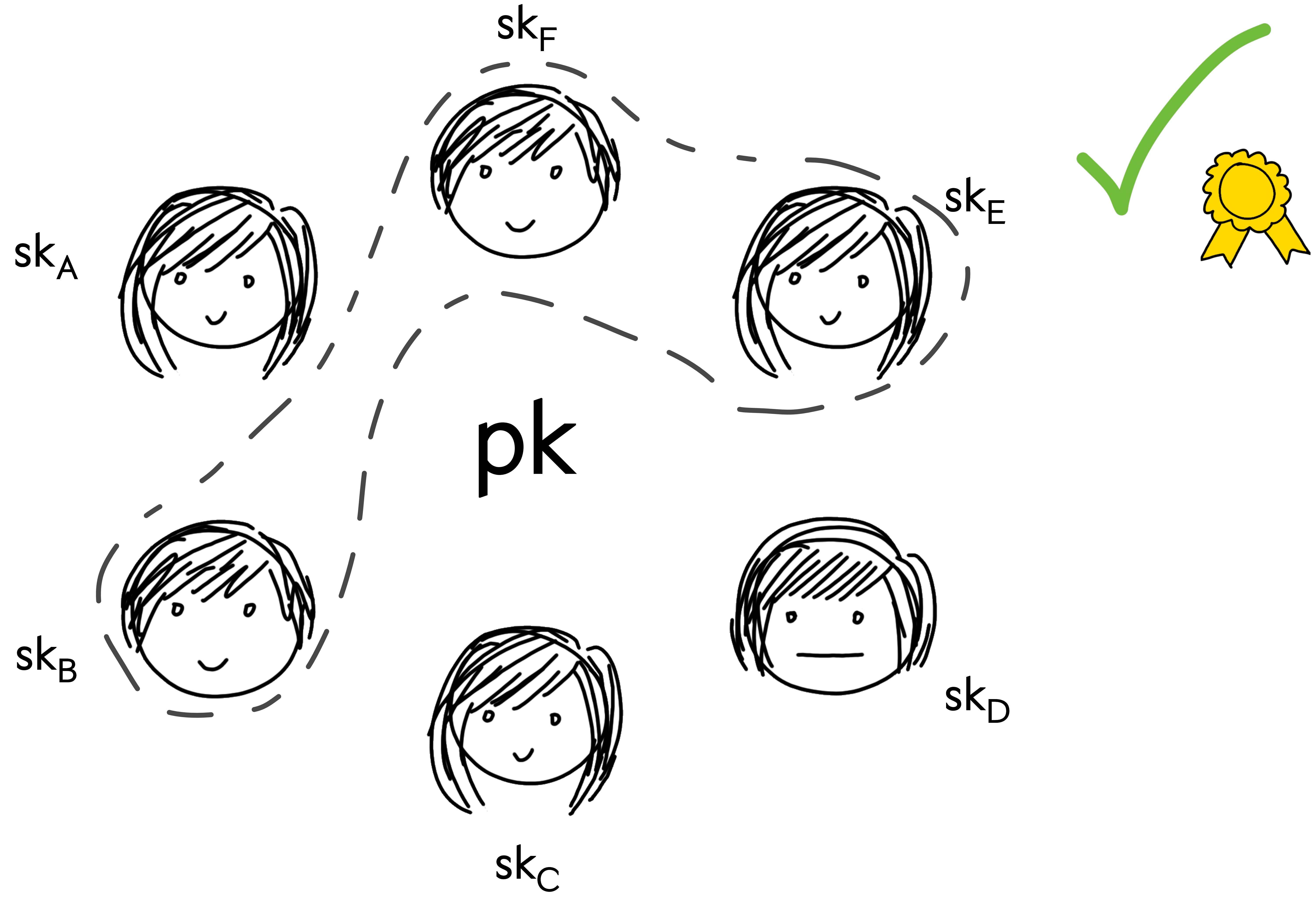
sk_D

sk_C

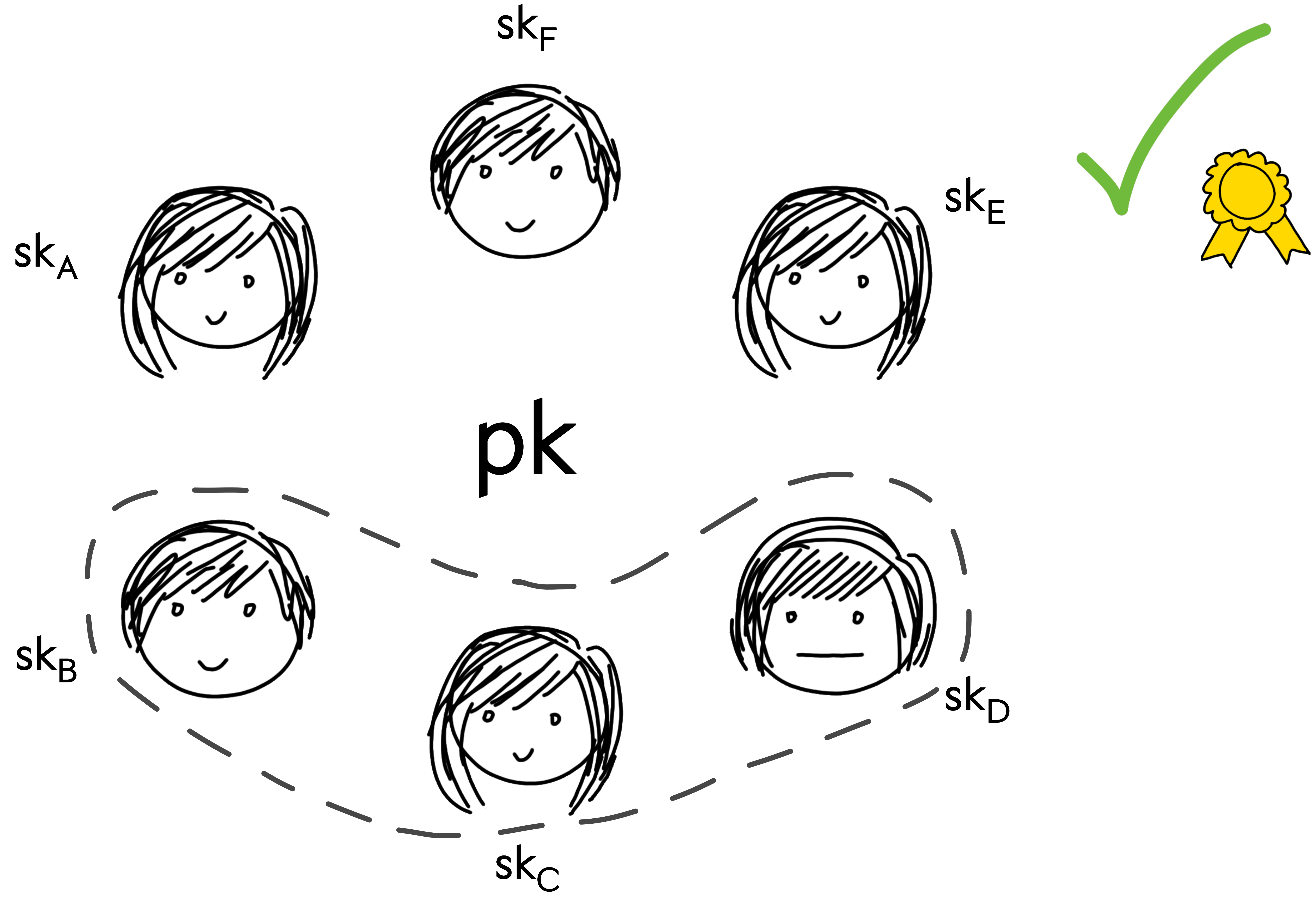
3-of-n Signature Scheme



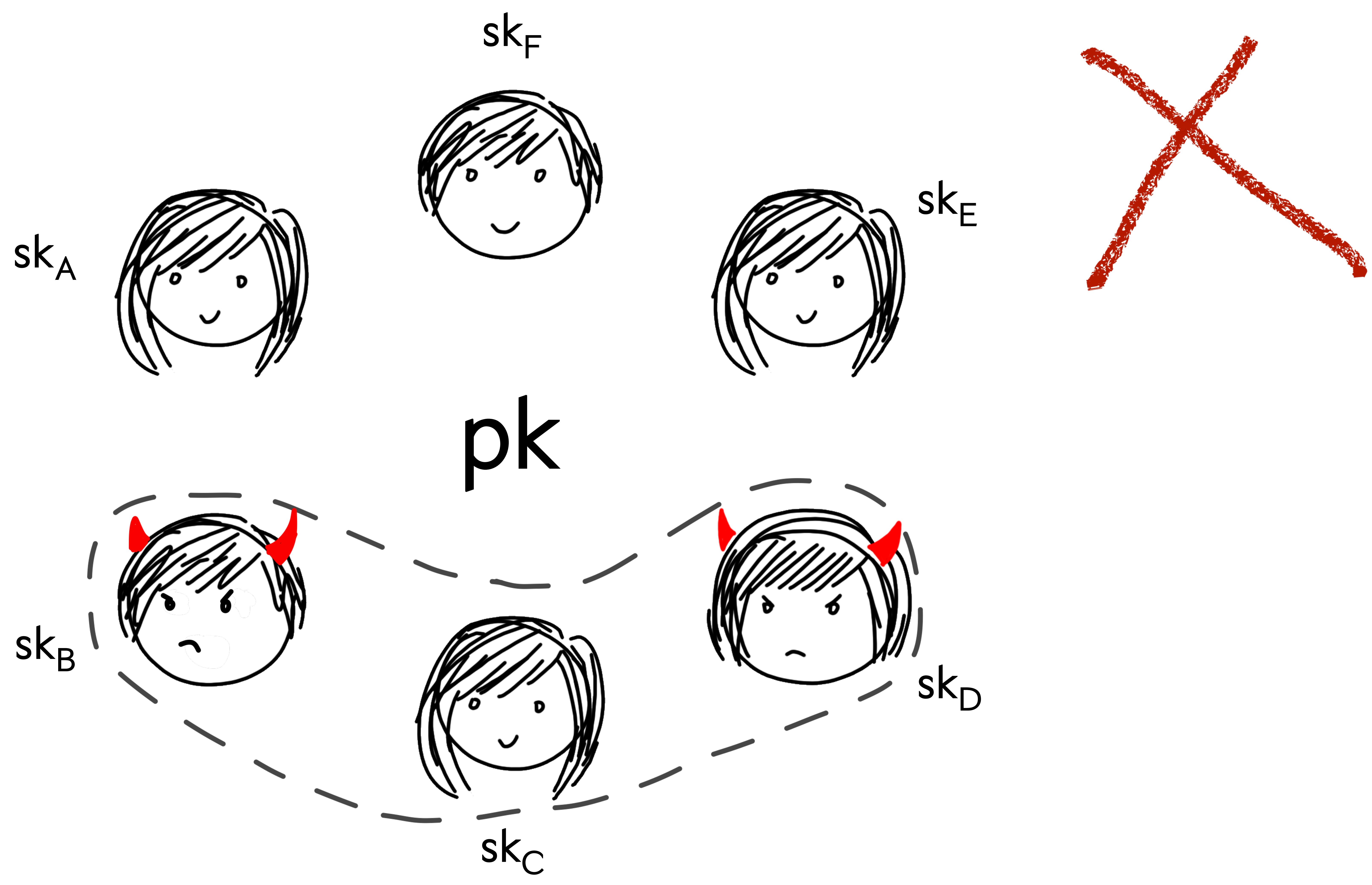
3-of-n Signature Scheme



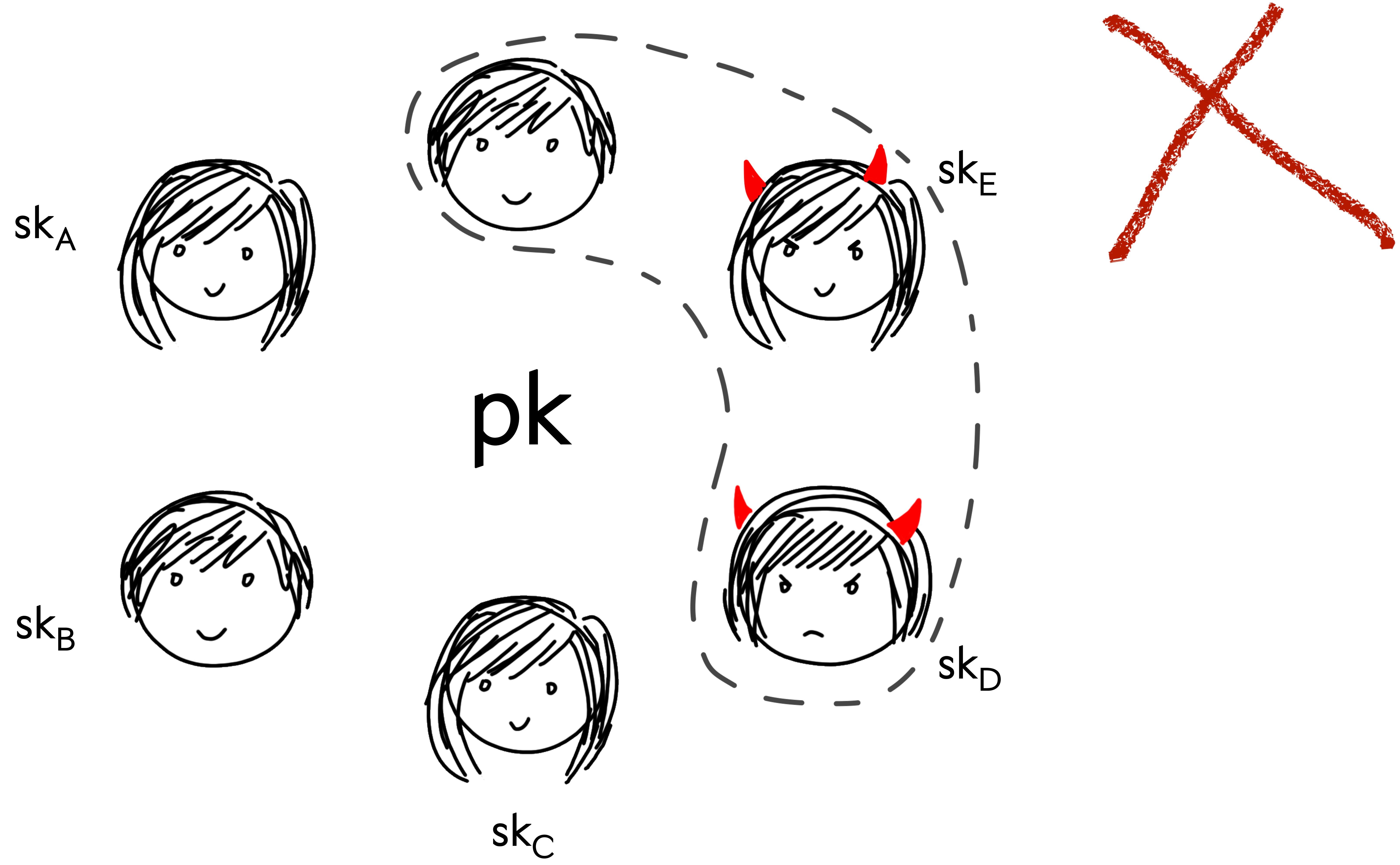
3-of-n Signature Scheme



3-of-n Signature Scheme



3-of-n Signature Scheme



Full Threshold

- Scheme can be instantiated with any *t* <= *n*

- Adversary corrupts up to *t*-1 parties

Notation

Notation

Elliptic curve parameters $G \quad q$

Notation

Elliptic curve parameters G q

Secret values sk k

Notation

Elliptic curve parameters	G	q
Secret values	sk	k
Public values	pk	R

Schnorr Signatures

Now With Fewer Patents!

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

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

Schnorr Signatures

Now With Fewer Patents!

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

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

$$R = k \cdot G$$

Schnorr Signatures

Now With Fewer Patents!

```
$\mathrm{SchnorrSign}(\mathsf{sk}, m)$ :
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(R \| m)$
```

Schnorr Signatures

Now With Fewer Patents!

```
$\mathrm{SchnorrSign}(\mathsf{sk}, m):$
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(R \| m)$
    $s = k - \mathsf{sk} \cdot e$
```

Schnorr Signatures

Now With Fewer Patents!

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

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

$$R = k \cdot G$$

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

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

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

$$\text{output } \sigma$$

Schnorr Signatures

Now With Fewer Patents!

```
$\text{SchnorrSign}(\text{sk}, m):$
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(R \| m)$
    $\boxed{s = k - \text{sk} \cdot e}$
    $\sigma = (s, e)$
    output $\sigma$
```

Linear function of *k*, sk
Threshold friendly w.
linear secret sharing

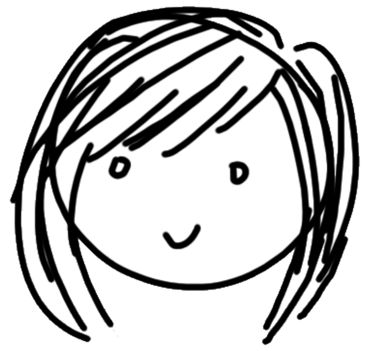
Verification

```
$\mathrm{SchnorrSign}(\mathsf{sk}, m):$
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(R \| m)$
    $s = k - \mathsf{sk} \cdot e$
    $\sigma = (s, e)$
    output $\sigma$
```

```
$\mathrm{SchnorrVerify}(\mathsf{pk}, m, s, e):$
    $\hat{R} = s \cdot G + e \cdot \mathsf{pk}$
    $\hat{e} = H(\hat{R} \| m)$
    output $\hat{e} \overset{?}{=} e$
```

2P-Schnorr

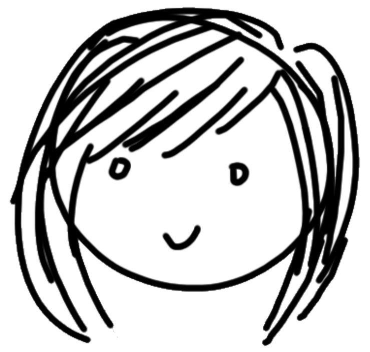
```
$\mathsf{SchnorrSign}(\mathsf{sk}, m)$ :
        $k \leftarrow \mathbb{Z}_q$
        $R = k \cdot G$
        $e = H(R \| m)$
        $s = k - \mathsf{sk} \cdot e$
        $\sigma = (s, e)$
    output $\sigma$
```



$\mathsf{sk}_\mathsf{A} + \mathsf{sk}_\mathsf{B} = \mathsf{sk}$

2P-Schnorr

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

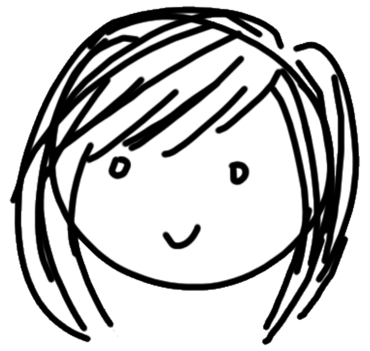


sk_A

sk_B

$k_\mathsf{A} + k_\mathsf{B} = k$

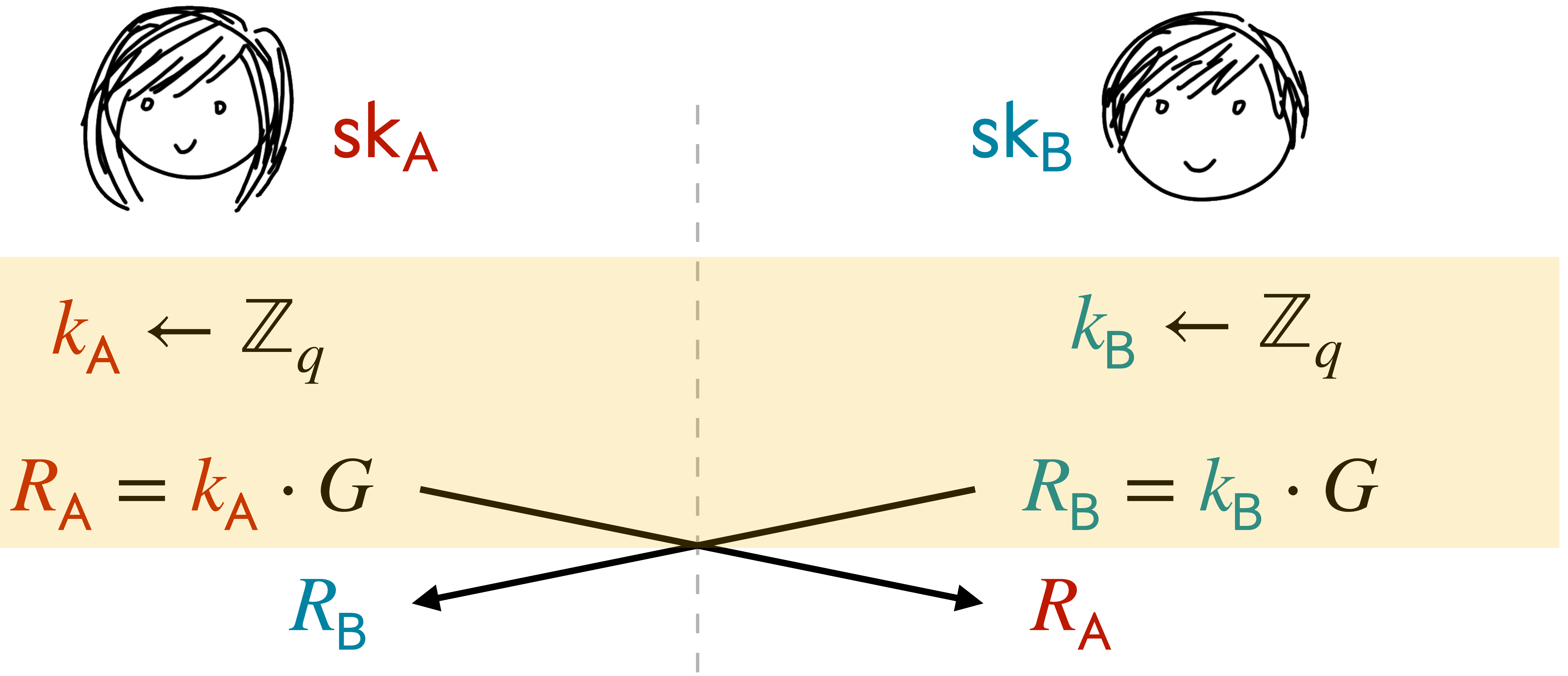
2P-Schnorr



$\mathrm{SchnorrSign}(\mathsf{sk}, m)$:	sk_A	sk_B
$k \leftarrow \mathbb{Z}_q$	$k_\mathsf{A} \leftarrow \mathbb{Z}_q$	$k_\mathsf{B} \leftarrow \mathbb{Z}_q$
$R = k \cdot G$	$R_\mathsf{A} = k_\mathsf{A} \cdot G$	$R_\mathsf{B} = k_\mathsf{B} \cdot G$
$e = H(R \Vert m)$		
$s = k - \mathsf{sk} \cdot e$		
$\sigma = (s, e)$		
output σ		

2P-Schnorr

```
$\mathrm{SchnorrSign}(\mathsf{sk}, m)$ :
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(R \| m)$
    $s = k - \mathsf{sk} \cdot e$
    $\sigma = (s, e)$
    output $\sigma$
```



sk_A

sk_B

$k_\mathsf{A} \leftarrow \mathbb{Z}_q$

$k_\mathsf{B} \leftarrow \mathbb{Z}_q$

$R_\mathsf{A} = k_\mathsf{A} \cdot G$

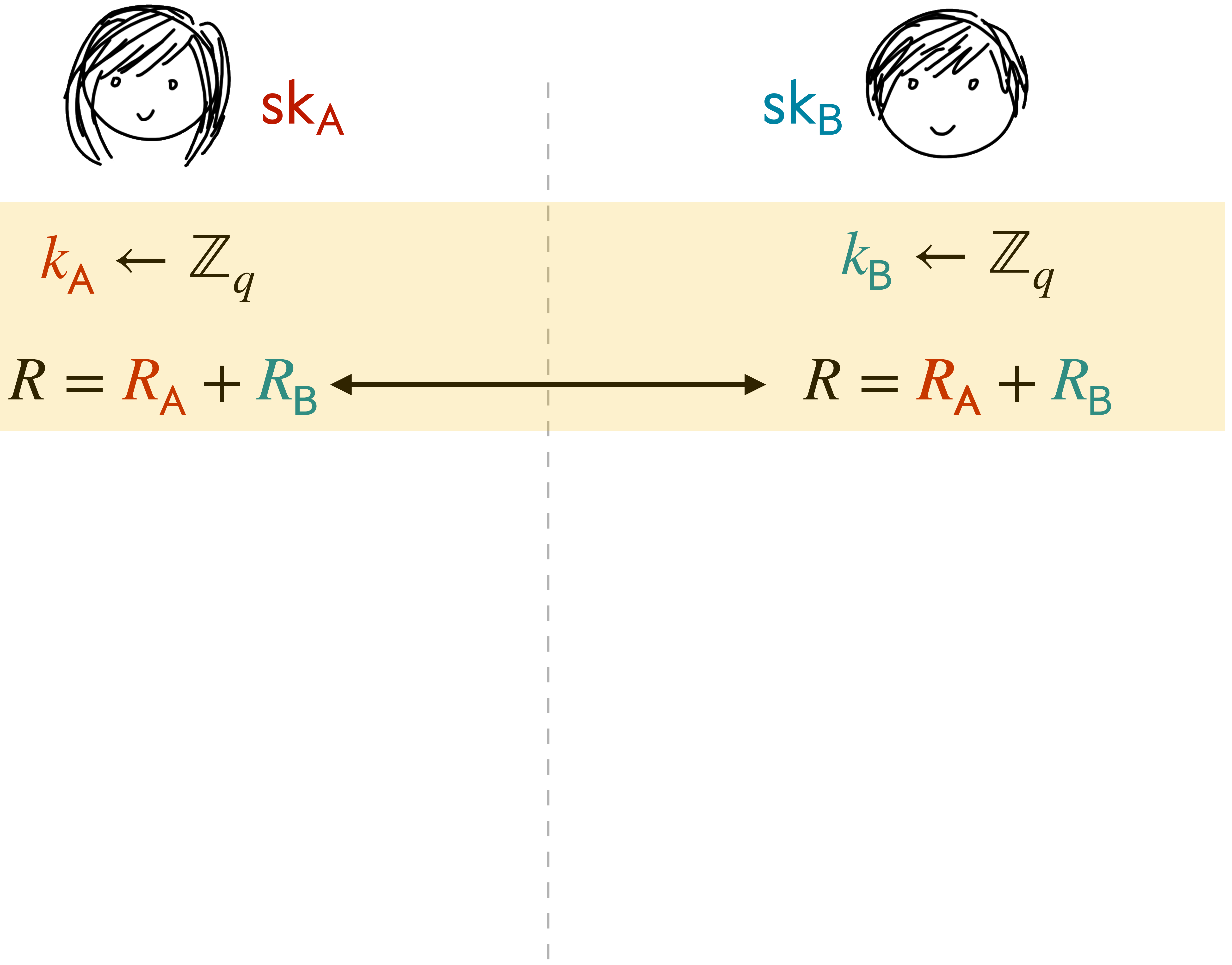
$R_\mathsf{B} = k_\mathsf{B} \cdot G$

R_B

R_A

2P-Schnorr

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



sk_A

sk_B

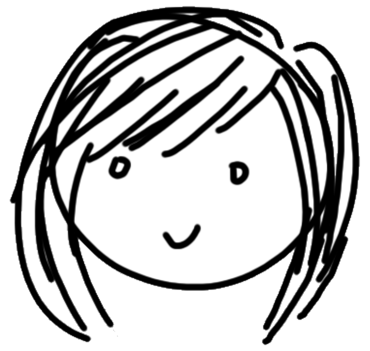
$k_\mathsf{A} \leftarrow \mathbb{Z}_q$

$k_\mathsf{B} \leftarrow \mathbb{Z}_q$

$R = R_\mathsf{A} + R_\mathsf{B} \longleftrightarrow R = R_\mathsf{A} + R_\mathsf{B}$

2P-Schnorr

```
$\mathrm{SchnorrSign}(\mathsf{sk}, m)$ :
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(R \| m)$
    $s = k - \mathsf{sk} \cdot e$
    $\sigma = (s, e)$
  output $\sigma$
```



sk_A

sk_B

$k_\mathsf{A} \leftarrow \mathbb{Z}_q$

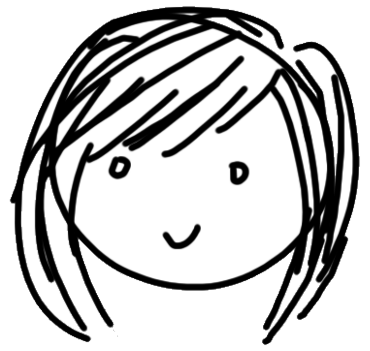
$k_\mathsf{B} \leftarrow \mathbb{Z}_q$

$$R = R_\mathsf{A} + R_\mathsf{B} \longleftrightarrow R = R_\mathsf{A} + R_\mathsf{B}$$

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

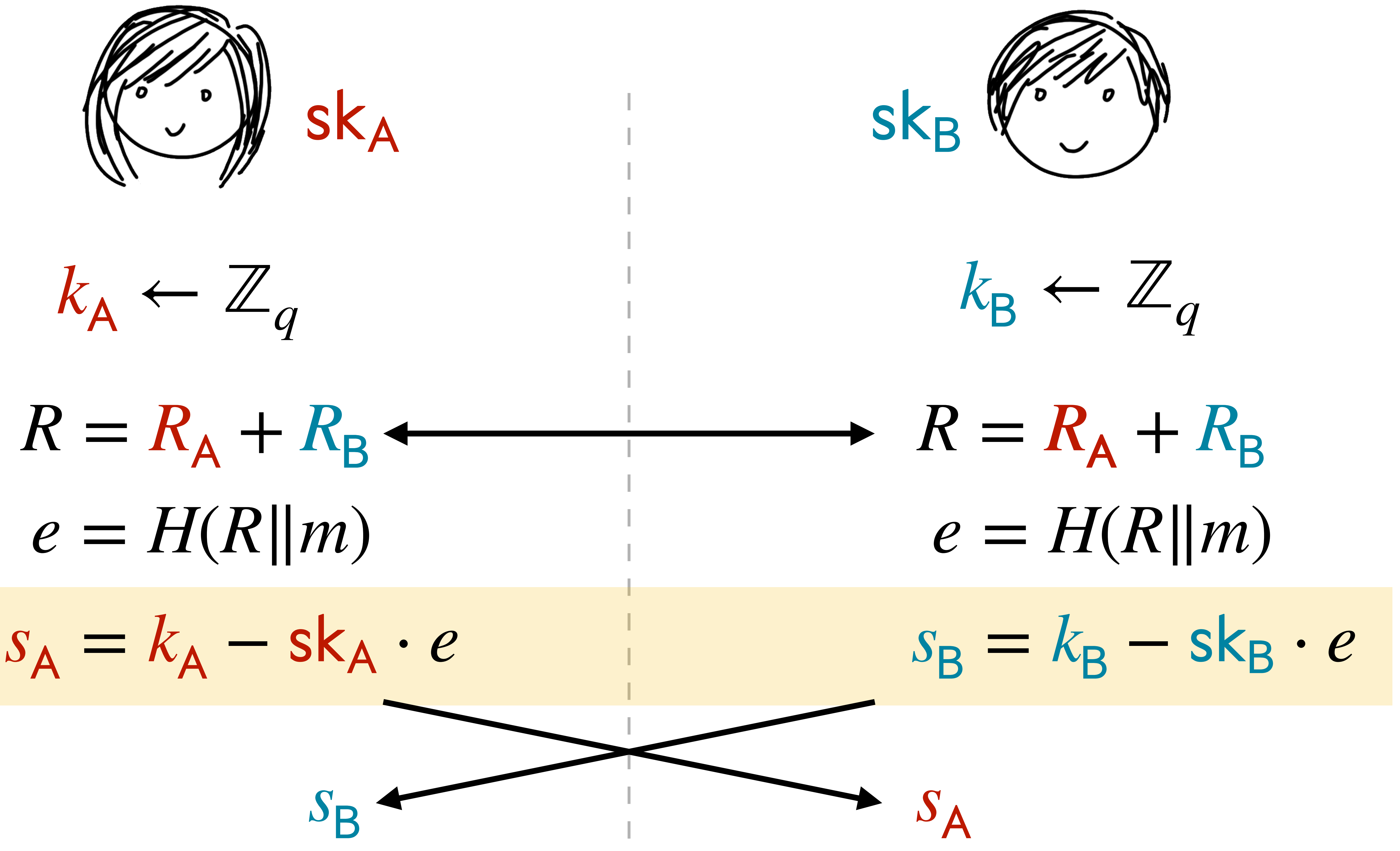
2P-Schnorr

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

 sk_A		sk_B
$k_\mathsf{A} \leftarrow \mathbb{Z}_q$		$k_\mathsf{B} \leftarrow \mathbb{Z}_q$
$R = R_\mathsf{A} + R_\mathsf{B}$	$\longleftrightarrow$	$R = R_\mathsf{A} + R_\mathsf{B}$
$e = H(R \| m)$		$e = H(R \| m)$
$s_\mathsf{A} = k_\mathsf{A} - \mathsf{sk}_\mathsf{A} \cdot e$		$s_\mathsf{B} = k_\mathsf{B} - \mathsf{sk}_\mathsf{B} \cdot e$

2P-Schnorr

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



sk_A

sk_B

$k_A \leftarrow \mathbb{Z}_q$

$k_B \leftarrow \mathbb{Z}_q$

$R = R_A + R_B \longleftrightarrow R = R_A + R_B$

$e = H(R \| m)$

$e = H(R \| m)$

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

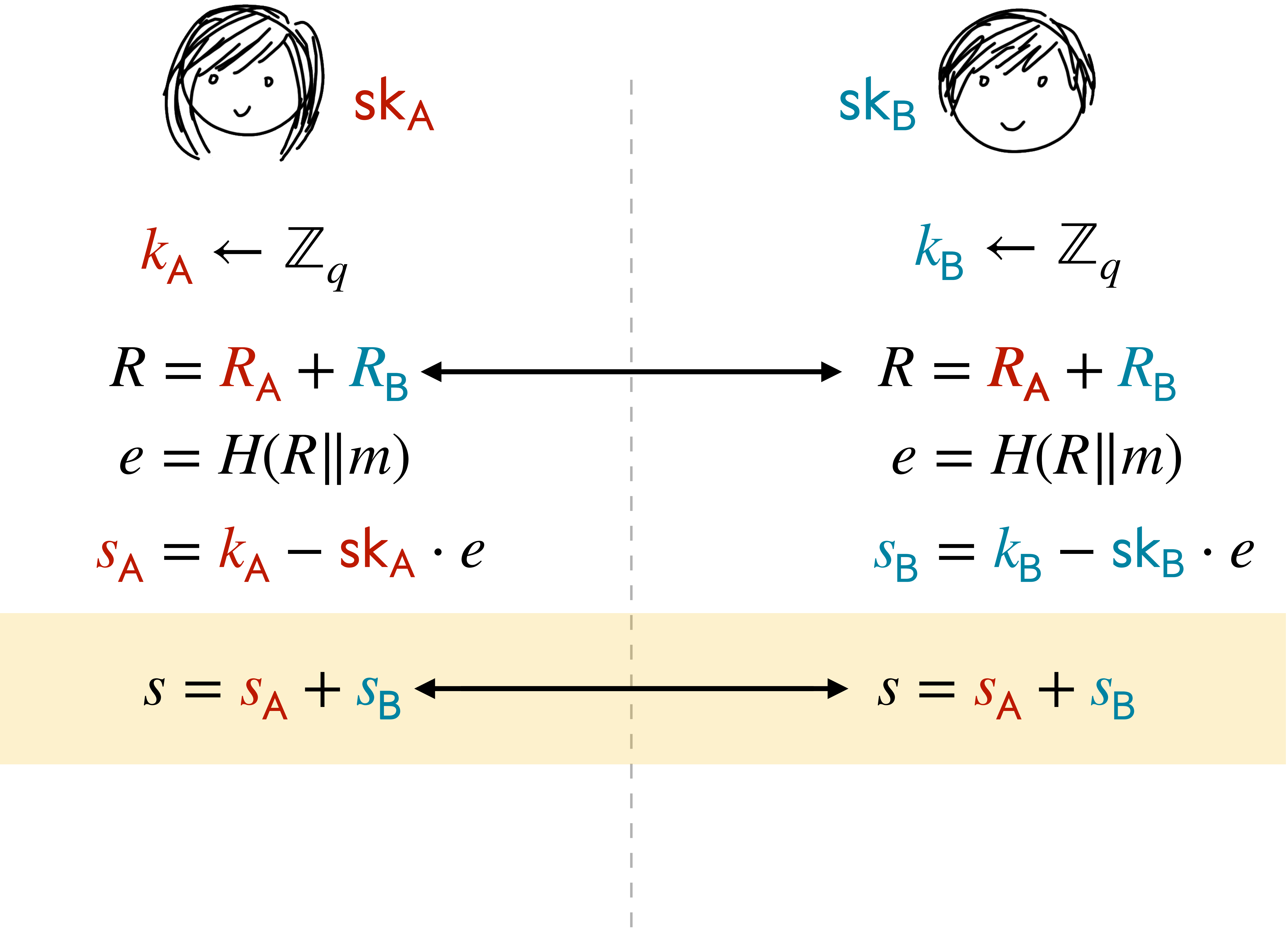
$s_B = k_B - \mathsf{sk}_B \cdot e$

s_B

s_A

2P-Schnorr

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



sk_A | sk_B

$k_\mathsf{A} \leftarrow \mathbb{Z}_q$ | $k_\mathsf{B} \leftarrow \mathbb{Z}_q$

$R = R_\mathsf{A} + R_\mathsf{B} \longleftrightarrow R = R_\mathsf{A} + R_\mathsf{B}$

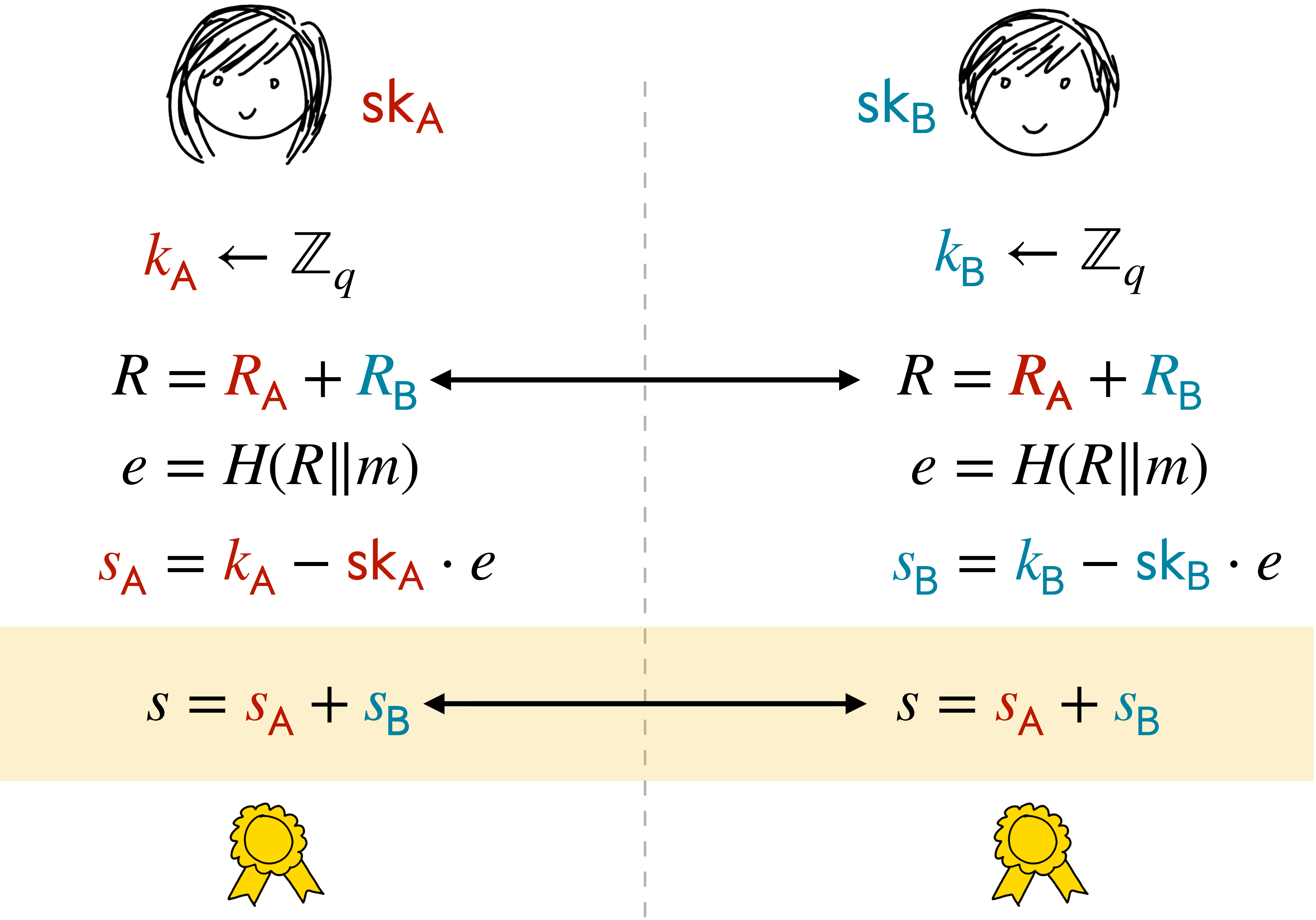
$e = H(R \| m)$ | $e = H(R \| m)$

$s_\mathsf{A} = k_\mathsf{A} - \mathsf{sk}_\mathsf{A} \cdot e$ | $s_\mathsf{B} = k_\mathsf{B} - \mathsf{sk}_\mathsf{B} \cdot e$

$s = s_\mathsf{A} + s_\mathsf{B} \longleftrightarrow s = s_\mathsf{A} + s_\mathsf{B}$

2P-Schnorr

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



sk_A | sk_B

$k_\text{A} \leftarrow \mathbb{Z}_q$ | $k_\text{B} \leftarrow \mathbb{Z}_q$

$R = R_\text{A} + R_\text{B}$ ⟷ $R = R_\text{A} + R_\text{B}$

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

$s_\text{A} = k_\text{A} - \text{sk}_\text{A} \cdot e$ | $s_\text{B} = k_\text{B} - \text{sk}_\text{B} \cdot e$

$s = s_\text{A} + s_\text{B}$ ⟷ $s = s_\text{A} + s_\text{B}$

ECDSA

- **E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm
- Devised by David Kravitz, standardized by NIST
- Widespread adoption across the internet

ECDSA

- **E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm
- Devised by David Kravitz, standardized by NIST
- Widespread adoption across the internet

Schnorr versus ECDSA

```
$\mathrm{SchnorrSign}(\mathsf{sk}, m):$
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(R \| m)$
    $s = k - \mathsf{sk} \cdot e$
    $\sigma = (s, e)$
    output $\sigma$
```

Schnorr versus ECDSA

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

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

$$R = k \cdot G$$

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

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

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

$\text{output } \sigma$

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

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

$$R = k \cdot G$$

Schnorr versus ECDSA

```
$\mathrm{SchnorrSign}(\mathsf{sk}, m):$
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(R \| m)$
    $s = k - \mathsf{sk} \cdot e$
    $\sigma = (s, e)$
    output $\sigma$
```

```
$\mathrm{ECDSASign}(\mathsf{sk}, m):$
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(m)$
```

Schnorr versus ECDSA

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

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

Schnorr versus ECDSA

```
$\mathrm{SchnorrSign}(\mathsf{sk}, m):$
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(R \| m)$
    $s = k - \mathsf{sk} \cdot e$
    $\sigma = (s, e)$
    output $\sigma$
```

The x-coordinate of R

```
$\mathrm{ECDSASign}(\mathsf{sk}, m):$
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(m)$
    $s = \dfrac{e + \mathsf{sk} \cdot r_x}{k}$
```

Schnorr versus ECDSA

```
$\mathrm{SchnorrSign}(\mathsf{sk}, m):$
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(R \| m)$
    $s = k - \mathsf{sk} \cdot e$
    $\sigma = (s, e)$
    output $\sigma$
```

The x-coordinate of R

```
$\mathrm{ECDSASign}(\mathsf{sk}, m):$
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(m)$
    $s = \dfrac{e + \mathsf{sk} \cdot r_x}{k}$
    $\sigma = (s, e)$
    output $\sigma$
```

MP-ECDSA Challenges

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

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

$$R = k \cdot G$$

$$e = H(m)$$

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

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

$$\text{output } \sigma$$

MP-ECDSA Challenges

```
$\mathrm{ECDSASign}(\mathsf{sk}, m):$
    $k \leftarrow \mathbb{Z}_q$
    $R = k \cdot G$
    $e = H(m)$
    $s = \dfrac{e + \mathsf{sk} \cdot r_x}{k}$
    $\sigma = (s, e)$
    output $\sigma$
```

MP-ECDSA Challenges

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

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

$$R = k \cdot G$$

$$e = H(m)$$

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

Modular inverse

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

$$\text{output } \sigma$$

MP-ECDSA Challenges

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

Multiply secrets

Modular inverse

Threshold ECDSA

Threshold ECDSA

- Limited schemes based on Paillier encryption: [MacKenzie Reiter 04], [Gennaro Goldfeder Narayanan 16], [Lindell 17]

Threshold ECDSA

- Limited schemes based on Paillier encryption: [MacKenzie Reiter 04], [Gennaro Goldfeder Narayanan 16], [Lindell 17]
- Practical key generation and efficient signing (full threshold):

Threshold ECDSA

- Limited schemes based on Paillier encryption: [MacKenzie Reiter 04], [Gennaro Goldfeder Narayanan 16], [Lindell 17]
- Practical key generation and efficient signing (full threshold):
  - [Gennaro Goldfeder 18]: Paillier-based

Threshold ECDSA

- Limited schemes based on Paillier encryption: [MacKenzie Reiter 04], [Gennaro Goldfeder Narayanan 16], [Lindell 17]
- Practical key generation and efficient signing (full threshold):
  - [Gennaro Goldfeder 18]: Paillier-based
  - [Lindell Nof Ranellucci 18]: El-Gamal based

Threshold ECDSA

- Limited schemes based on Paillier encryption: [MacKenzie Reiter 04], [Gennaro Goldfeder Narayanan 16], [Lindell 17]
- Practical key generation and efficient signing (full threshold):
  - [Gennaro Goldfeder 18]: Paillier-based
  - [Lindell Nof Ranellucci 18]: El-Gamal based
- **Our work:**

Threshold ECDSA

- Limited schemes based on Paillier encryption: [MacKenzie Reiter 04], [Gennaro Goldfeder Narayanan 16], [Lindell 17]

- Practical key generation and efficient signing (full threshold):
  - [Gennaro Goldfeder 18]: Paillier-based
  - [Lindell Nof Ranellucci 18]: El-Gamal based

- **Our work:**
  - [DKLs18]: 2-of-n ECDSA under native assumptions

Threshold ECDSA

- Limited schemes based on Paillier encryption: [MacKenzie Reiter 04], [Gennaro Goldfeder Narayanan 16], [Lindell 17]
- Practical key generation and efficient signing (full threshold):
  - [Gennaro Goldfeder 18]: Paillier-based
  - [Lindell Nof Ranellucci 18]: El-Gamal based
- **Our work:**
  - [DKLs18]: 2-of-n ECDSA under native assumptions
  - [DKLs19]: Full-Threshold ECDSA under native assumptions

Our Approach

Our Approach

- 2-party multipliers: **Oblivious Transfer** in **ECDSA curve**

Our Approach

- 2-party multipliers: **Oblivious Transfer** in **ECDSA curve**
  - **Pros**:

Our Approach

- 2-party multipliers: **Oblivious Transfer** in **ECDSA curve**
  - **Pros**:
    - With **OT Extension** (no extra assumptions) just a **few milliseconds**

Our Approach

- 2-party multipliers: **Oblivious Transfer** in **ECDSA curve**
  - **Pros**:
    - With **OT Extension** (no extra assumptions) just a **few milliseconds**
    - **Native assumptions** (**CDH** in the same curve)

Our Approach

- 2-party multipliers: **Oblivious Transfer** in **ECDSA curve**
  - **Pros**:
    - With **OT Extension** (no extra assumptions) just a **few milliseconds**
    - **Native assumptions** (**CDH** in the same curve)
    - **NEW!** [K-Magri-Orlandi-Shlomovits] Proactive-friendly

Our Approach

- 2-party multipliers: **Oblivious Transfer** in **ECDSA curve**
  - **Pros**:
    - With **OT Extension** (no extra assumptions) just a **few milliseconds**
    - **Native assumptions** (**CDH** in the same curve)
    - **NEW!** [K-Magri-Orlandi-Shlomovits] Proactive-friendly
  - **Con:** Higher bandwidth (**100s of KB/party**)

Our Approach

Our Approach

- OT-MUL secure up to choice of inputs

Our Approach

- OT-MUL secure up to choice of inputs

- **Light consistency check (unique to our protocol):**

Our Approach

- OT-MUL secure up to choice of inputs

- **Light consistency check (unique to our protocol):**
  - Verify shares in the exponent before reveal

Our Approach

- OT-MUL secure up to choice of inputs

- **Light consistency check (unique to our protocol)**:
  - Verify shares in the exponent before reveal
  - Costs **5 exponentiations+curve points**/party

Our Approach

- OT-MUL secure up to choice of inputs

- **Light consistency check (unique to our protocol)**:
  - Verify shares in the exponent before reveal
  - Costs **5 exponentiations+curve points**/party
  - Subverting checks implies solving **CDH** in ECDSA curve

Tradeoffs

Tradeoffs

- Our protocol **avoids expensive zero-knowledge proofs** and **assumptions foreign to ECDSA** itself, required by other works in the area

Tradeoffs

- Our protocol **avoids expensive zero-knowledge proofs** and **assumptions foreign to ECDSA** itself, required by other works in the area

- Using OT-MUL is very light on computation, but more demanding of bandwidth than alternative approaches; we argue this is not an issue for many applications

Tradeoffs

- Our protocol **avoids expensive zero-knowledge proofs** and **assumptions foreign to ECDSA** itself, required by other works in the area

- Using OT-MUL is very light on computation, but more demanding of bandwidth than alternative approaches; we argue this is not an issue for many applications

- Our wall clock times (even WAN) are an **order of magnitude** better than the next best concurrent work

Our Model

Our Model

- **Universal Composability** [Canetti ’01] (static adv., global RO)

Our Model

- **Universal Composability** [Canetti ’01] (static adv., global RO)
- **Functionality (trusted third party *emulated* by protocol)**:

Our Model

- **Universal Composability** [Canetti ’01] (static adv., global RO)
- **Functionality (trusted third party *emulated* by protocol)**:
  - Store secret key

Our Model

- **Universal Composability** [Canetti ’01] (static adv., global RO)
- **Functionality (trusted third party *emulated* by protocol)**:
  - Store secret key
  - Compute ECDSA signature when enough parties ask

Our Model

- **Universal Composability** [Canetti ’01] (static adv., global RO)
- **Functionality (trusted third party *emulated* by protocol)**:
  - Store secret key
  - Compute ECDSA signature when enough parties ask
- **Assumption**: CDH is hard in the ECDSA curve

Our Model

- **Universal Composability** [Canetti ’01] (static adv., global RO)
- **Functionality (trusted third party *emulated* by protocol)**:
  - Store secret key
  - Compute ECDSA signature when enough parties ask
- **Assumption**: CDH is hard in the ECDSA curve
- **Network**: Synchronous, broadcast

Our Model

- **Universal Composability** [Canetti ’01] (static adv., global RO)
- **Functionality (trusted third party *emulated* by protocol)**:
  - Store secret key
  - Compute ECDSA signature when enough parties ask
- **Assumption**: CDH is hard in the ECDSA curve
- **Network**: Synchronous, broadcast
- Security with abort

Our Approach

- **Setup:** MUL setup, VSS for [sk]
- **Signing:**
  1. Get candidate shares [k], [$1/k$], and $R=k\cdot G$
  2. Compute [sk/k] = MUL([$1/k$], [sk])
  3. Check relations in exponent
  4. Reconstruct *sig* = [$1/k$]$\cdot H(m)$+[sk/k]

Setup

Setup

- **Fully distributed**

Setup

- **Fully distributed**
- **MUL setup**: Pairwise among parties (Base OTs for OTe)

Setup

- **Fully distributed**
- **MUL setup**: Pairwise among parties (Base OTs for OTe)
- **Key generation**: (Pedersen-style)

Setup

- **Fully distributed**
- **MUL setup**: Pairwise among parties (Base OTs for OTe)
- **Key generation**: (Pedersen-style)
  - Every party Shamir-shares a random secret

Setup

- **Fully distributed**
- **MUL setup**: Pairwise among parties (Base OTs for OTe)
- **Key generation**: (Pedersen-style)
  - Every party Shamir-shares a random secret
  - Secret key is sum of parties’ contributions

Setup

- **Fully distributed**
- **MUL setup**: Pairwise among parties (Base OTs for OTe)
- **Key generation**: (Pedersen-style)
  - Every party Shamir-shares a random secret
  - Secret key is sum of parties’ contributions
  - Verify in the exponent that parties’ shares are on the same polynomial

Our Approach

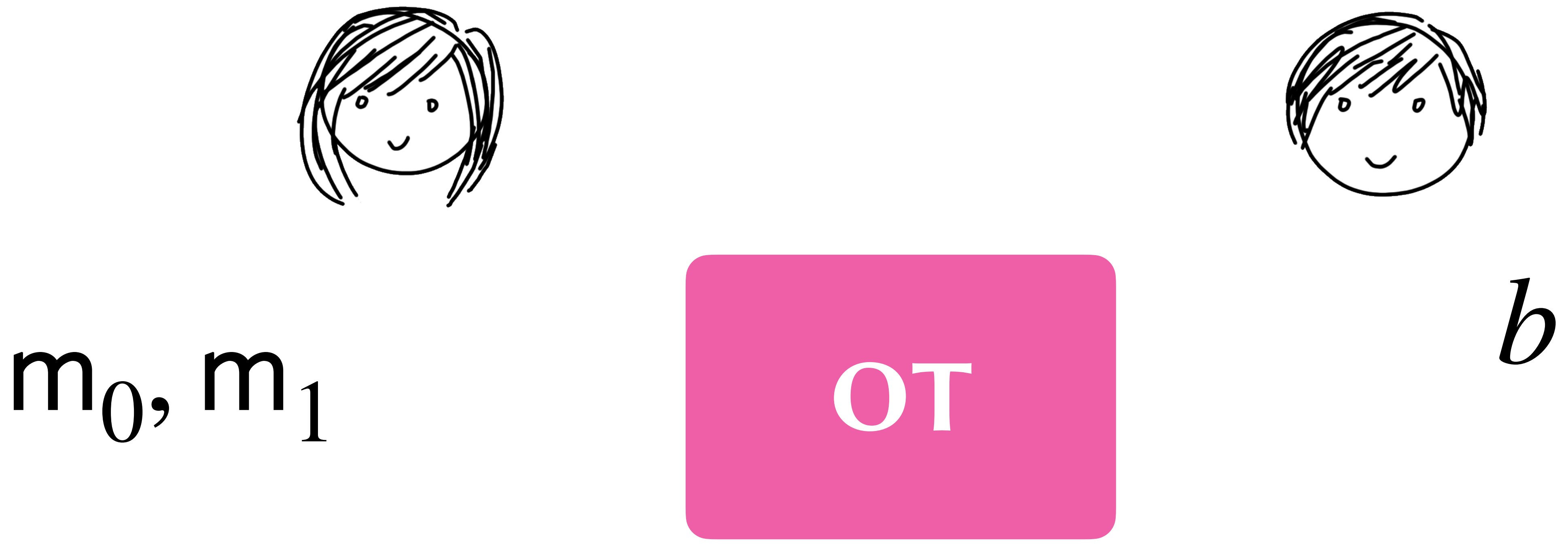
- **Setup**: MUL setup, VSS for [sk]
- **Signing:**

1. Get candidate shares [k], [$1/k$], and $R=k\cdot G$
2. Compute [sk/k] = MUL([$1/k$], [sk])
3. Check relations in exponent
4. Reconstruct *sig* = [$1/k$]$\cdot H(m)$+[sk/k]

Obtaining Candidate Shares

- **Building Block**: Two party MUL with full security [DKLs18]

Oblivious Transfer

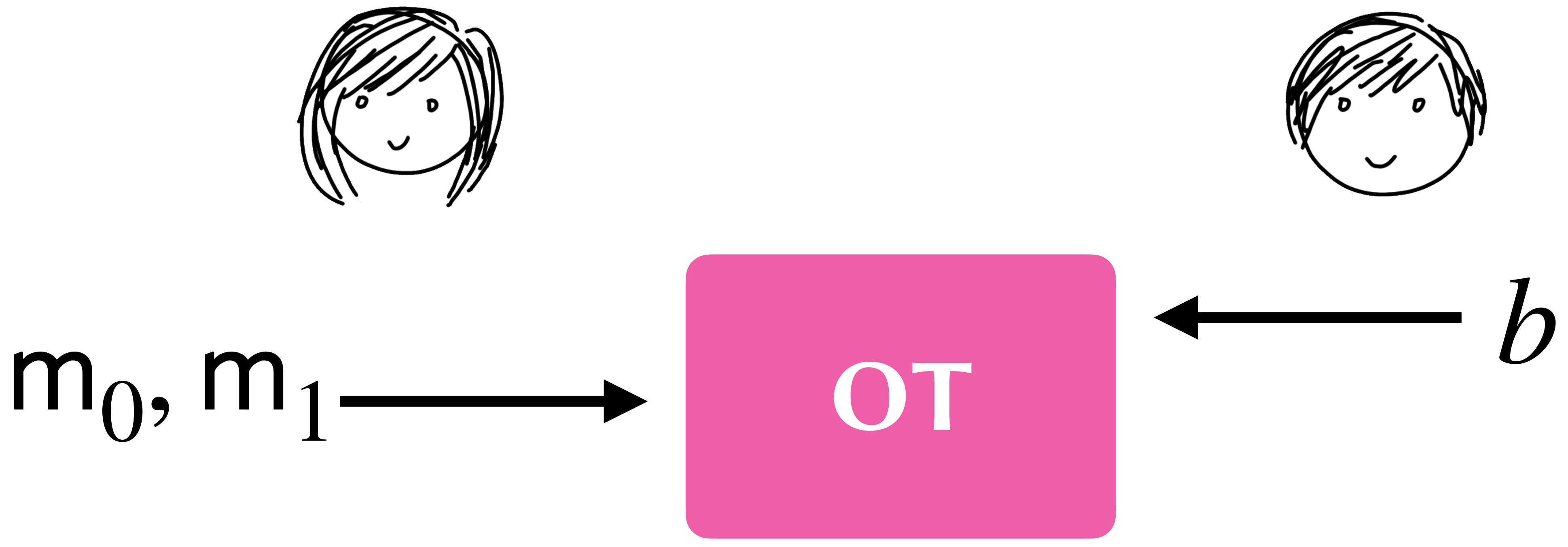


m_0, m_1

OT

b

Oblivious Transfer

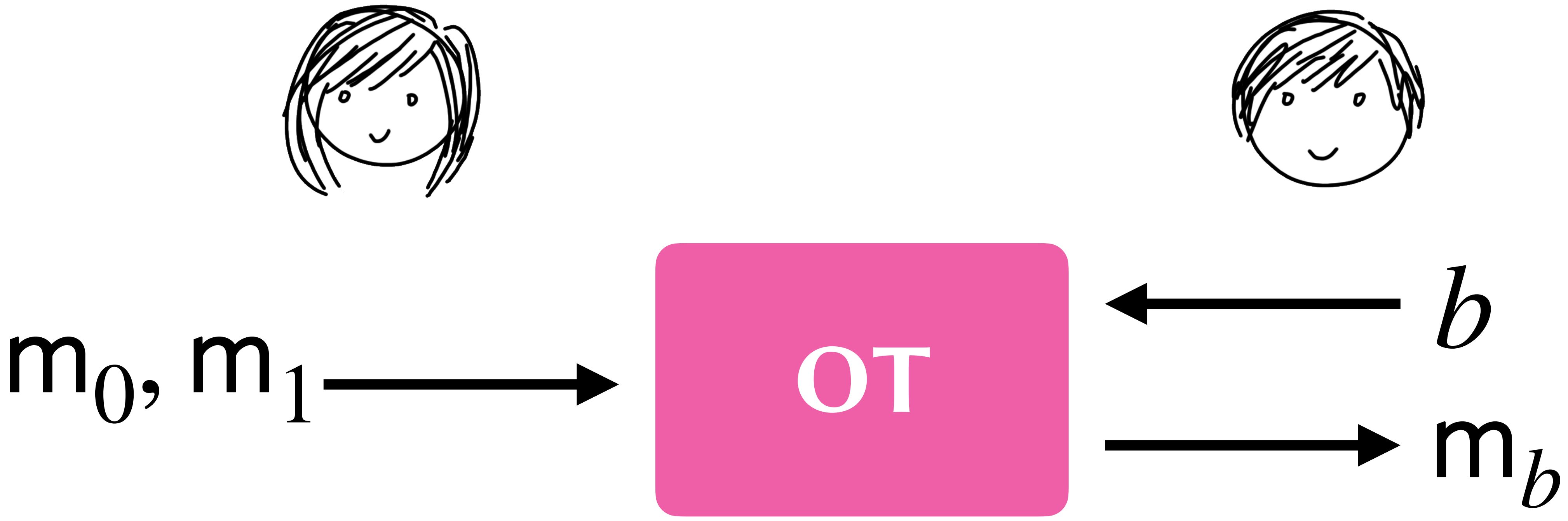


m_0, m_1

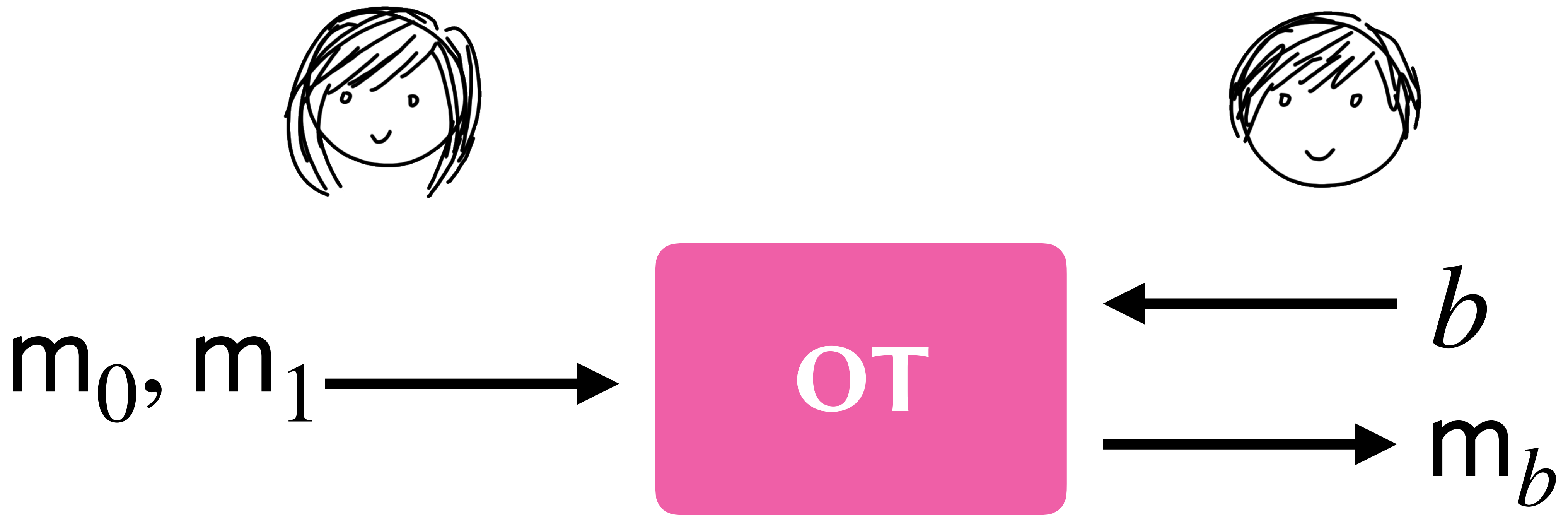
OT

b

Oblivious Transfer



Oblivious Transfer



m_0, m_1

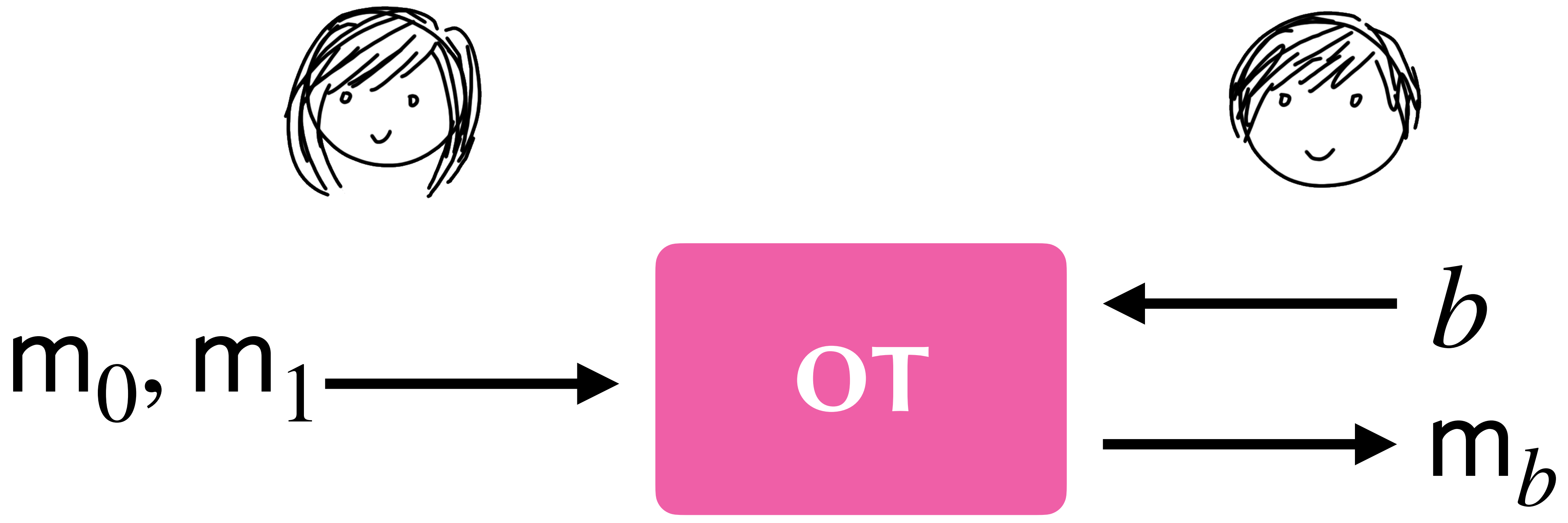
OT

b

m_b

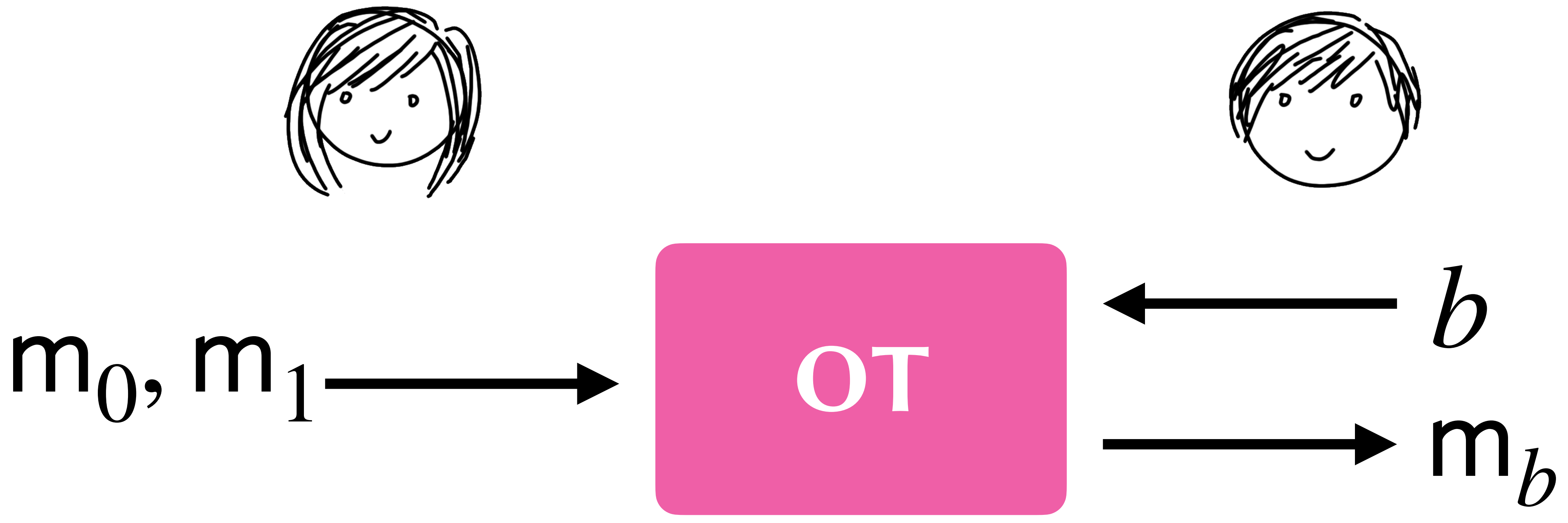
- **Instantiation:** “Verified” Simplest Oblivious Transfer [Chou&Orlandi15]

Oblivious Transfer



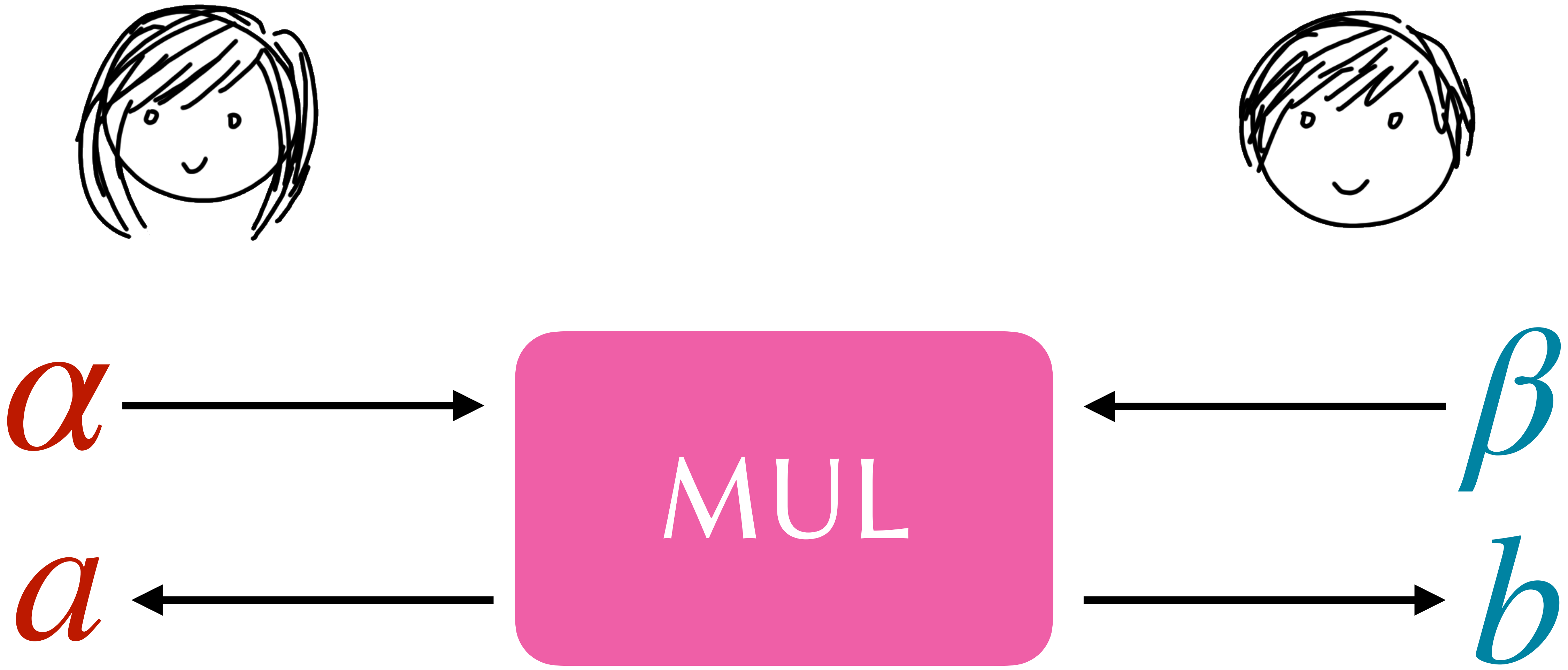
- **Instantiation:** “Verified” Simplest Oblivious Transfer [Chou&Orlandi15]
- UC-secure (RO model) assuming CDH in the same curve as ECDSA

Oblivious Transfer



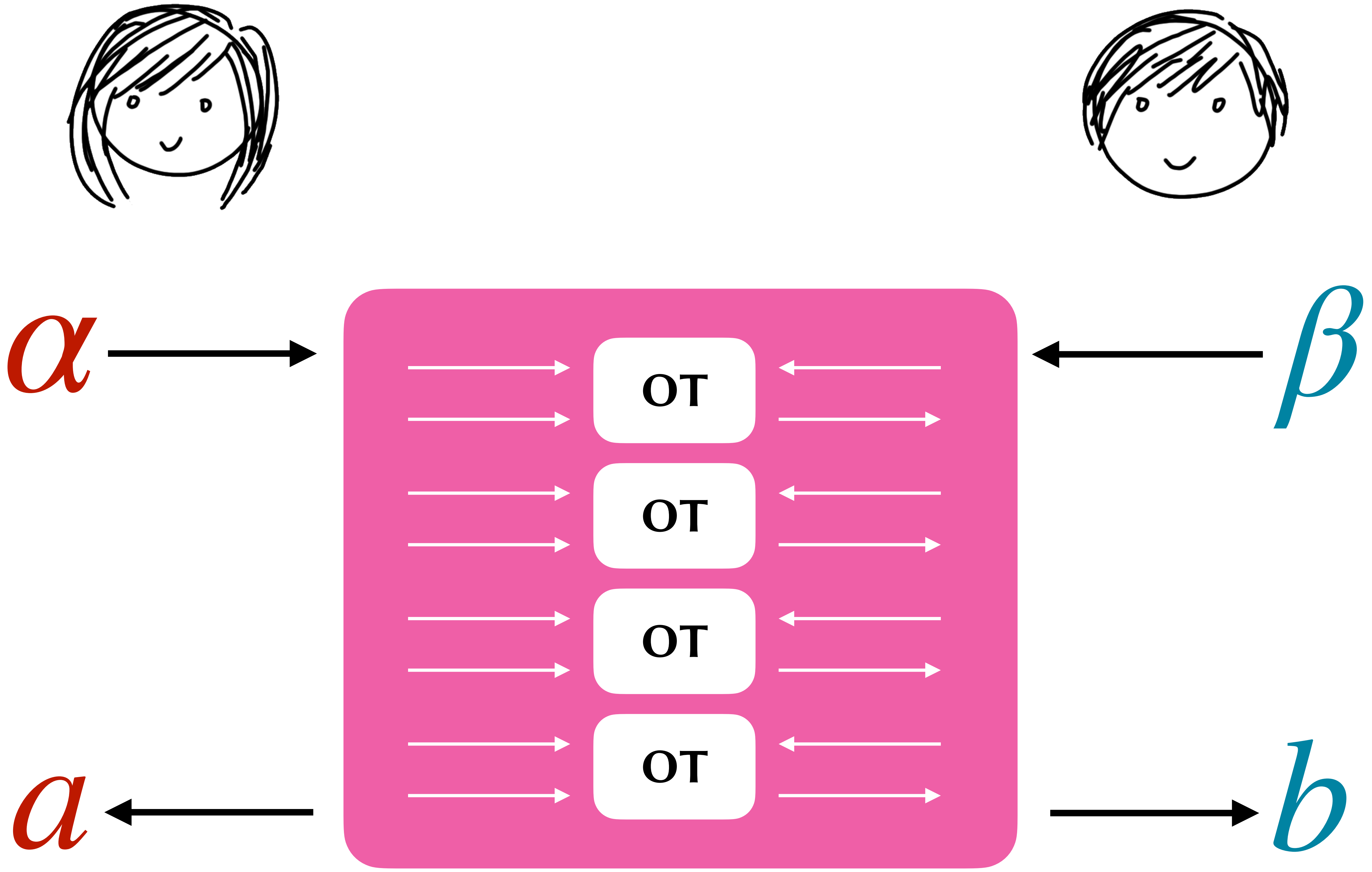
- **Instantiation:** “Verified” Simplest Oblivious Transfer [Chou&Orlandi15]
- UC-secure (RO model) assuming CDH in the same curve as ECDSA
- **OT Extension**: [Keller Orsini Scholl ’15] only needs RO

2P-MUL

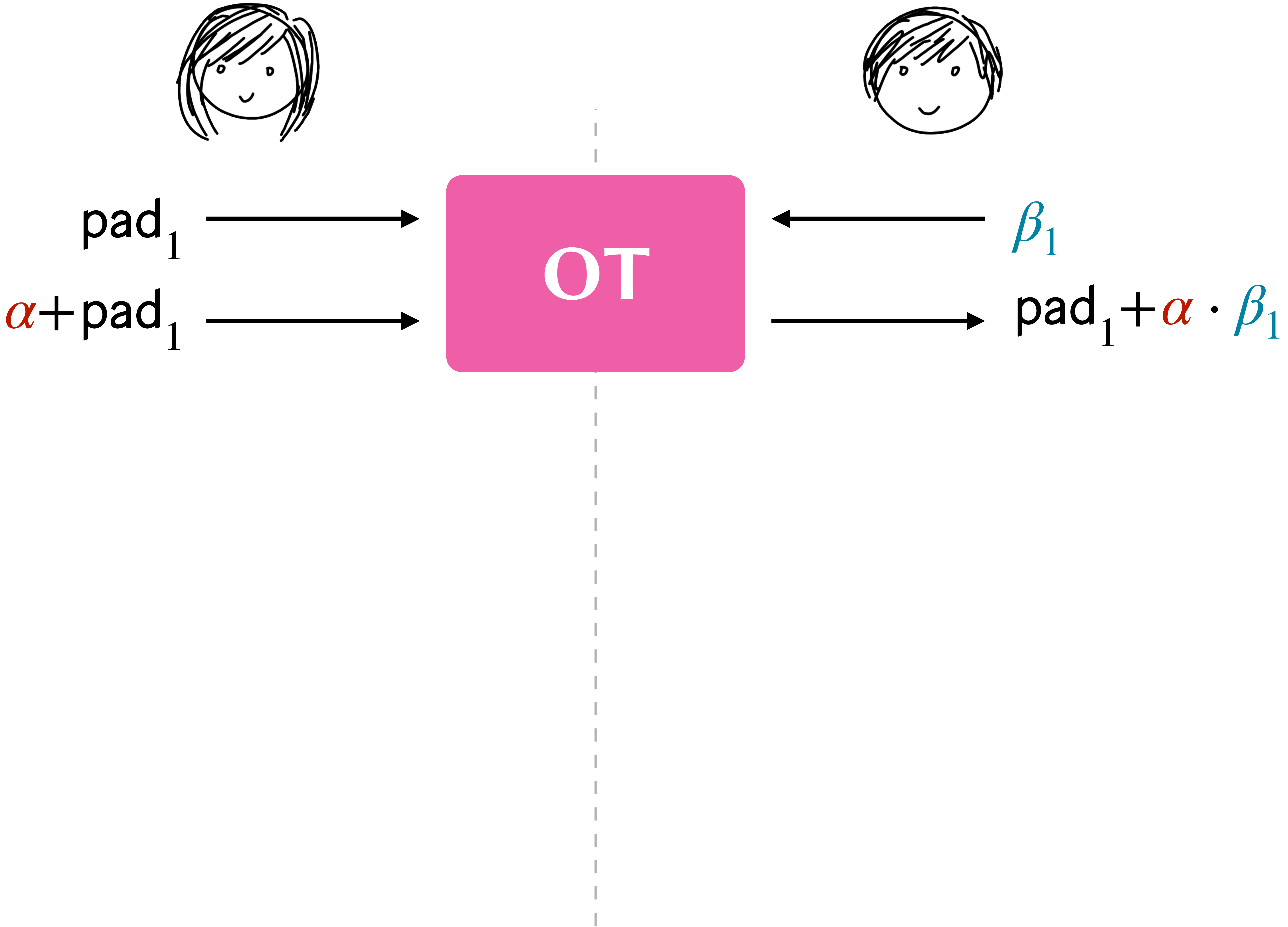


$$a + b = \alpha \cdot \beta$$

2P-MUL from OT [Gil99]



2P-MUL from OT [Gil99]



pad_1

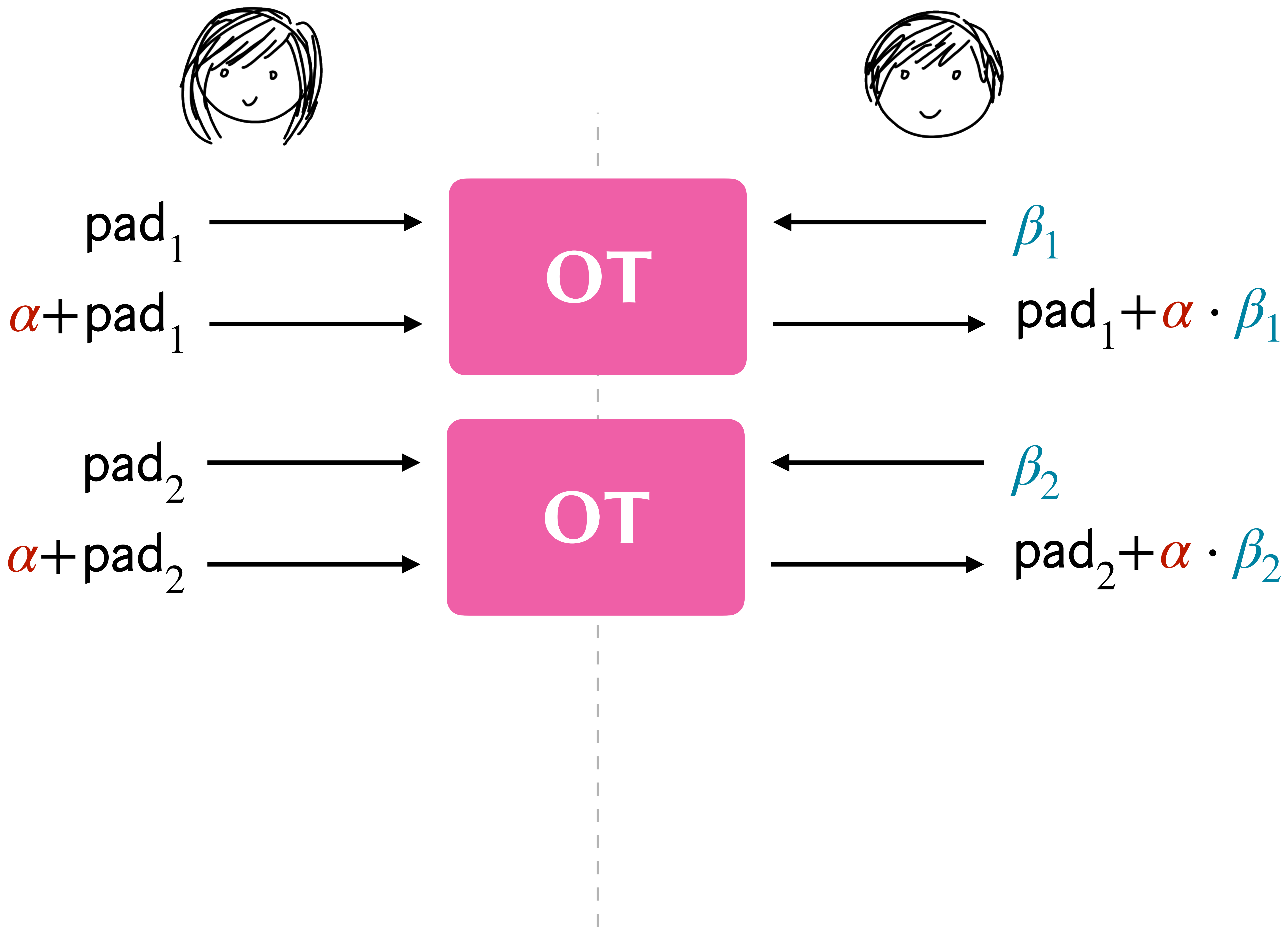
$\alpha + \mathsf{pad}_1$

OT

β_1

$\mathsf{pad}_1 + \alpha \cdot \beta_1$

2P-MUL from OT [Gil99]



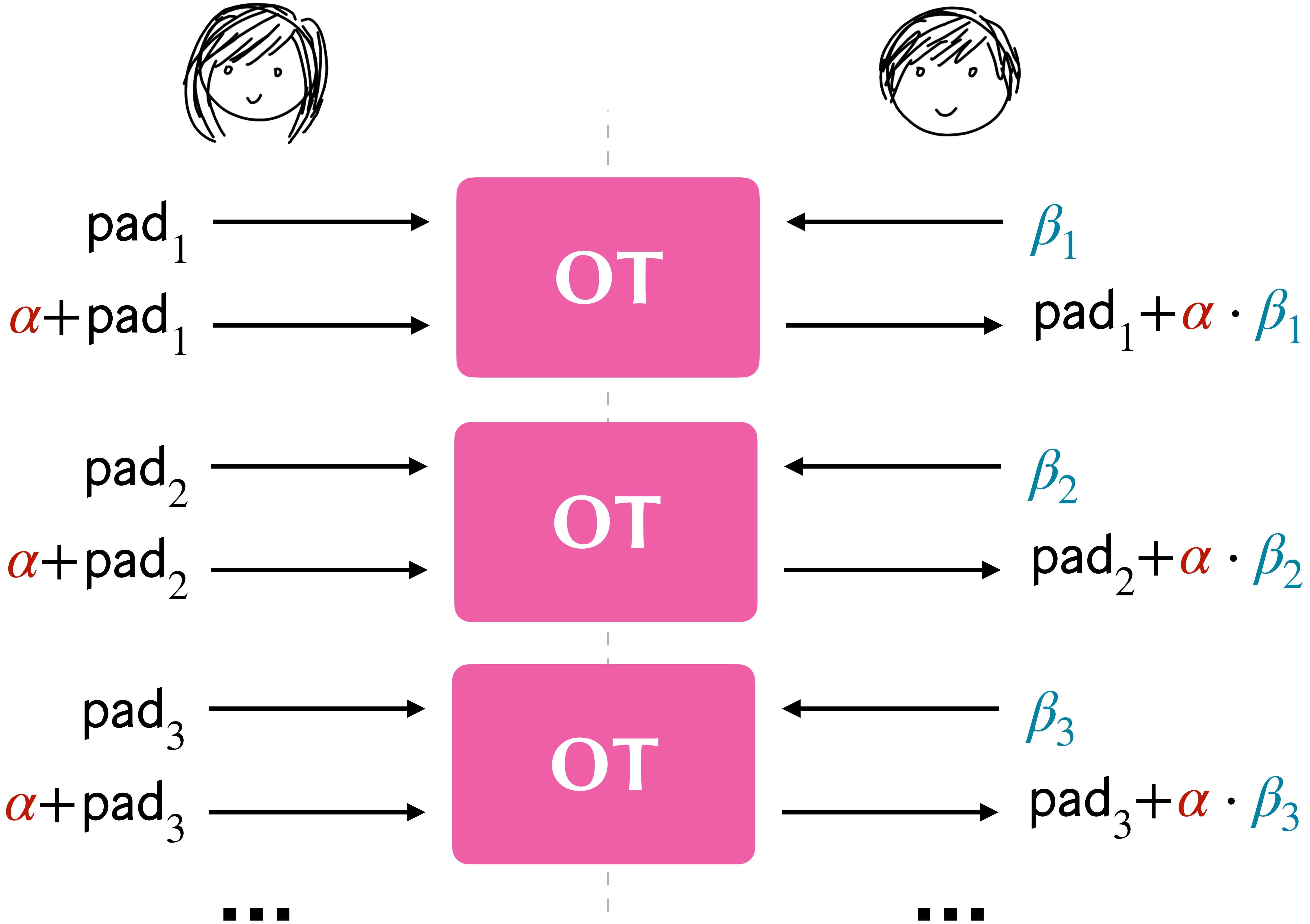
pad_1 → OT ← β_1

$\alpha+\text{pad}_1$ → OT → $\text{pad}_1+\alpha\cdot\beta_1$

pad_2 → OT ← β_2

$\alpha+\text{pad}_2$ → OT → $\text{pad}_2+\alpha\cdot\beta_2$

2P-MUL from OT [Gil99]



pad_1 → OT ← β_1

$\alpha + \text{pad}_1$ → OT → $\text{pad}_1 + \alpha \cdot \beta_1$

pad_2 → OT ← β_2

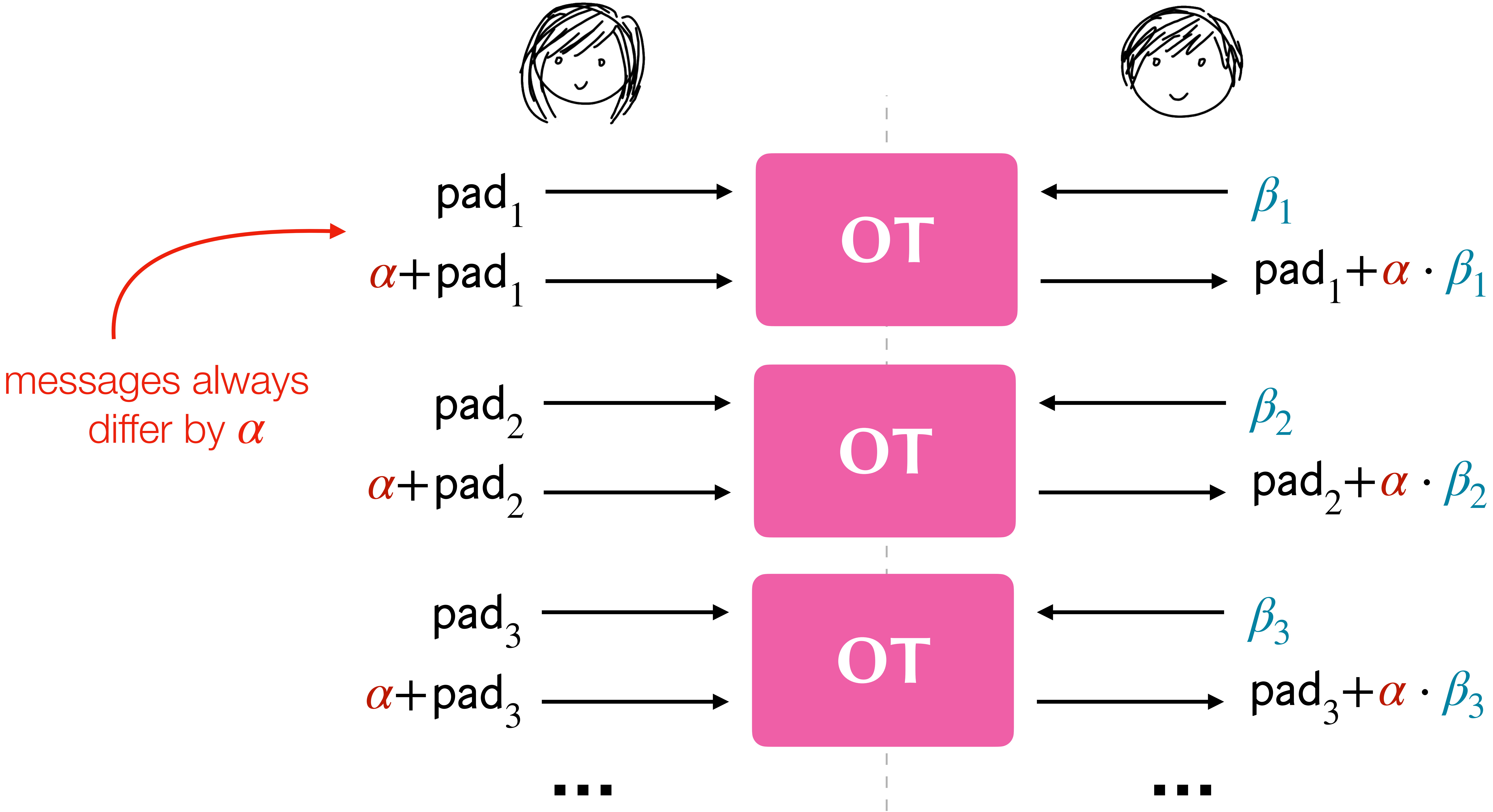
$\alpha + \text{pad}_2$ → OT → $\text{pad}_2 + \alpha \cdot \beta_2$

pad_3 → OT ← β_3

$\alpha + \text{pad}_3$ → OT → $\text{pad}_3 + \alpha \cdot \beta_3$

... ...

2P-MUL from OT [Gil99]



pad_1 → OT ← β_1

$\alpha + \mathsf{pad}_1$ → OT → $\mathsf{pad}_1 + \alpha \cdot \beta_1$

pad_2 → OT ← β_2

$\alpha + \mathsf{pad}_2$ → OT → $\mathsf{pad}_2 + \alpha \cdot \beta_2$

pad_3 → OT ← β_3

$\alpha + \mathsf{pad}_3$ → OT → $\mathsf{pad}_3 + \alpha \cdot \beta_3$

... ...

messages always differ by α

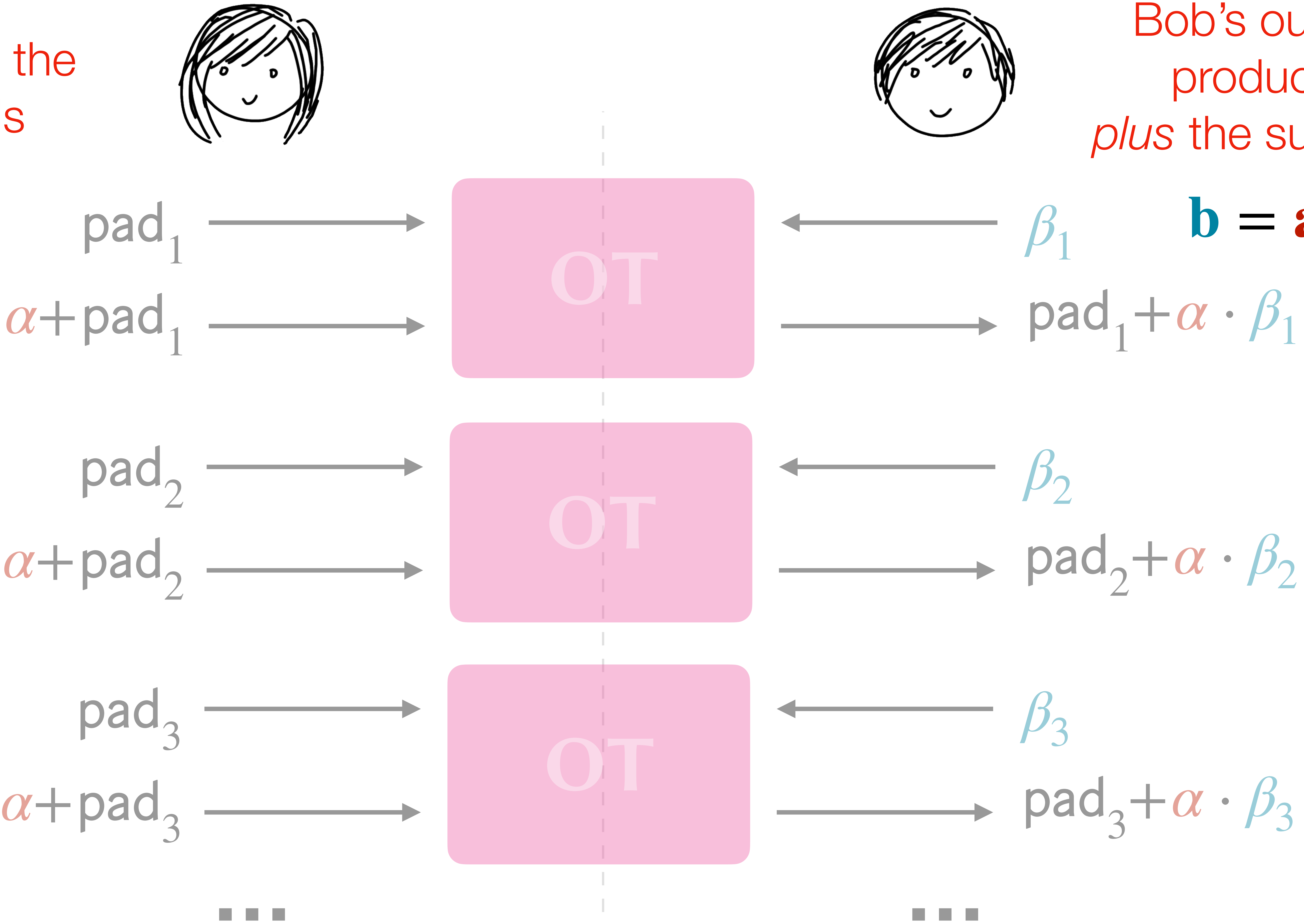
2P-MUL from OT [Gil99]

Alice's output ***a*** is the sum of the pads

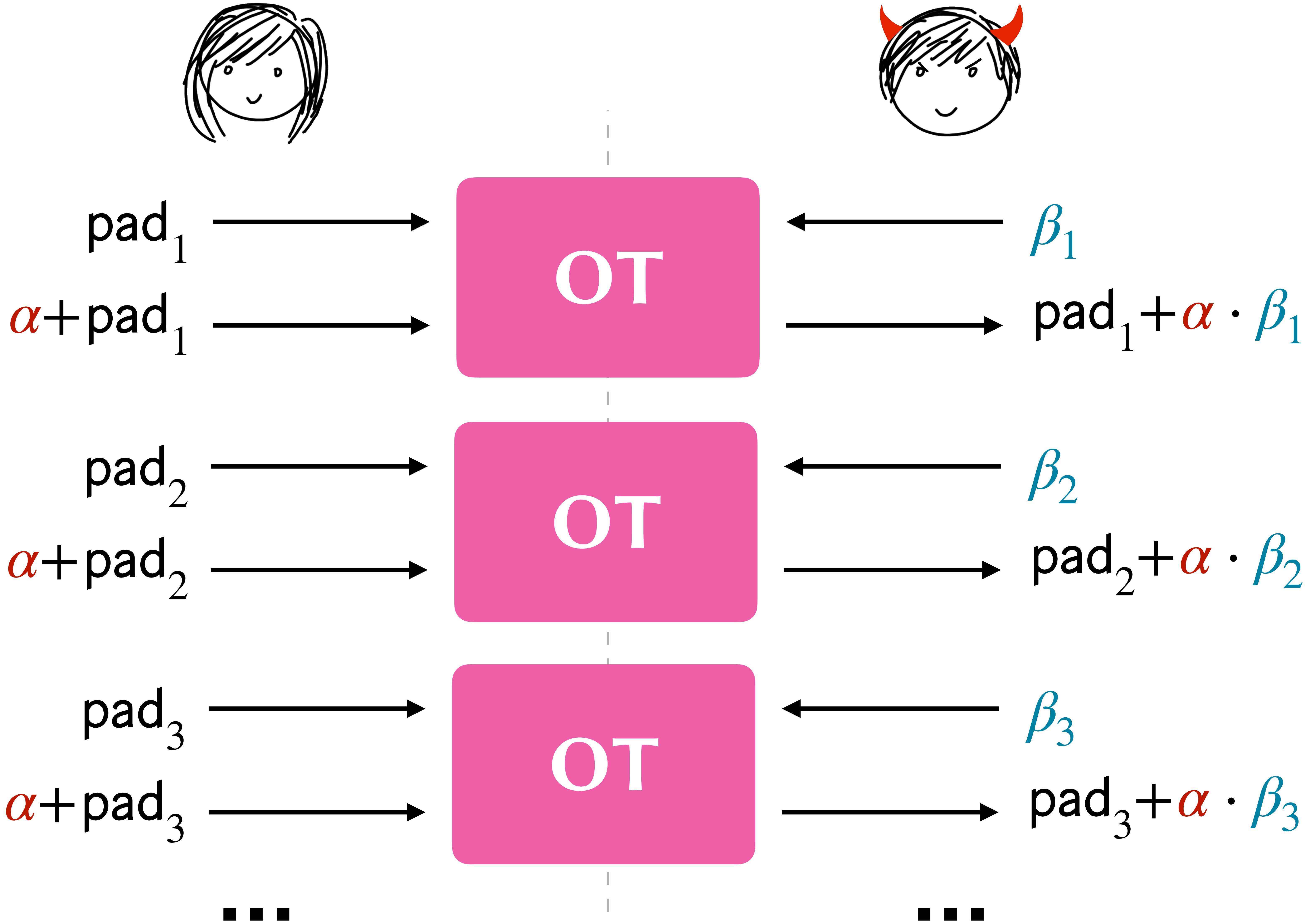
$$\mathbf{a} = \left(\sum \text{pad}_i\right)$$

Bob's output ***b*** is the product of inputs *plus* the sum of the pads

$$\mathbf{b} = \mathbf{a} + \alpha \cdot \beta$$



Malicious Bob: Secure OT



pad_1 → OT ← β_1

$\alpha+\text{pad}_1$ → OT → $\text{pad}_1+\alpha \cdot \beta_1$

pad_2 → OT ← β_2

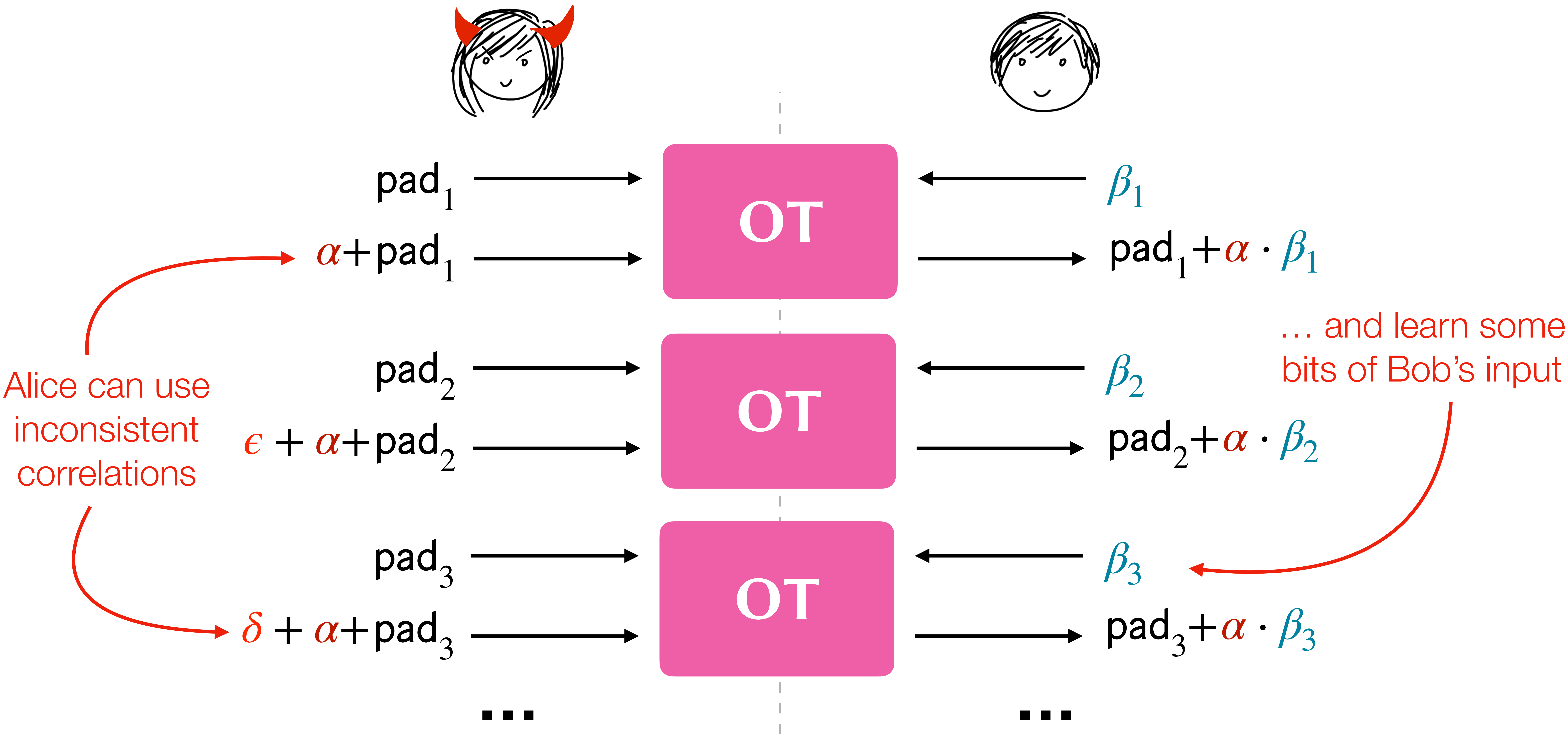
$\alpha+\text{pad}_2$ → OT → $\text{pad}_2+\alpha \cdot \beta_2$

pad_3 → OT ← β_3

$\alpha+\text{pad}_3$ → OT → $\text{pad}_3+\alpha \cdot \beta_3$

... ...

(M)Alice: Selective Failure



pad_1 → OT ← β_1

$\alpha + \text{pad}_1$ → OT → $\text{pad}_1 + \alpha \cdot \beta_1$

pad_2 → OT ← β_2

$\epsilon + \alpha + \text{pad}_2$ → OT → $\text{pad}_2 + \alpha \cdot \beta_2$

pad_3 → OT ← β_3

$\delta + \alpha + \text{pad}_3$ → OT → $\text{pad}_3 + \alpha \cdot \beta_3$

... ...

Alice can use inconsistent correlations

… and learn some bits of Bob's input

(M)Alice: Checks and Encoding

(M)Alice: Checks and Encoding

1. High-entropy encoding of Bob’s input ensures Alice must correctly guess many bits to learn anything

(M)Alice: Checks and Encoding

1. High-entropy encoding of Bob’s input ensures Alice must correctly guess many bits to learn anything

- Any ***s*** bits taken in isolation look ≈uniform

(M)Alice: Checks and Encoding

1. High-entropy encoding of Bob’s input ensures Alice must correctly guess many bits to learn anything

Based on [IN96]

- Any ***s*** bits taken in isolation look ≈uniform

(M)Alice: Checks and Encoding

1. High-entropy encoding of Bob's input ensures Alice must correctly guess many bits to learn anything

Based on [IN96]

- Any ***s*** bits taken in isolation look ≈uniform

2. Check system: each additional cheat halves probability of 'getting away'

(M)Alice: Checks and Encoding

1. High-entropy encoding of Bob's input ensures Alice must correctly guess many bits to learn anything

Based on [IN96]

- Any ***s*** bits taken in isolation look ≈uniform

2. Check system: each additional cheat halves probability of 'getting away'

- 2^{-s} chance of learning more than *s* bits

Obtaining Candidate Shares

Obtaining Candidate Shares

- **Building Block**: Two party MUL with full security [DKLs18]

Obtaining Candidate Shares

- **Building Block**: Two party MUL with full security [DKLs18]
- **One approach** (implemented): Evaluate along binary tree

Obtaining Candidate Shares

- **Building Block**: Two party MUL with full security [DKLs18]

- **One approach** (implemented): Evaluate along binary tree

  - Each party starts with multiplicative shares of *k* and *1*/*k*

Obtaining Candidate Shares

- **Building Block**: Two party MUL with full security [DKLs18]
- **One approach** (implemented): Evaluate along binary tree
  - Each party starts with multiplicative shares of *k* and *1*/*k*
  - Multiplicative to additive shares: log(*t*)+c rounds

Obtaining Candidate Shares

- **Building Block**: Two party MUL with full security [DKLs18]
- **One approach** (implemented): Evaluate along binary tree
  - Each party starts with multiplicative shares of k and $1/k$
  - Multiplicative to additive shares: log(t)+c rounds
- **Alternative**: [Bar-Ilan&Beaver '89] approach yields constant round protocol (work in progress)

Our Approach

- **Setup**: MUL setup, VSS for [sk]
- **Signing:**
    1. Get candidate shares $[k]$, $[1/k]$, and $R=k\cdot G$
    2. Compute $[\mathrm{sk}/k] = \mathrm{MUL}([1/k], [\mathrm{sk}])$
    3. Check relations in exponent
    4. Reconstruct $sig = [1/k]\cdot H(m)+[\mathrm{sk}/k]$

Our Approach

- **Setup**: MUL setup, VSS for [sk]
- **Signing:**

1. Get candidate shares [k], [$1/k$], and $R=k\cdot G$
2. Compute [sk/k] = MUL([$1/k$], [sk]) => GMW
3. Check relations in exponent
4. Reconstruct sig = [$1/k$]$\cdot H(m)$+[sk/k]

Our Approach

- **Setup**: MUL setup, VSS for [sk]
- **Signing**:

1. Get candidate shares [*k*], [*1/k*], and *R*=*k*·*G*
2. Compute [sk/*k*] = MUL([*1/k*], [sk])
3. Check relations in exponent
4. Reconstruct *sig* = [*1/k*]·*H*(*m*)+[sk/*k*]

Check in Exponent

- There are **three** relations that have to be verified to guarantee that inputs to multipliers were correct

$$[k] \qquad \left[\frac{1}{k}\right] \qquad \left[\frac{\mathsf{sk}}{k}\right]$$

Check in Exponent

$$[k] \qquad \left[\frac{1}{k}\right] \qquad \left[\frac{\mathsf{sk}}{k}\right]$$

Check in Exponent

$$[k] \qquad \left[\frac{1}{k}\right] \qquad \left[\frac{\mathsf{sk}}{k}\right]$$

- **Technique:** Each equation is verified in the exponent, using ‘auxiliary’ information that’s already available

Check in Exponent

$$[k] \qquad \left[\frac{1}{k}\right] \qquad \left[\frac{\mathsf{sk}}{k}\right]$$

- **Technique**: Each equation is verified in the exponent, using ‘auxiliary’ information that’s already available
- **Cost**: 5 exponentiations, 5 group elements per party independent of party count, and no ZK proofs

Check in Exponent

Check in Exponent

- **Task:** verify relationship between [*k*] and [1/*k*]

Check in Exponent

- **Task:** verify relationship between [*k*] and [1/*k*]

- **Idea**: verify $\left[\frac{1}{k}\right][k] = 1$ by verifying $\left[\frac{1}{k}\right][k] \cdot G = G$

Check in Exponent

Attempt at a solution:

Check in Exponent

Attempt at a solution:

Public R

Check in Exponent

Attempt at a solution:

Public R

Broadcast $\Gamma_i = \left[\frac{1}{k}\right]_i \cdot R$

Check in Exponent

Attempt at a solution:

Public

$$R$$

Broadcast

$$\Gamma_i = \left[\frac{1}{k}\right]_i \cdot R$$

Verify

$$\sum_{i \in [n]} \Gamma_i = G$$

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public

$$R = \overset{\downarrow}{k_A}\,\overset{\downarrow}{k_h} \cdot G$$

Broadcast

$$\Gamma_i = \left[\frac{1}{k_{\mathrm{A}}}\frac{1}{k_h}\right]_i \cdot R$$

Verify

$$\sum_{i\in[n]} \Gamma_i = G$$

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public

$$R = k_A k_h \cdot G$$

Broadcast

$$\Gamma_i = \left[\left(\frac{1}{k_{\mathrm{A}}}+\epsilon\right)\frac{1}{k_h}\right]_i \cdot R$$

Verify

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public

$$R = k_A k_h \cdot G$$

Broadcast

$$\Gamma_i = \left[\left(\frac{1}{k_A} + \epsilon \right) \frac{1}{k_h} \right]_i \cdot R$$

Verify

$$\sum_{i \in [n]} \Gamma_i = G + \epsilon k_A \cdot G$$

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public

$$R = \overset{\downarrow}{k_A}\overset{\downarrow}{k_h} \cdot G$$

Broadcast

$$\Gamma_i = \left[\left(\frac{1}{k_\mathrm{A}} + \epsilon\right)\frac{1}{k_h}\right]_i \cdot R$$

Verify

$$\sum_{i \in [n]} \Gamma_i = G + \underline{\epsilon k_A \cdot G}$$

Easy for Adv. to offset

Idea: Randomize Target

Idea: Randomize Target

- Currently we expect $\sum \Gamma_i$ to hit a fixed target *G*

Idea: Randomize Target

- Currently we expect $\sum \Gamma_i$ to hit a fixed target *G*

- **Idea**: randomize the multiplication so target is unpredictable

Idea: Randomize Target

- Currently we expect $\sum \Gamma_i$ to hit a fixed target *G*

- **Idea**: randomize the multiplication so target is unpredictable

- Compute $\left[\frac{\phi}{k}\right]$ instead of $\left[\frac{1}{k}\right]$

Idea: Randomize Target

- Currently we expect $\sum \Gamma_i$ to hit a fixed target *G*
- **Idea**: randomize the multiplication so target is unpredictable
- Compute $\left[\frac{\phi}{k}\right]$ instead of $\left[\frac{1}{k}\right]$
- Reveal ϕ only after *every* other value is committed

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public

$$R = k_A k_h \cdot G$$

Broadcast

$$\Gamma_i = \left[\frac{1}{k_A} \frac{1}{k_h} \right]_i \cdot R$$

Check in Exponent

Attempt at a solution:

Public

$$R = \overset{\substack{\text{Adversary's contribution}\\ \downarrow}}{k_A}\,\overset{\substack{\text{Honest Party's contribution}\\ \downarrow}}{k_h} \cdot G$$

Broadcast

$$\Gamma_i = \left[\frac{\phi_A}{k_{\mathrm{A}}}\frac{\phi_h}{k_h}\right]_i \cdot R$$

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public

$$R = \overset{\downarrow}{k_A}\overset{\downarrow}{k_h} \cdot G$$

Broadcast

$$\Gamma_i = \left[\frac{\phi_A}{k_{\mathrm{A}}}\ \frac{\phi_h}{k_h}\right]_i \cdot R$$

Verify

$$\sum_{i \in [n]} \Gamma_i = \phi_A \phi_h \cdot G$$

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public $$R = k_A k_h \cdot G$$

Broadcast $$\Gamma_i = \left[\frac{\phi_A}{k_A} \frac{\phi_h}{k_h} \right]_i \cdot R$$

Verify $$\sum_{i \in [n]} \Gamma_i = \Phi$$

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public

$$R = \overset{\downarrow}{k_A}\overset{\downarrow}{k_h} \cdot G$$

Broadcast

$$\Gamma_i = \left[\left(\frac{\phi_A}{k_A} + \epsilon\right)\frac{\phi_h}{k_h}\right]_i \cdot R$$

Verify

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public

$$R = k_A k_h \cdot G$$

Broadcast

$$\Gamma_i = \left[\left(\frac{\phi_A}{k_{\mathrm{A}}} + \epsilon \right) \frac{\phi_h}{k_h} \right]_i \cdot R$$

Verify

$$\sum_{i \in [n]} \Gamma_i = \Phi + \epsilon \phi_h k_A \cdot G$$

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public

$$R = k_A k_h \cdot G$$

Broadcast

$$\Gamma_i = \left[\left(\frac{\phi_A}{k_{\mathrm{A}}} + \epsilon \right) \frac{\phi_h}{k_h} \right]_i \cdot R$$

Verify

$$\sum_{i \in [n]} \Gamma_i = \Phi + \epsilon \underline{\phi_h} k_A \cdot G$$

Completely unpredictable

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public

$$R = k_A k_h \cdot G$$

Broadcast

$$\Gamma_i = \left[\left(\frac{\phi_A}{k_{\mathrm{A}}} + \epsilon\right)\frac{\phi_h}{k_h}\right]_i \cdot R$$

Verify

$$\sum_{i \in [n]} \Gamma'_i = \Phi' + \epsilon \underline{\mathsf{sk}_h k_h} \cdot G$$

Hard to compute assuming CDH

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public $R = k_A k_h \cdot G$

Broadcast $\Gamma_i = \left[\left(\frac{\phi_A}{k_{\mathrm{A}}} + \epsilon\right)\frac{\phi_h}{k_h}\right]_i \cdot R$

Verify $\sum_{i \in [n]} \Gamma'_i = \Phi' + \epsilon \underline{\mathrm{sk}_h k_h} \cdot G$

Hard to compute assuming CDH

(Given $\mathrm{sk}_h G, k_h G$ compute $\mathrm{sk}_h k_h G$)

Check in Exponent

There are **two** relations that have to be verified

$$[k] \cdot \left[\frac{1}{k}\right] \overset{?}{=} 1$$

$$[\mathsf{sk}] \cdot \left[\frac{1}{k}\right] \overset{?}{=} \left[\frac{\mathsf{sk}}{k}\right]$$

Check in Exponent

There are **two** relations that have to be verified

$$R \quad [k] \cdot \left[\frac{1}{k}\right] \stackrel{?}{=} 1$$

$$[\mathrm{sk}] \cdot \left[\frac{1}{k}\right] \stackrel{?}{=} \left[\frac{\mathrm{sk}}{k}\right]$$

Check in Exponent

There are **two** relations that have to be verified

$$R \quad [k] \cdot \left[\frac{1}{k}\right] \overset{?}{=} 1$$

$$[\mathsf{sk}] \cdot \left[\frac{1}{k}\right] \overset{?}{=} \left[\frac{\mathsf{sk}}{k}\right] \quad \mathsf{pk}$$

Check in Exponent

There are **two** relations that have to be verified

R

$$[k] \cdot \left[\frac{1}{k}\right] \overset{?}{=} 1$$

pk

$$[\mathsf{sk}] \cdot \left[\frac{1}{k}\right] \overset{?}{=} \left[\frac{\mathsf{sk}}{k}\right]$$

R, pk

Conditioned on
correct [sk]

Our Approach

- **Setup**: MUL setup, VSS for [sk]
- **Signing:**

1. Get candidate shares [k], [$1/k$], and $R=k\cdot G$
2. Compute [sk/k] = MUL([$1/k$], [sk])
3. Check relations in exponent
4. Reconstruct *sig* = [$1/k$]$\cdot H(m)$+[sk/k]

Our Approach

- **Setup**: MUL setup, VSS for [sk]
- **Signing**:

1. Get candidate shares [k], [$1/k$], and $R=k\cdot G$
2. Compute [sk/k] = MUL([$1/k$], [sk])
3. Check relations in exponent
4. Reconstruct sig = [$1/k$]$\cdot H(m)$+[sk/k]

Broadcast linear combination of shares

Our Approach

- **Setup**: MUL setup, VSS for [sk]
- **Signing:**

1. Get candidate shares [k], [$1/k$], and $R=k\cdot G$
2. Compute [sk/k] = MUL([$1/k$], [sk])
3. Check relations in exponent
4. Reconstruct *sig* = [$1/k$]$\cdot H(m)$+[sk/k]

Independent of message being signed: ECDSA-specific correlated randomness allowing one ‘online’ round

Our Approach

- **Setup**: MUL setup, VSS for [sk]

- **Signing:**

1. Get candidate shares [k], [$1/k$], and $R=k\cdot G$
2. Compute [sk/k] = MUL([$1/k$], [sk])
3. Check relations in exponent
4. Reconstruct sig = [$1/k$]$\cdot H(m)$+[sk/k]

Independent of message being signed: ECDSA-specific correlated randomness allowing one 'online' round

We report "from scratch" efficiency

Dominant Costs

(All costs for 256-bit elliptic curves)

Setup	
Signing	

Dominant Costs

(All costs for 256-bit elliptic curves)

	Rounds
Setup	
Signing	

Dominant Costs

(All costs for 256-bit elliptic curves)

	Rounds	Public Key
Setup		
Signing		

Dominant Costs

(All costs for 256-bit elliptic curves)

	Rounds	Public Key	Bandwidth
Setup			
Signing			

Dominant Costs

(All costs for 256-bit elliptic curves)

	Rounds	Public Key	Bandwidth
Setup			
Signing			

Dominant Costs

(All costs for 256-bit elliptic curves)

	Rounds	Public Key	Bandwidth
Setup	5		
Signing			

Dominant Costs

(All costs for 256-bit elliptic curves)

	Rounds	Public Key	Bandwidth
Setup	5	520*n*	
Signing			

Dominant Costs

(All costs for 256-bit elliptic curves)

	Rounds	Public Key	Bandwidth
Setup	5	520n	21n KB
Signing			

Dominant Costs

(All costs for 256-bit elliptic curves)

	Rounds	Public Key	Bandwidth
Setup	5	520*n*	21*n* KB
Signing	log(*t*)+6		

Dominant Costs

(All costs for 256-bit elliptic curves)

	Rounds	Public Key	Bandwidth
Setup	5	520*n*	21*n* KB
Signing	log(*t*)+6	5	

Dominant Costs

(All costs for 256-bit elliptic curves)

	Rounds	Public Key	Bandwidth
Setup	5	520*n*	21*n* KB
Signing	log(*t*)+6	5	<100*t* KB

Dominant Costs

(All costs for 256-bit elliptic curves)

	Rounds	Public Key	Bandwidth
Setup	5	520n	21n KB
Signing	log(t)+6	5	<100t KB

Journal version (in progress): **8 round signing**

(à la [Bar-Ilan Beaver 89])

Benchmarks

Benchmarks

- Implementation in **Rust**

Benchmarks

- Implementation in **Rust**
- Ran benchmarks on Google Cloud

Benchmarks

- Implementation in **Rust**
- Ran benchmarks on Google Cloud
- One node per party

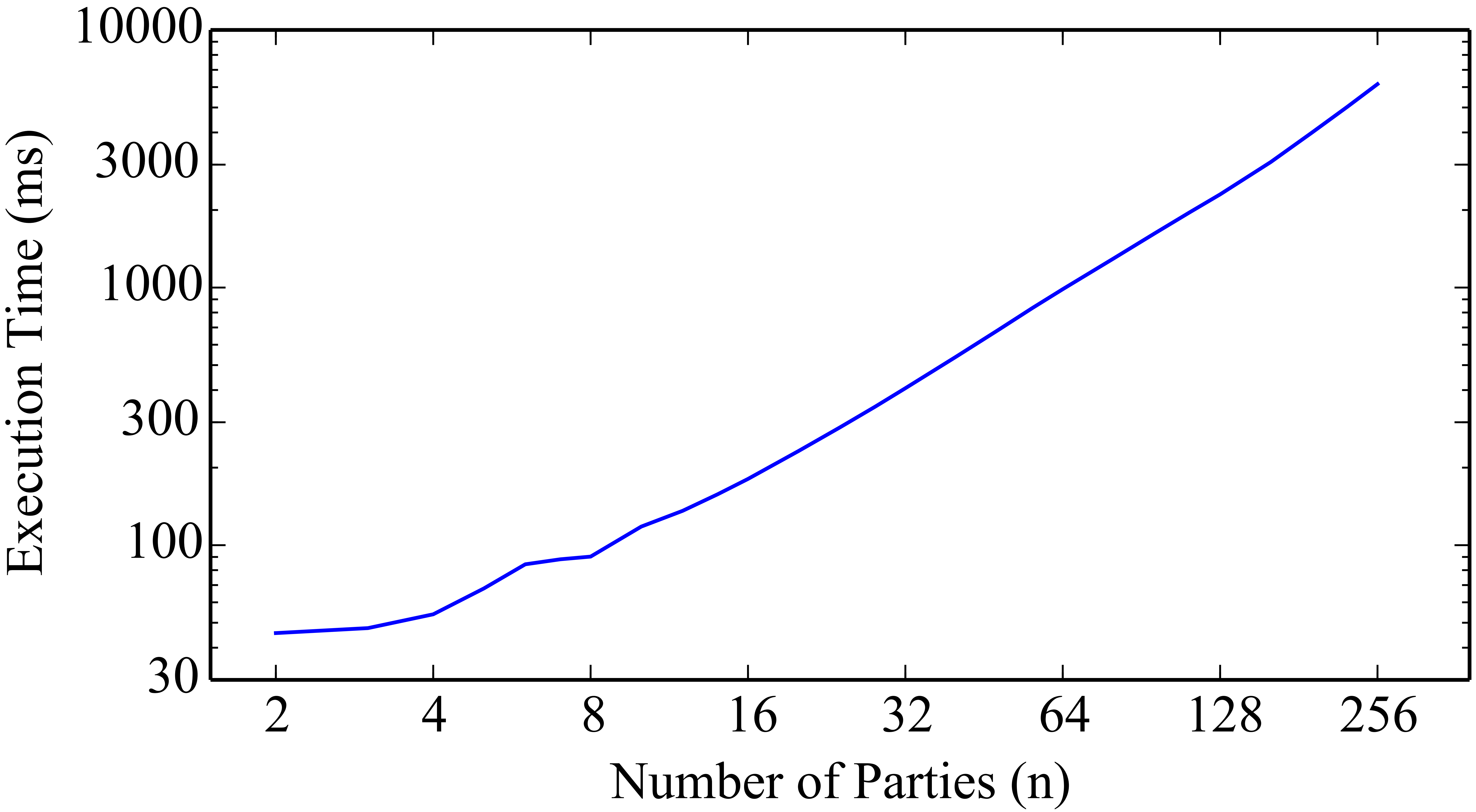
Benchmarks

- Implementation in **Rust**
- Ran benchmarks on Google Cloud
- One node per party
- **LAN** and **WAN** tests (up to **16 zones**)

Benchmarks

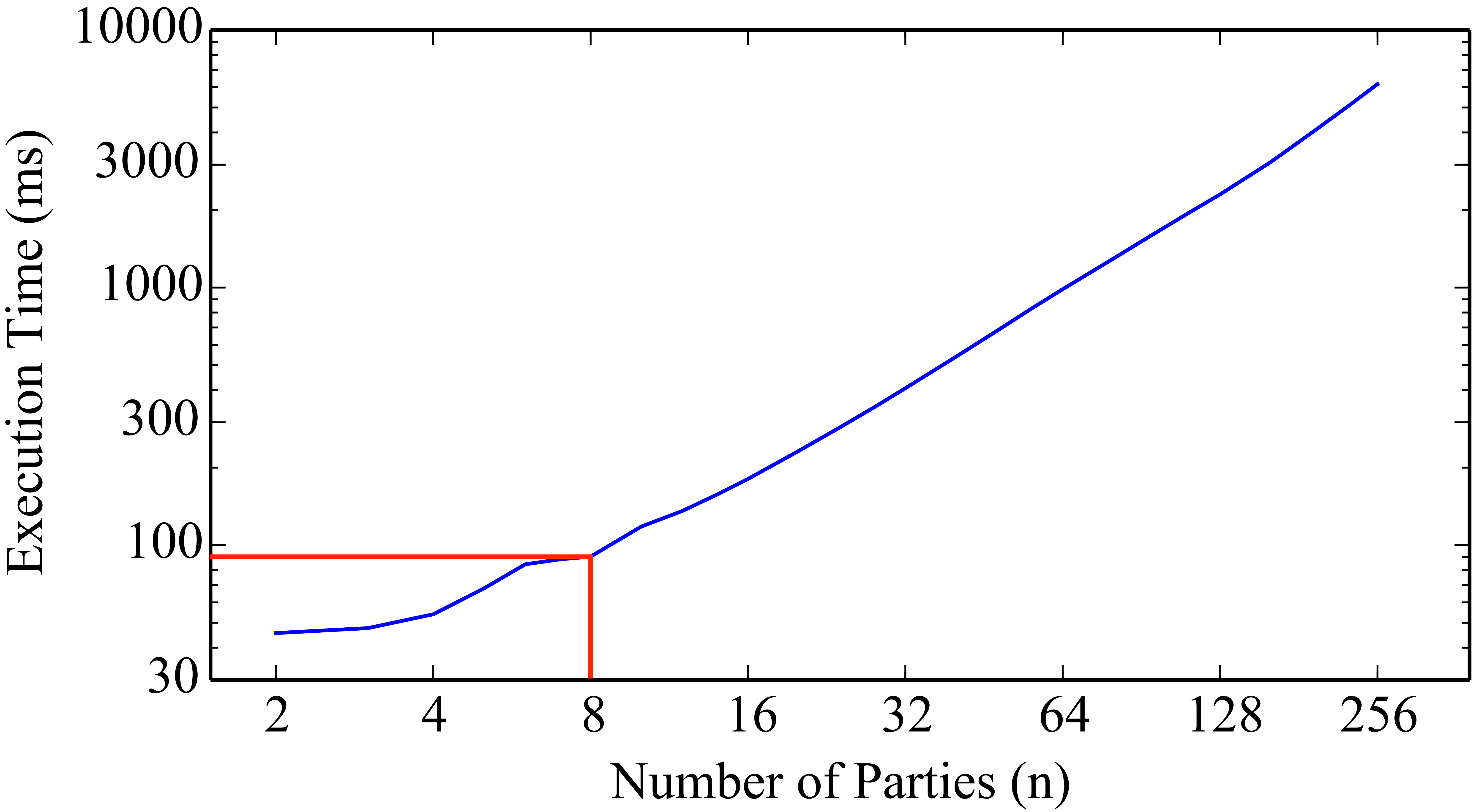
- Implementation in **Rust**
- Ran benchmarks on Google Cloud
- One node per party
- **LAN** and **WAN** tests (up to **16 zones**)
- **Low Power Friendliness**: Raspberry Pi (~93ms for 3-of-3)

LAN Setup



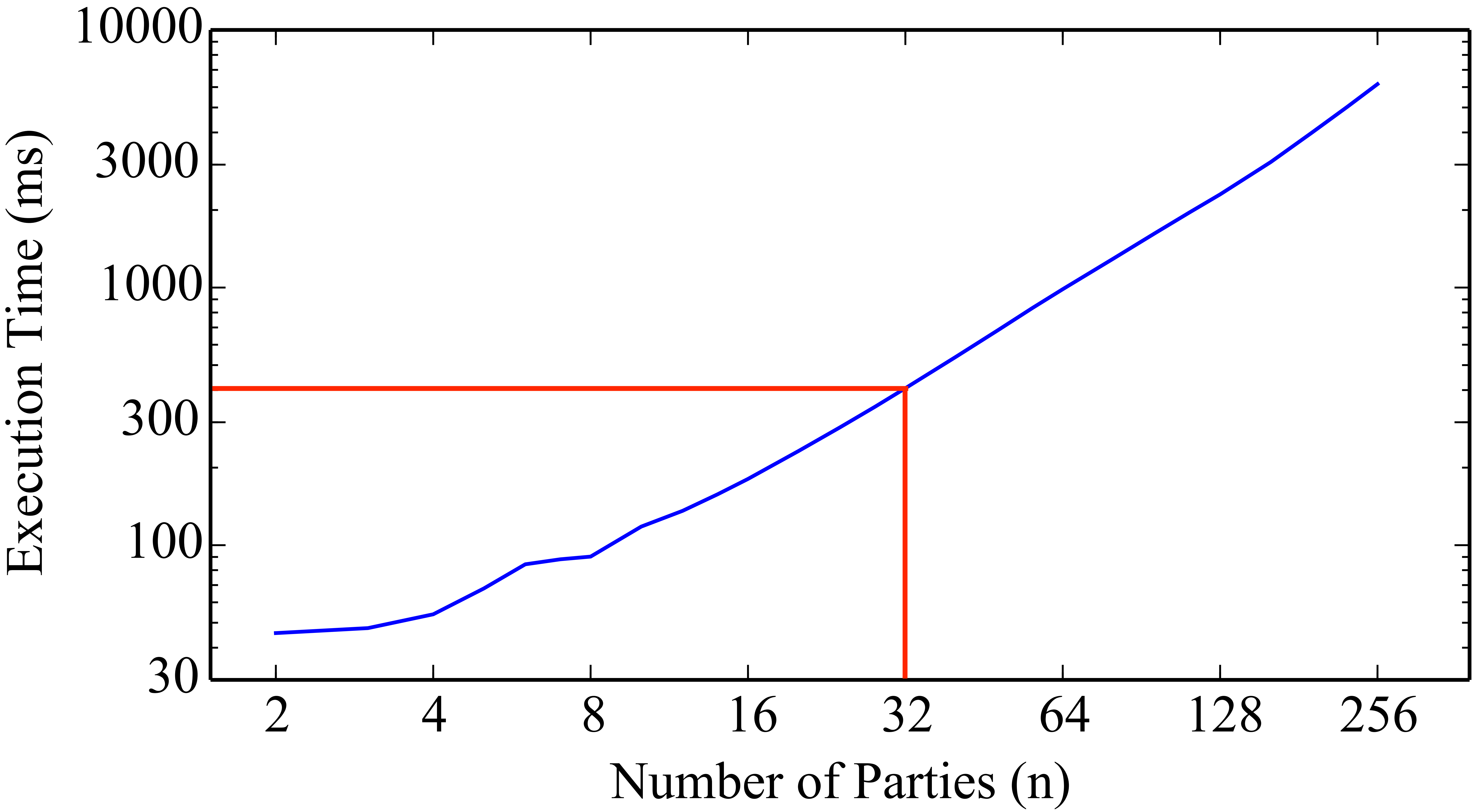
Broadcast PoK (DLog), **Pairwise**: 128 OTs

LAN Setup



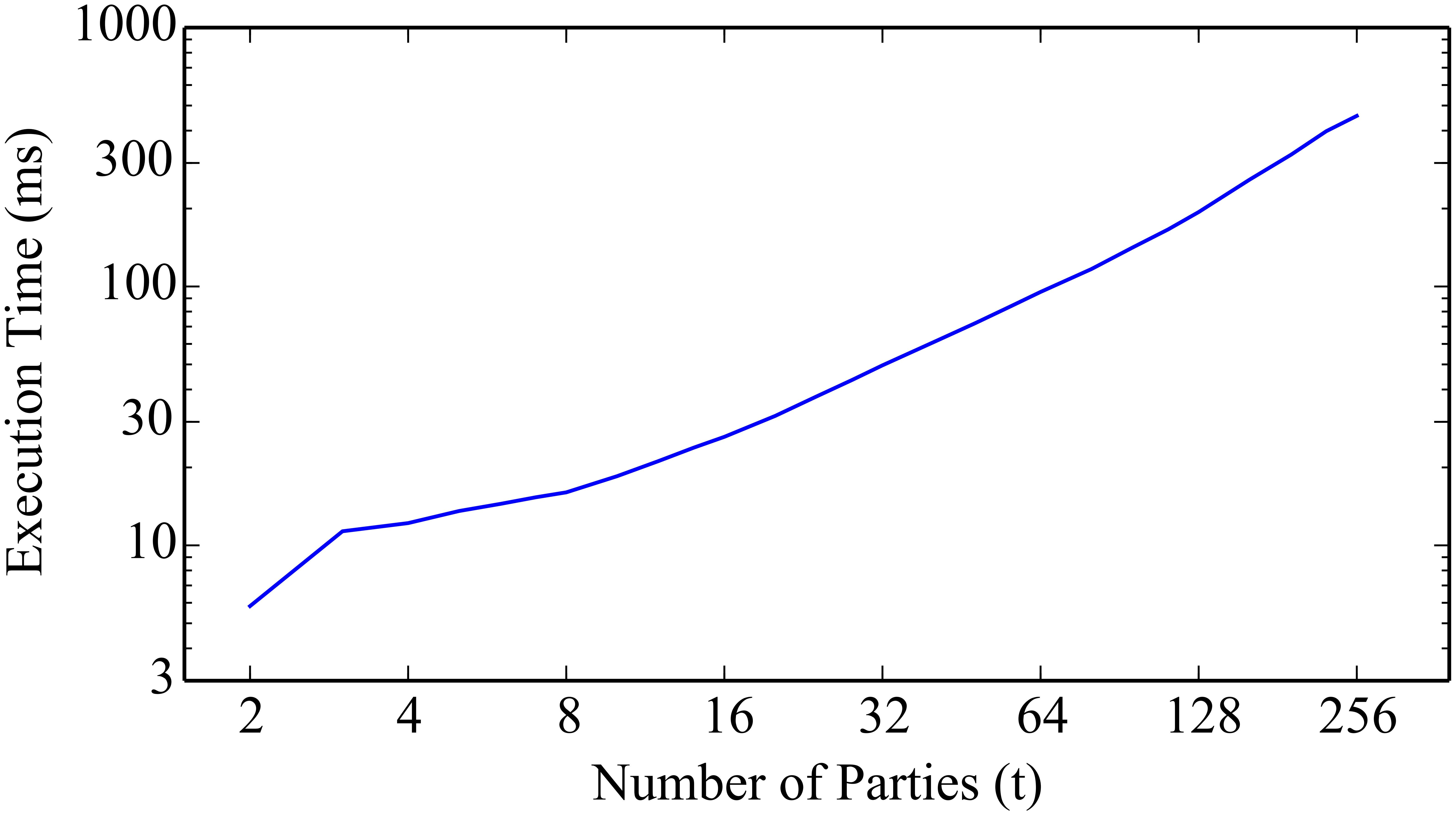
Broadcast PoK (DLog), **Pairwise**: 128 OTs

LAN Setup

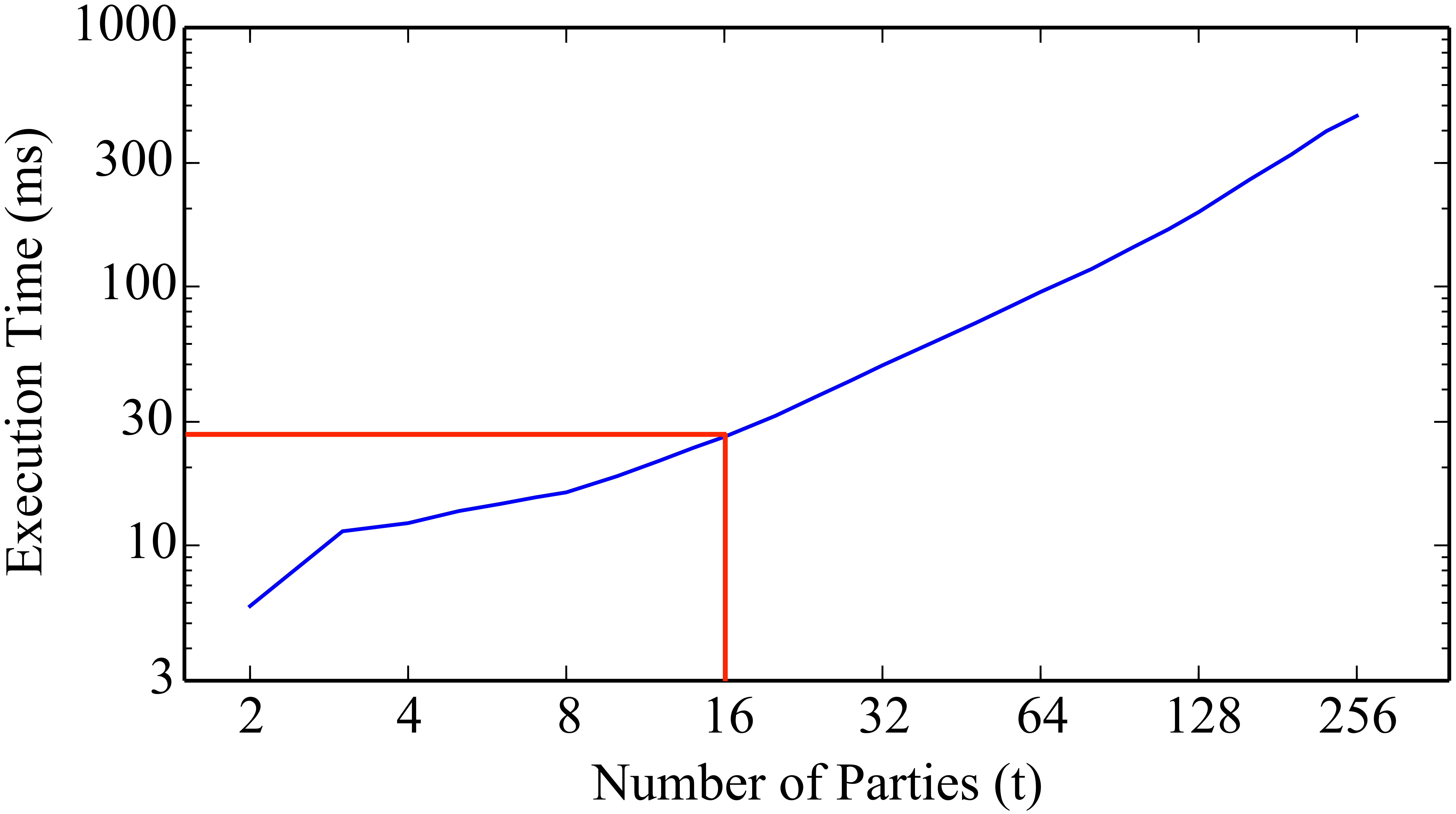


Broadcast PoK (DLog), **Pairwise**: 128 OTs

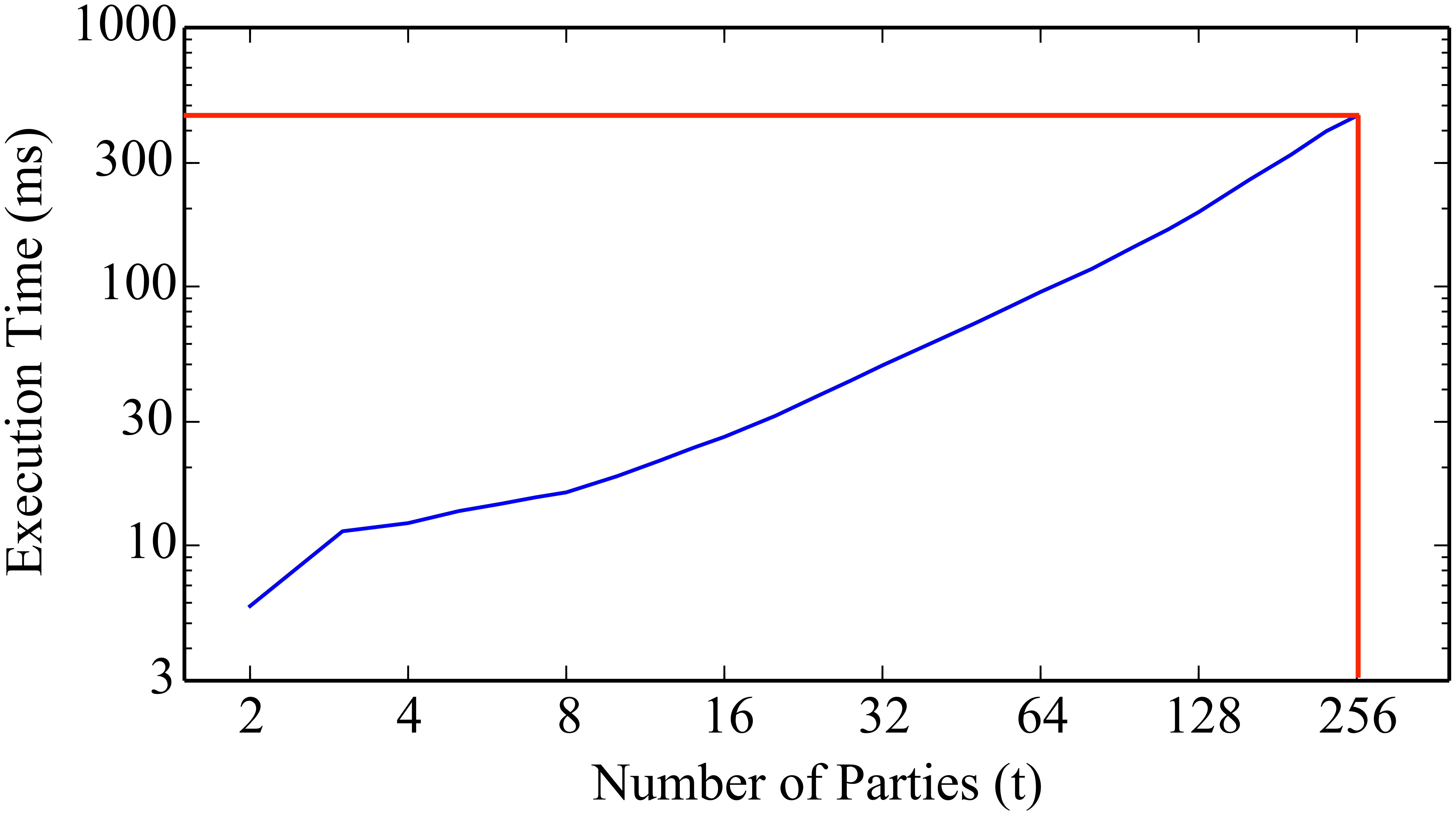
LAN Signing


1000
300
100
30
10
3
Execution Time (ms)
2
4
8
16
32
64
128
256
Number of Parties (t)

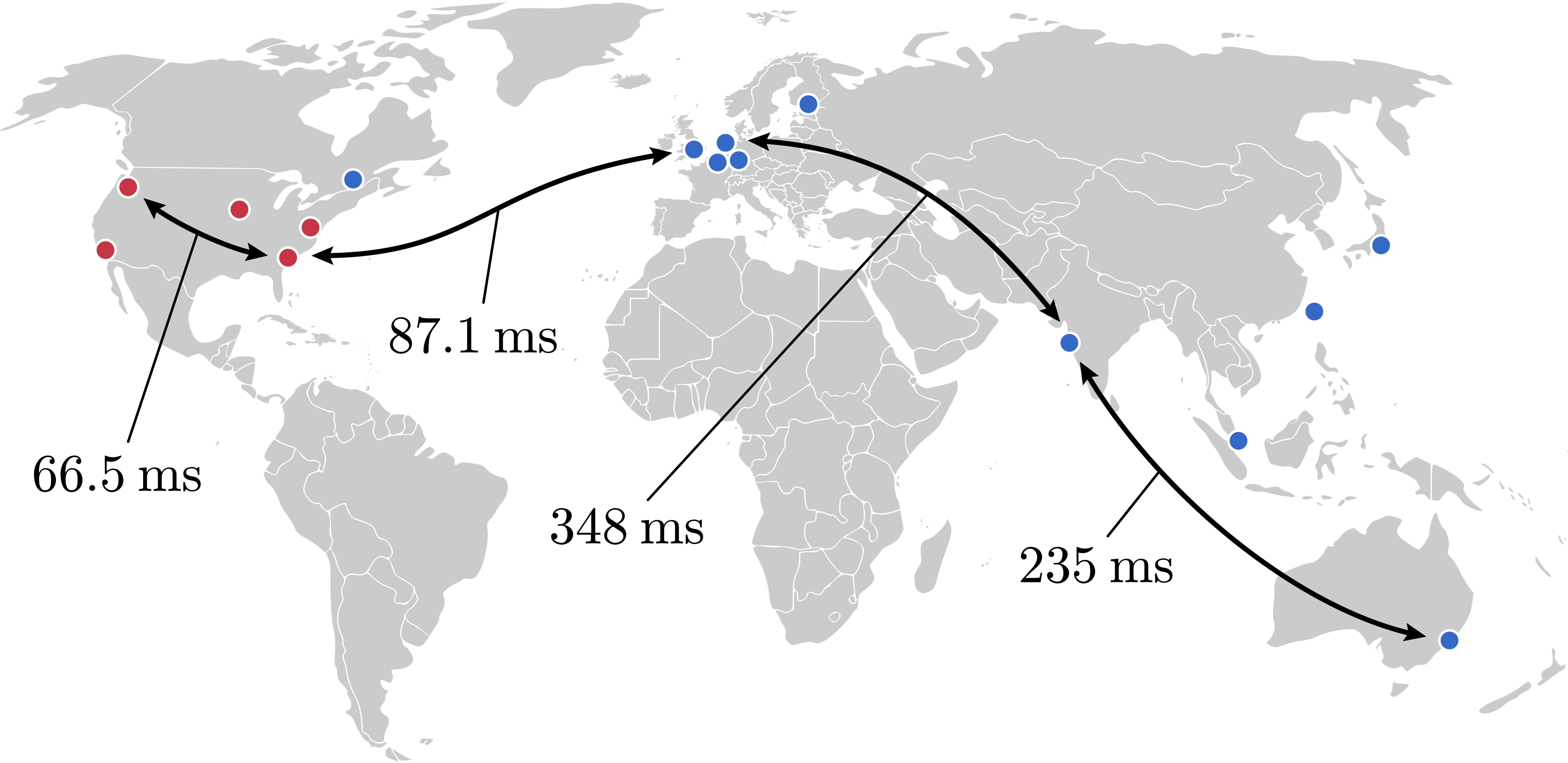
LAN Signing


Execution Time (ms)
1000
300
100
30
10
3
2
4
8
16
32
64
128
256
Number of Parties (t)

LAN Signing


1000
300
100
30
10
3
Execution Time (ms)
2
4
8
16
32
64
128
256
Number of Parties (t)

WAN Nodes



WAN Benchmarks

All time values in milliseconds

Parties/Zones	Signing Rounds	Signing Time	Setup Time
5/1	9	13.6	67.9
5/5	9	288	328
16/1	10	26.3	181
16/16	10	3045	1676
40/1	12	60.8	539
40/5	12	592	743
128/1	13	193.2	2300
128/16	13	4118	3424

WAN Benchmarks

All time values in milliseconds

Parties/Zones	Signing Rounds	Signing Time	Setup Time
5/1	9	13.6	67.9
5/5	9	288	328
16/1	10	26.3	181
16/16	10	3045	1676
40/1	12	60.8	539
40/5	12	592	743
128/1	13	193.2	2300
128/16	13	4118	3424

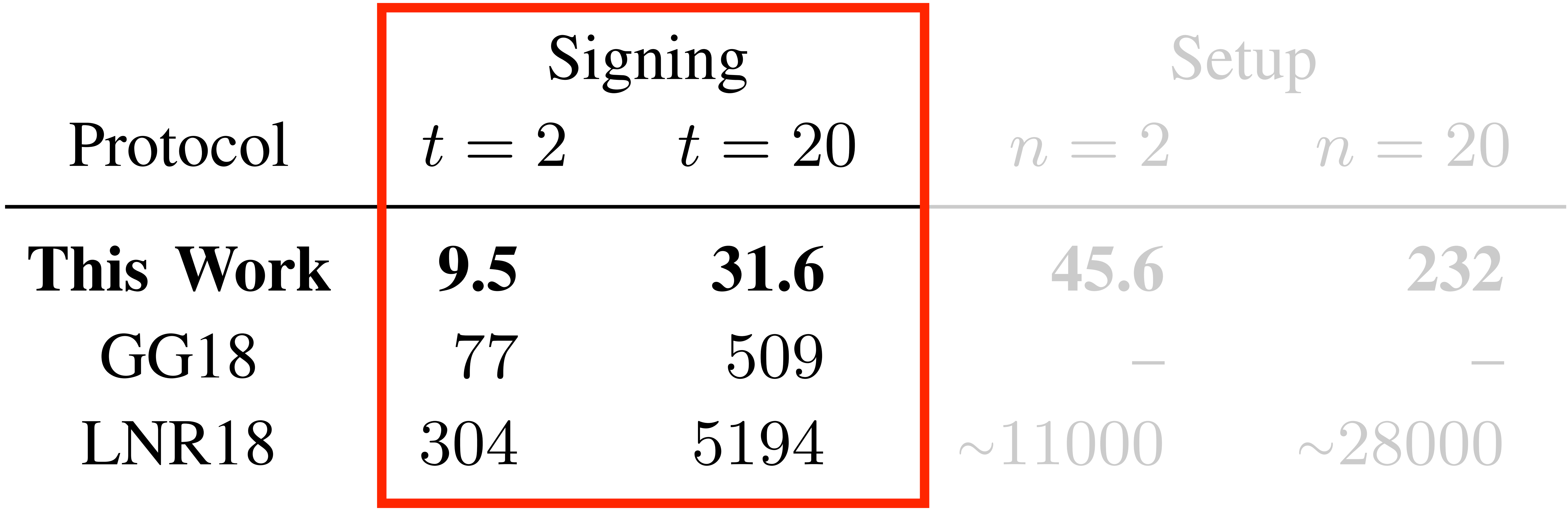
WAN Benchmarks

All time values in milliseconds

Parties/Zones	Signing Rounds	Signing Time	Setup Time
5/1	9	13.6	67.9
5/5	9	288	328
16/1	10	26.3	181
16/16	10	3045	1676
40/1	12	60.8	539
40/5	12	592	743
128/1	13	193.2	2300
128/16	13	4118	3424

Comparison

All time figures in milliseconds



	Signing		Setup	
Protocol	$t = 2$	$t = 20$	$n = 2$	$n = 20$
This Work	**9.5**	**31.6**	**45.6**	**232**
GG18	77	509	–	–
LNR18	304	5194	$\sim$11000	$\sim$28000

Note: Our figures are wall-clock times; includes network costs

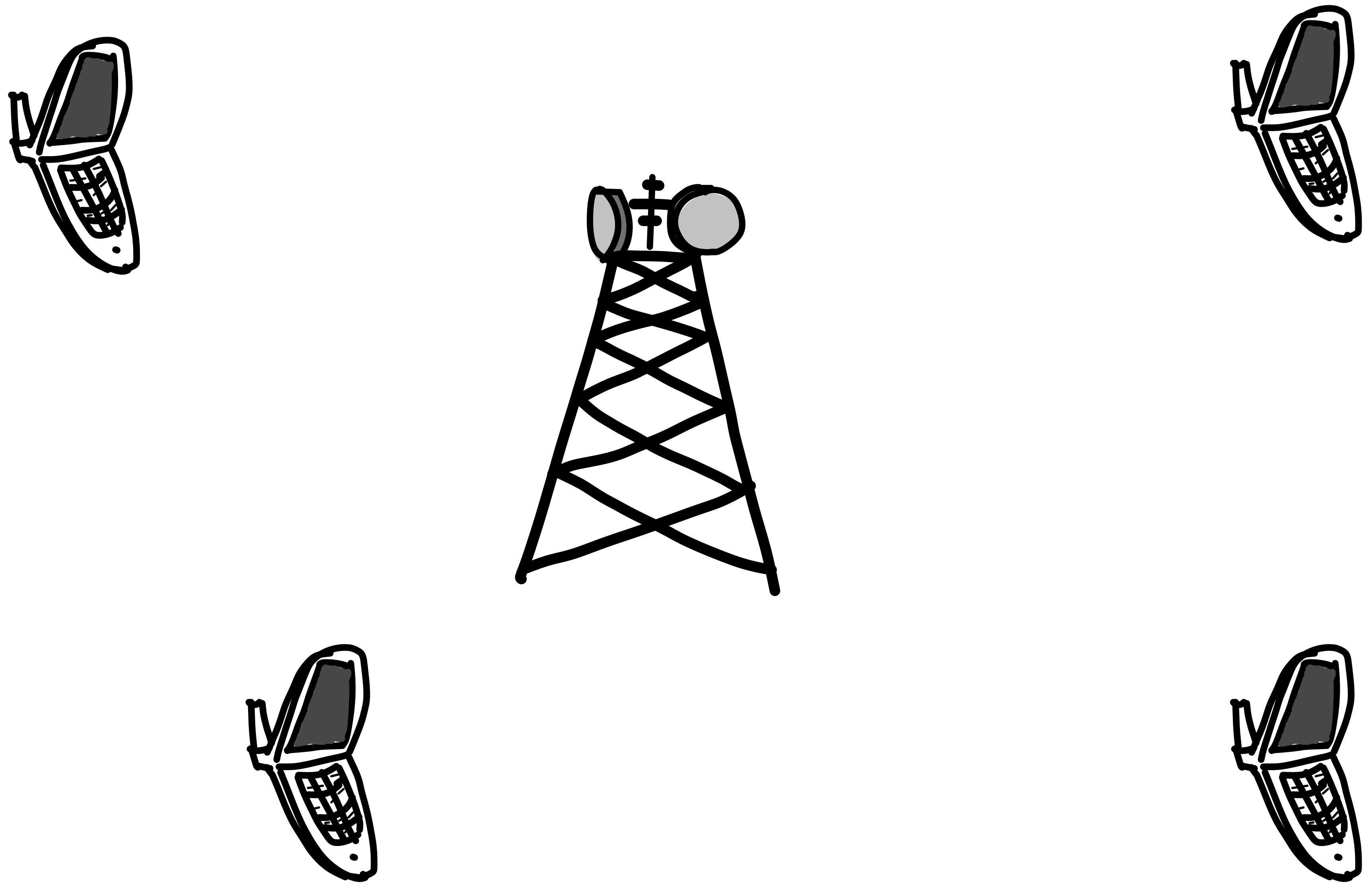
Comparison

All time figures in milliseconds

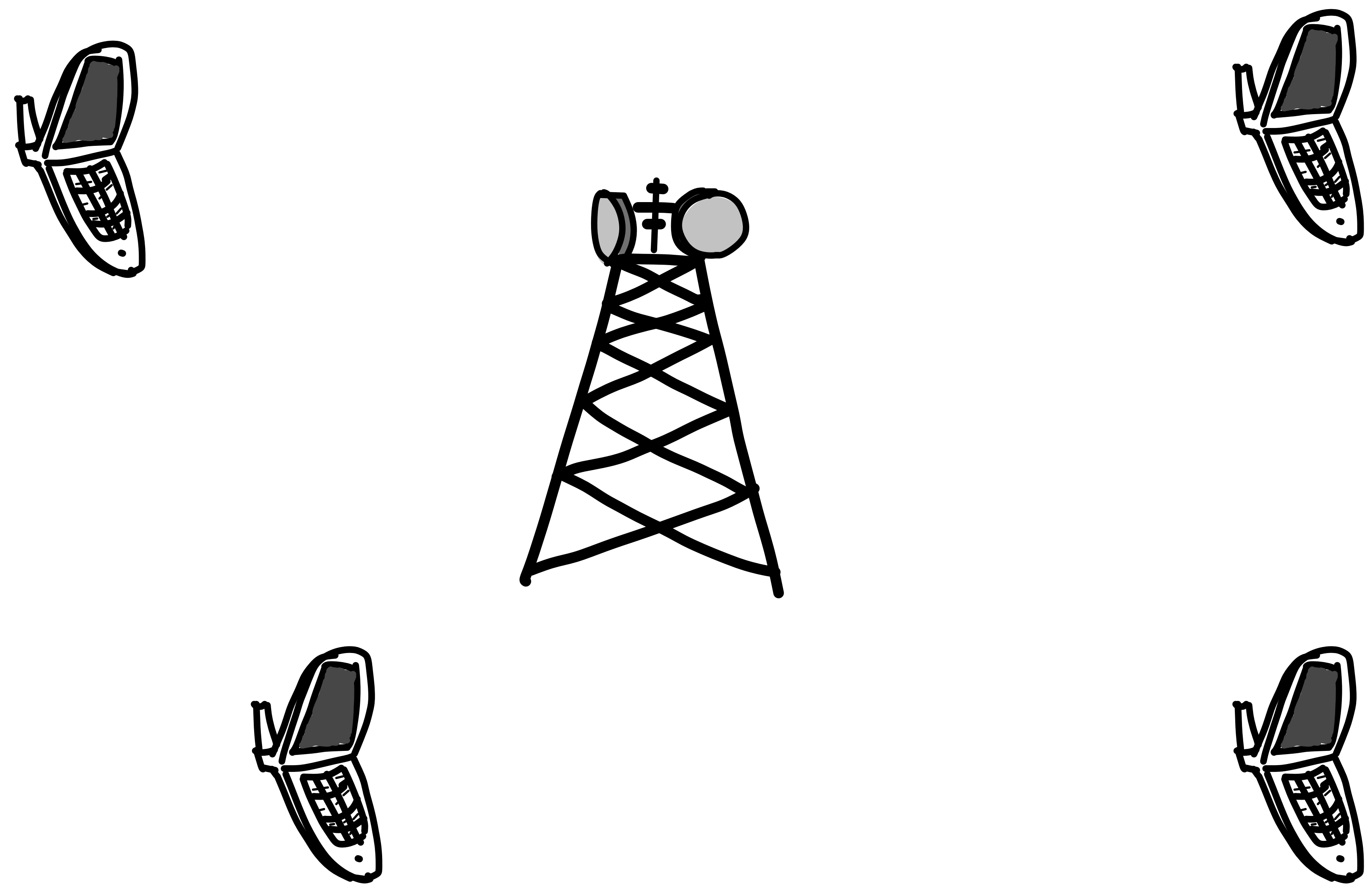
	Signing		Setup	
Protocol	$t = 2$	$t = 20$	$n = 2$	$n = 20$
This Work	**9.5**	**31.6**	**45.6**	**232**
GG18	77	509	–	–
LNR18	304	5194	$\sim$11000	$\sim$28000

Note: Our figures are wall-clock times; includes network costs

Is communication the bottleneck?



Is communication the bottleneck?



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

Is communication the bottleneck?

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

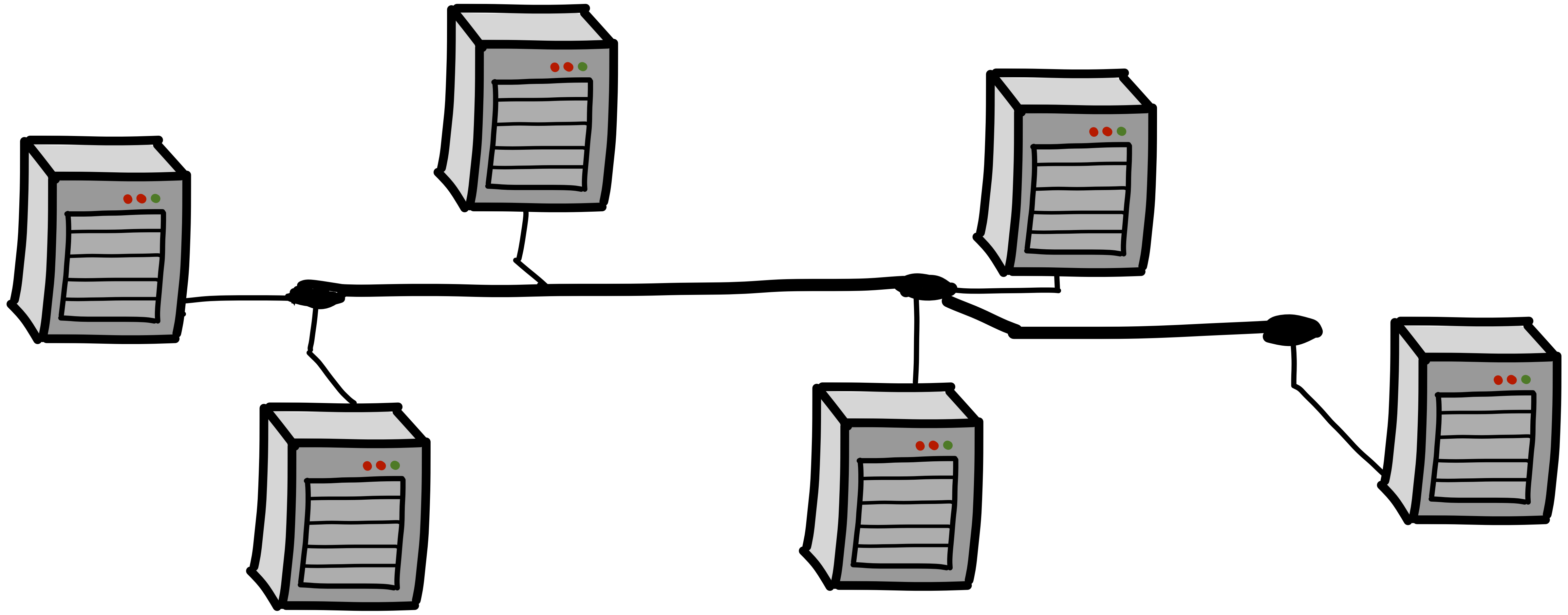
Is communication the bottleneck?

- **Mobile applications (human-initiated)**:
  - eg. t=4, <4Mb transmitted per party

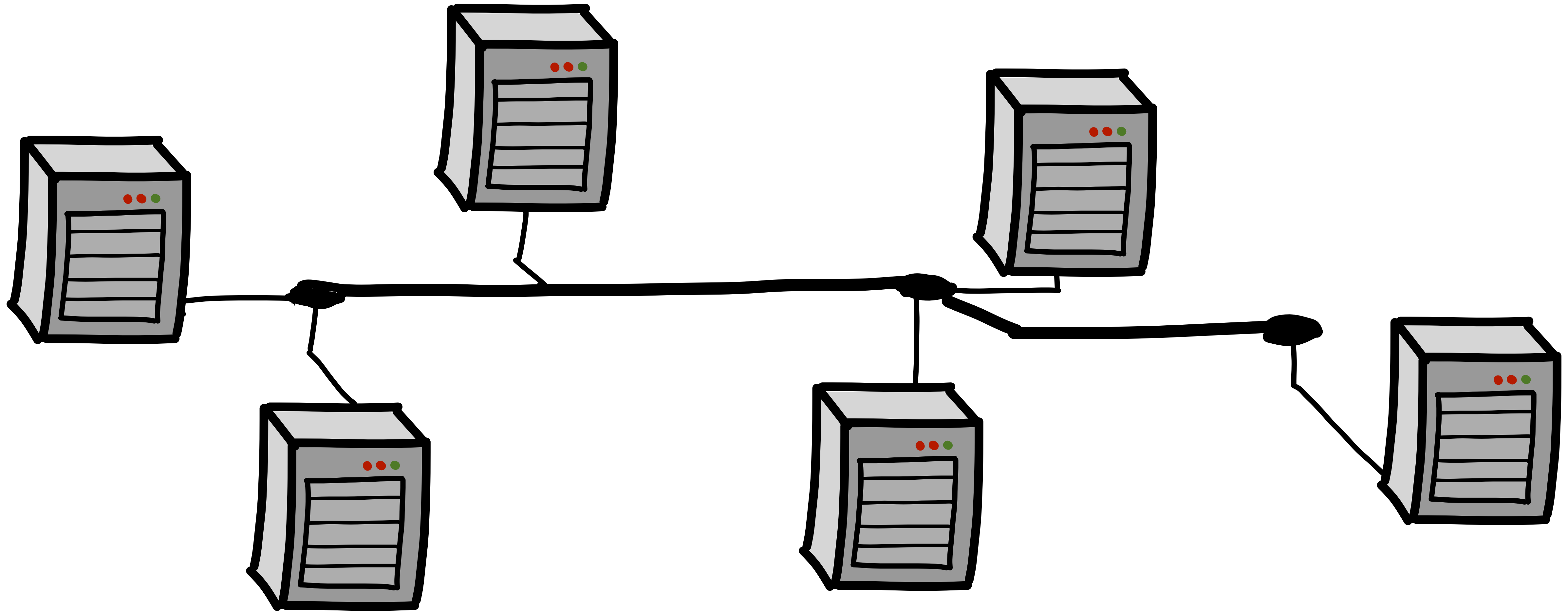
Is communication the bottleneck?

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

Is communication the bottleneck?

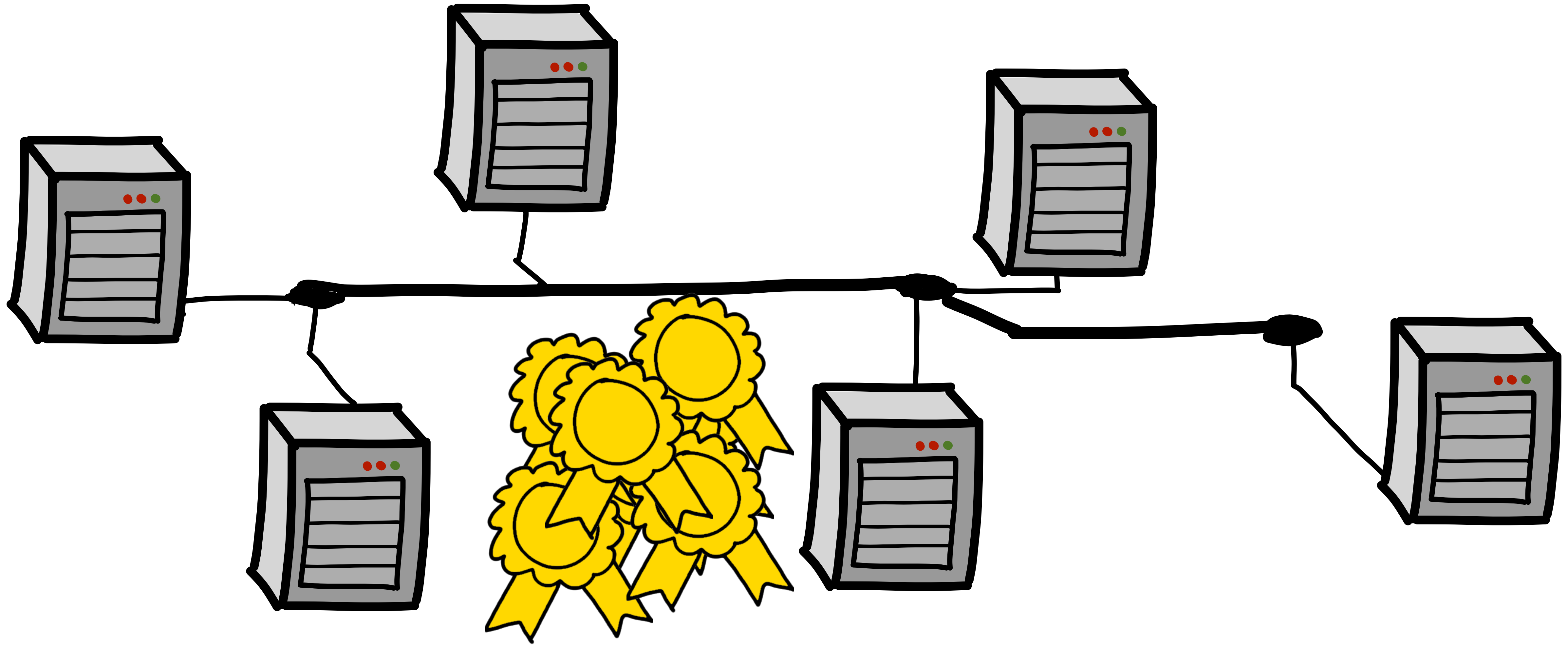


Is communication the bottleneck?



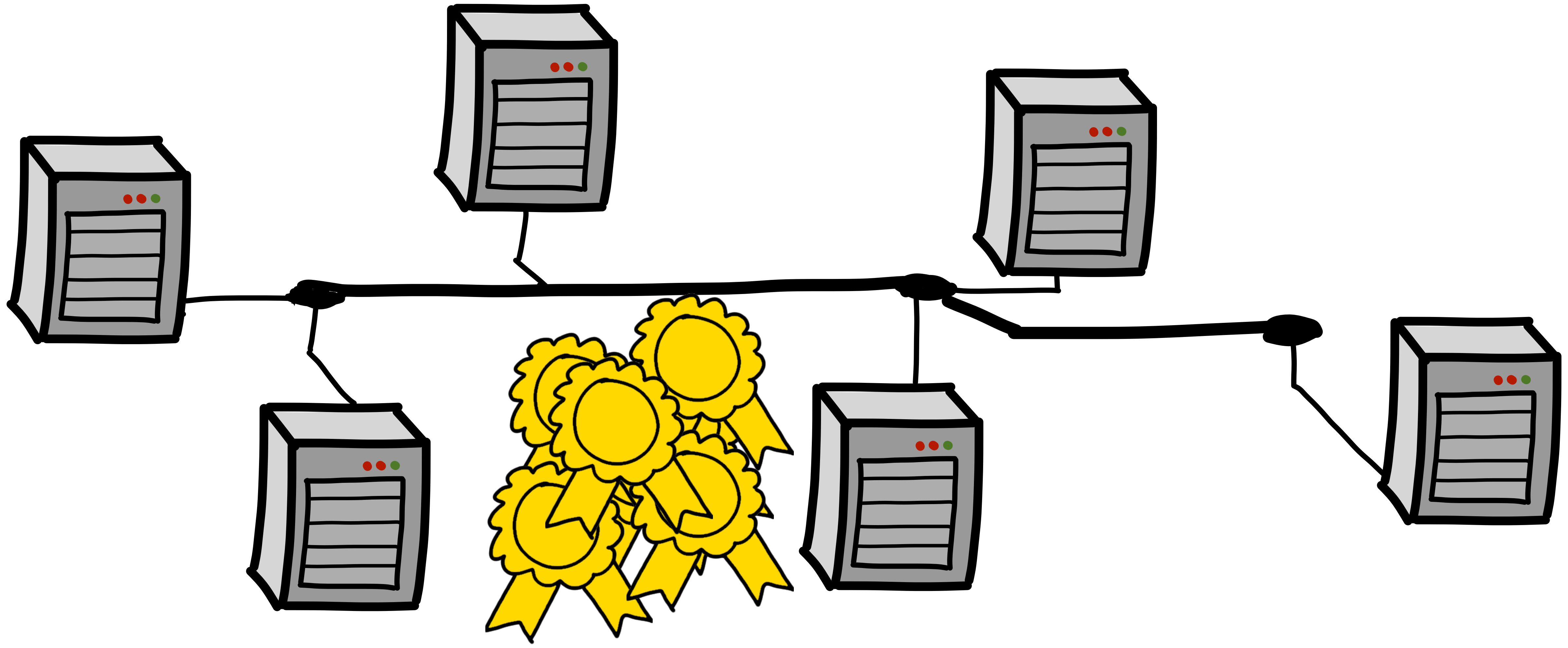
- **Large-scale automated distributed signing:**

Is communication the bottleneck?



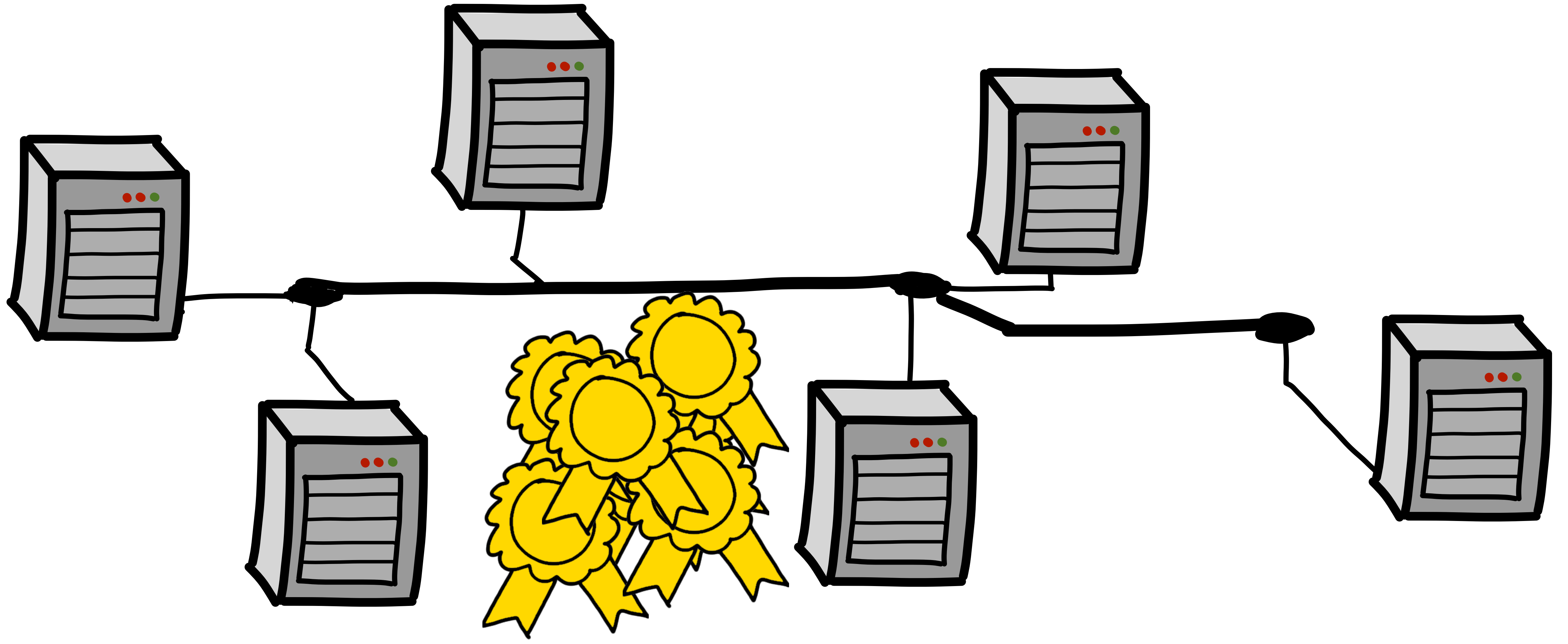
- **Large-scale automated distributed signing:**

Is communication the bottleneck?



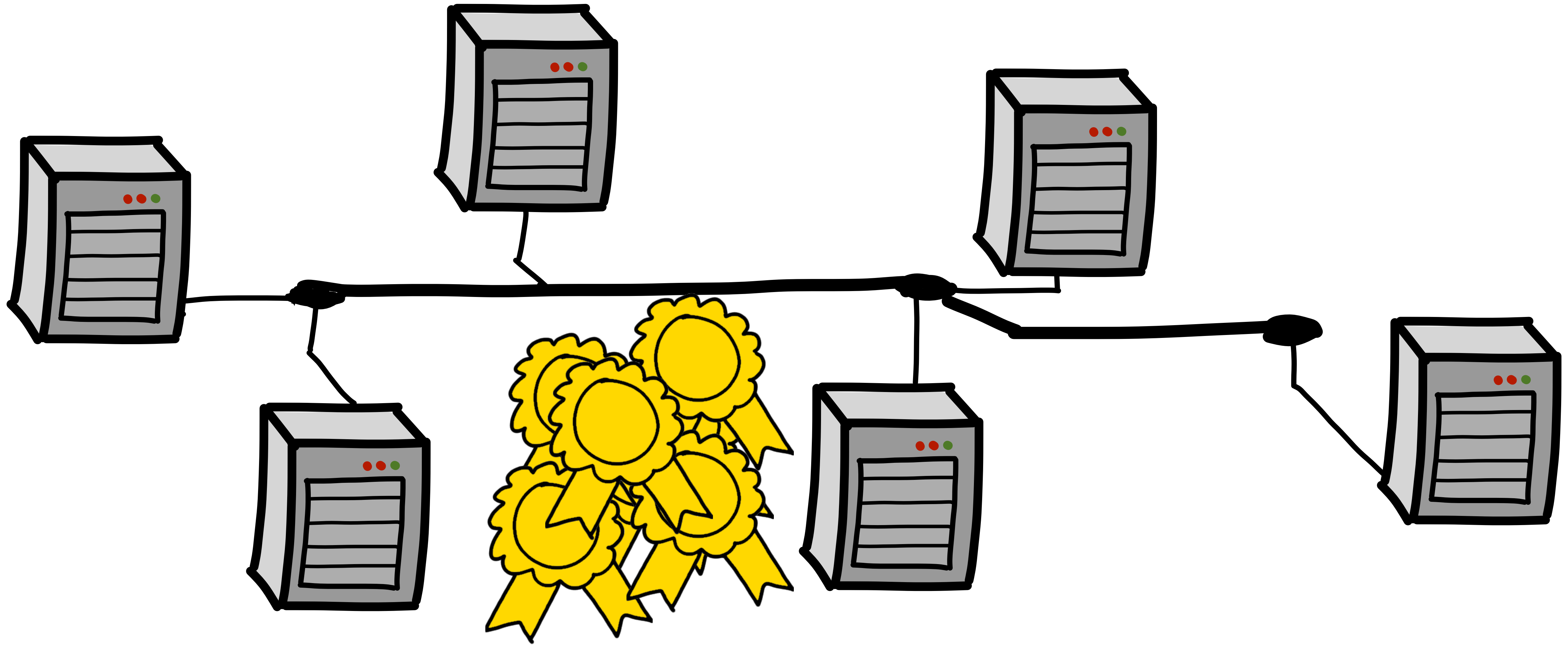
- **Large-scale automated distributed signing:**
  - Threshold 2:   3.8ms/sig   <= ~263 sig/second

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?

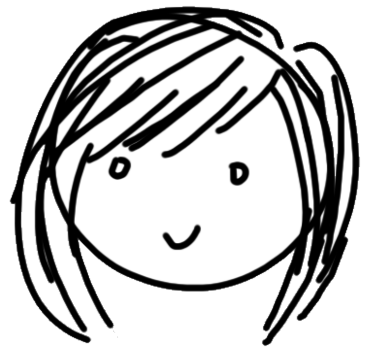


- **Large-scale automated distributed signing:**
  - Threshold 2: 3.8ms/sig <= ~263 sig/second
  - Threshold 20: 31.6ms/sig <= ~31 sig/second
- Both settings need <500Mbps bandwidth

Special Case: 2-of-n

- [DKLs18]: Specialized protocol when t=2
- Only one party gets output
- Weaker functionality: Other party can rejection-sample public nonce R

Result



...

...

$$\Gamma^{(1)} = t_A^{(1)} \cdot R + \phi \cdot k_A \cdot G \qquad \Gamma^{(1)} = G - t_B^{(1)} \cdot R$$

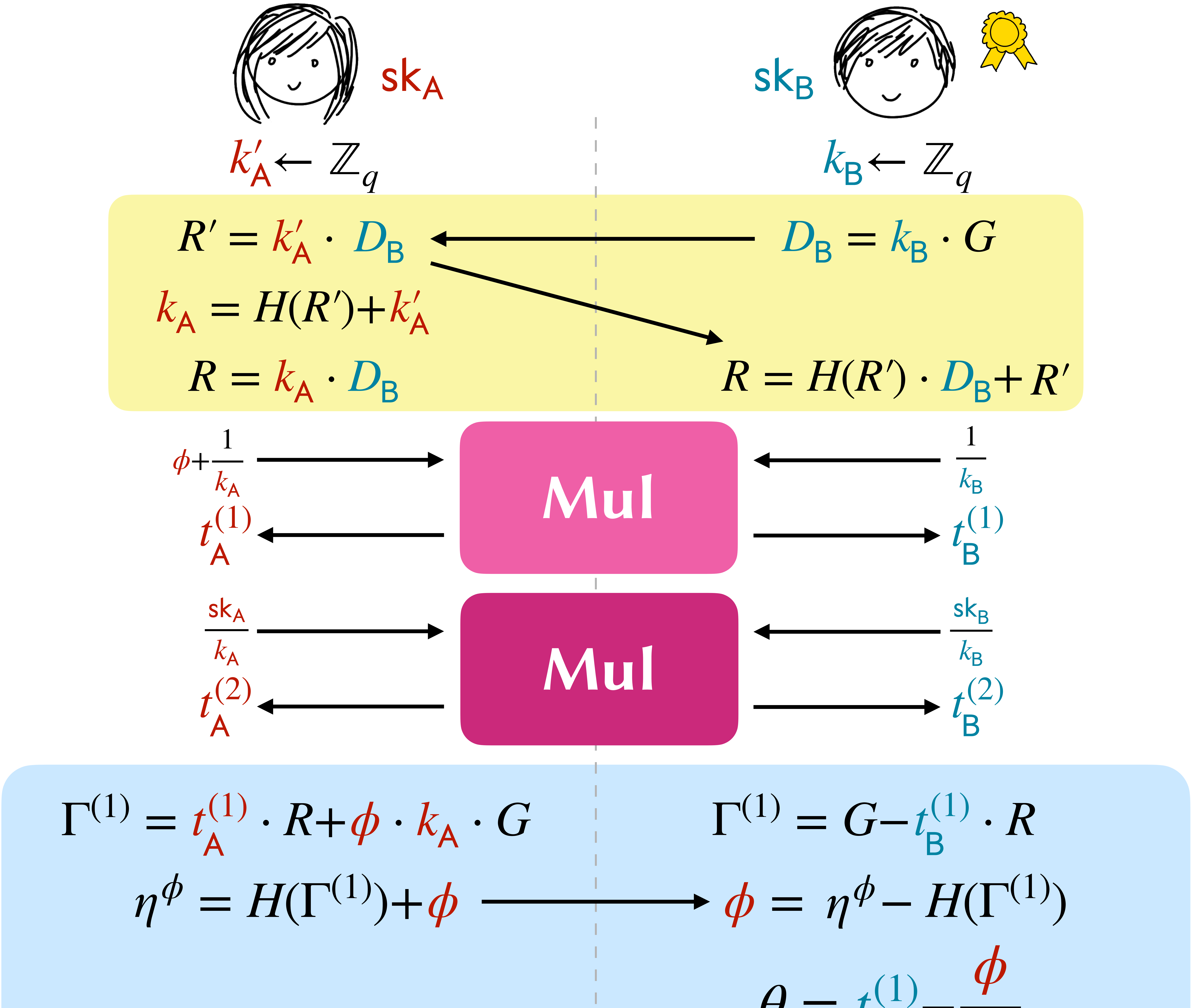
$$\eta^{\phi} = H(\Gamma^{(1)}) + \phi \longrightarrow \phi = \eta^{\phi} - H(\Gamma^{(1)})$$

$$\theta = t_B^{(1)} - \frac{\phi}{k_B}$$

$$\Gamma^{(2)} = t_A^{(1)} \cdot \mathsf{pk} - t_A^{(2)} \cdot G \qquad \Gamma^{(2)} = t_B^{(2)} \cdot G - \theta \cdot \mathsf{pk}$$

$$s_A = t_A^{(1)} \cdot H(m) + t_A^{(2)} \cdot r_x \qquad s_B = \theta \cdot H(m) + t_B^{(2)} \cdot r_x$$

$$\eta^{s} = H(\Gamma^{(2)}) + s_A \longrightarrow s = \eta^{s} - H(\Gamma^{(2)}) + s_B$$



sk_A

sk_B

$k'_\mathsf{A} \leftarrow \mathbb{Z}_q$

$k_\mathsf{B} \leftarrow \mathbb{Z}_q$

$R' = k'_\mathsf{A} \cdot D_\mathsf{B}$

$D_\mathsf{B} = k_\mathsf{B} \cdot G$

$k_\mathsf{A} = H(R') + k'_\mathsf{A}$

$R = k_\mathsf{A} \cdot D_\mathsf{B}$

$R = H(R') \cdot D_\mathsf{B} + R'$

$\phi + \frac{1}{k_\mathsf{A}}$

$t_\mathsf{A}^{(1)}$

$\frac{1}{k_\mathsf{B}}$

$t_\mathsf{B}^{(1)}$

$\frac{\mathsf{sk}_\mathsf{A}}{k_\mathsf{A}}$

$t_\mathsf{A}^{(2)}$

$\frac{\mathsf{sk}_\mathsf{B}}{k_\mathsf{B}}$

$t_\mathsf{B}^{(2)}$

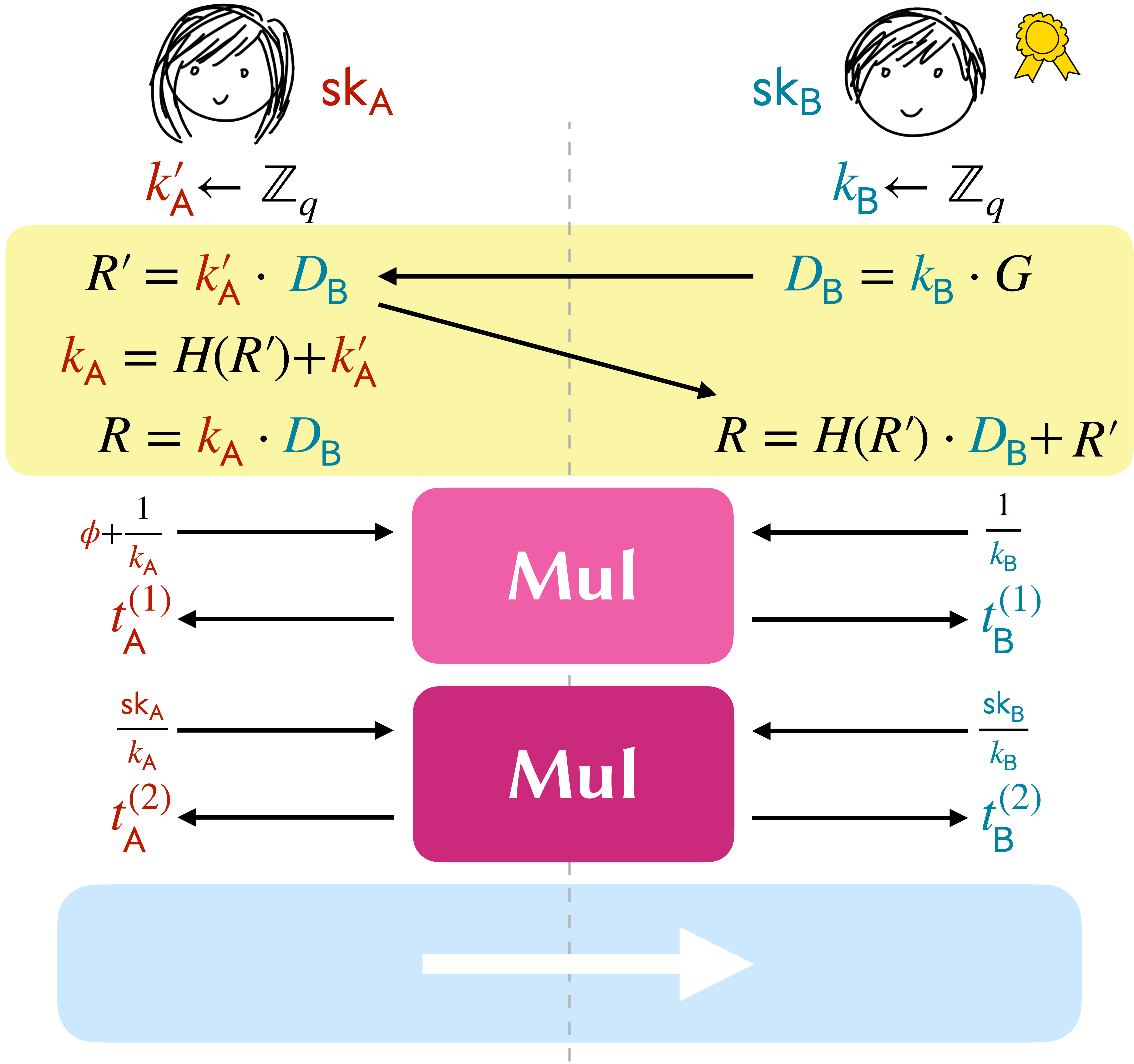
$\Gamma^{(1)} = t_\mathsf{A}^{(1)} \cdot R + \phi \cdot k_\mathsf{A} \cdot G$

$\Gamma^{(1)} = G - t_\mathsf{B}^{(1)} \cdot R$

$\eta^\phi = H(\Gamma^{(1)}) + \phi$

$\phi = \eta^\phi - H(\Gamma^{(1)})$

$\theta = t^{(1)} - \frac{\phi}{}$



sk_{A} sk_{B}

$k'_{\mathsf{A}} \leftarrow \mathbb{Z}_q$ $k_{\mathsf{B}} \leftarrow \mathbb{Z}_q$

$R' = k'_{\mathsf{A}} \cdot D_{\mathsf{B}}$ $D_{\mathsf{B}} = k_{\mathsf{B}} \cdot G$

$k_{\mathsf{A}} = H(R') + k'_{\mathsf{A}}$

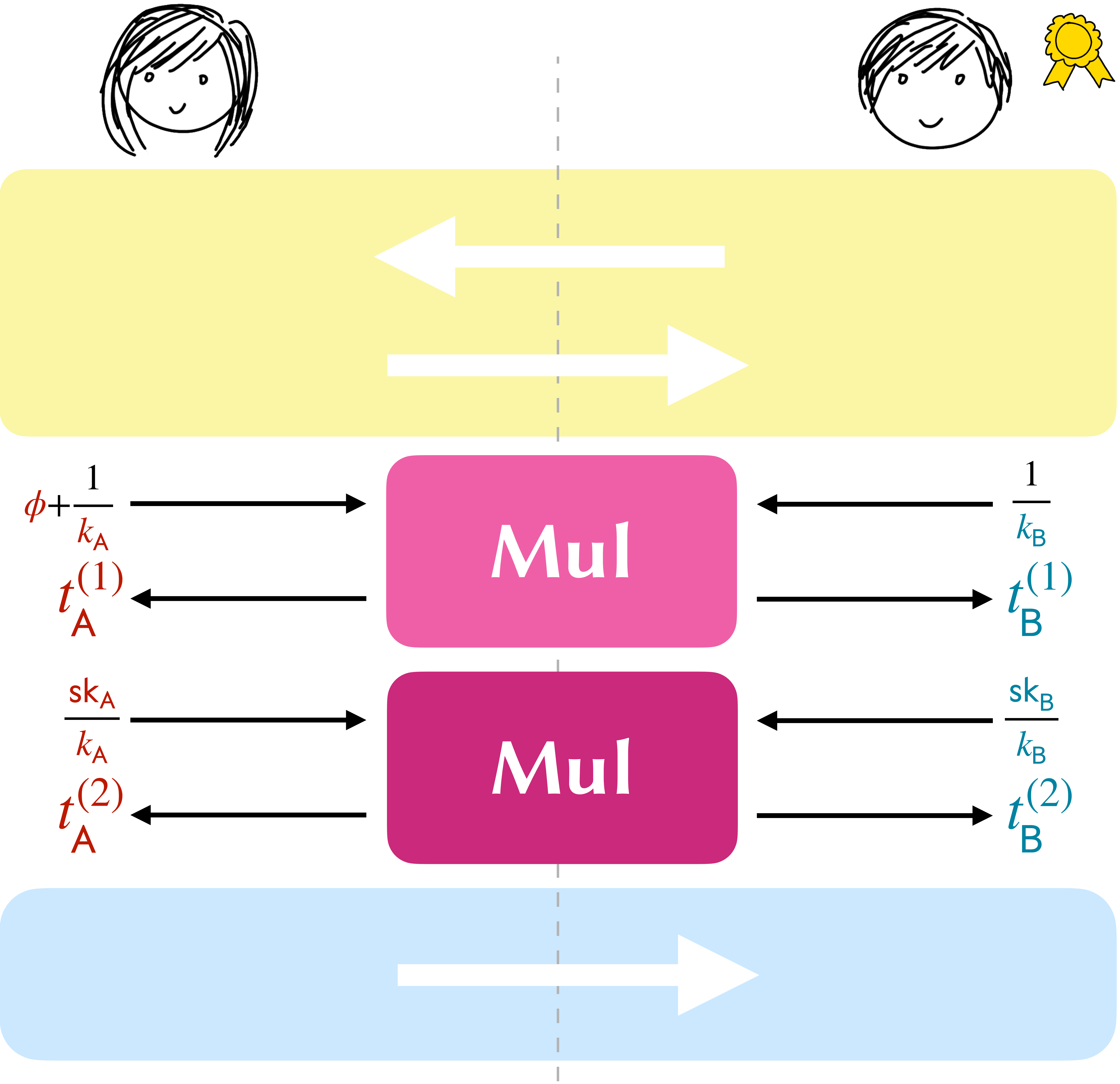
$R = k_{\mathsf{A}} \cdot D_{\mathsf{B}}$ $R = H(R') \cdot D_{\mathsf{B}} + R'$

$\phi + \frac{1}{k_{\mathsf{A}}}$ $\frac{1}{k_{\mathsf{B}}}$

$t_{\mathsf{A}}^{(1)}$ $t_{\mathsf{B}}^{(1)}$

$\frac{\mathsf{sk}_{\mathsf{A}}}{k_{\mathsf{A}}}$ $\frac{\mathsf{sk}_{\mathsf{B}}}{k_{\mathsf{B}}}$

$t_{\mathsf{A}}^{(2)}$ $t_{\mathsf{B}}^{(2)}$

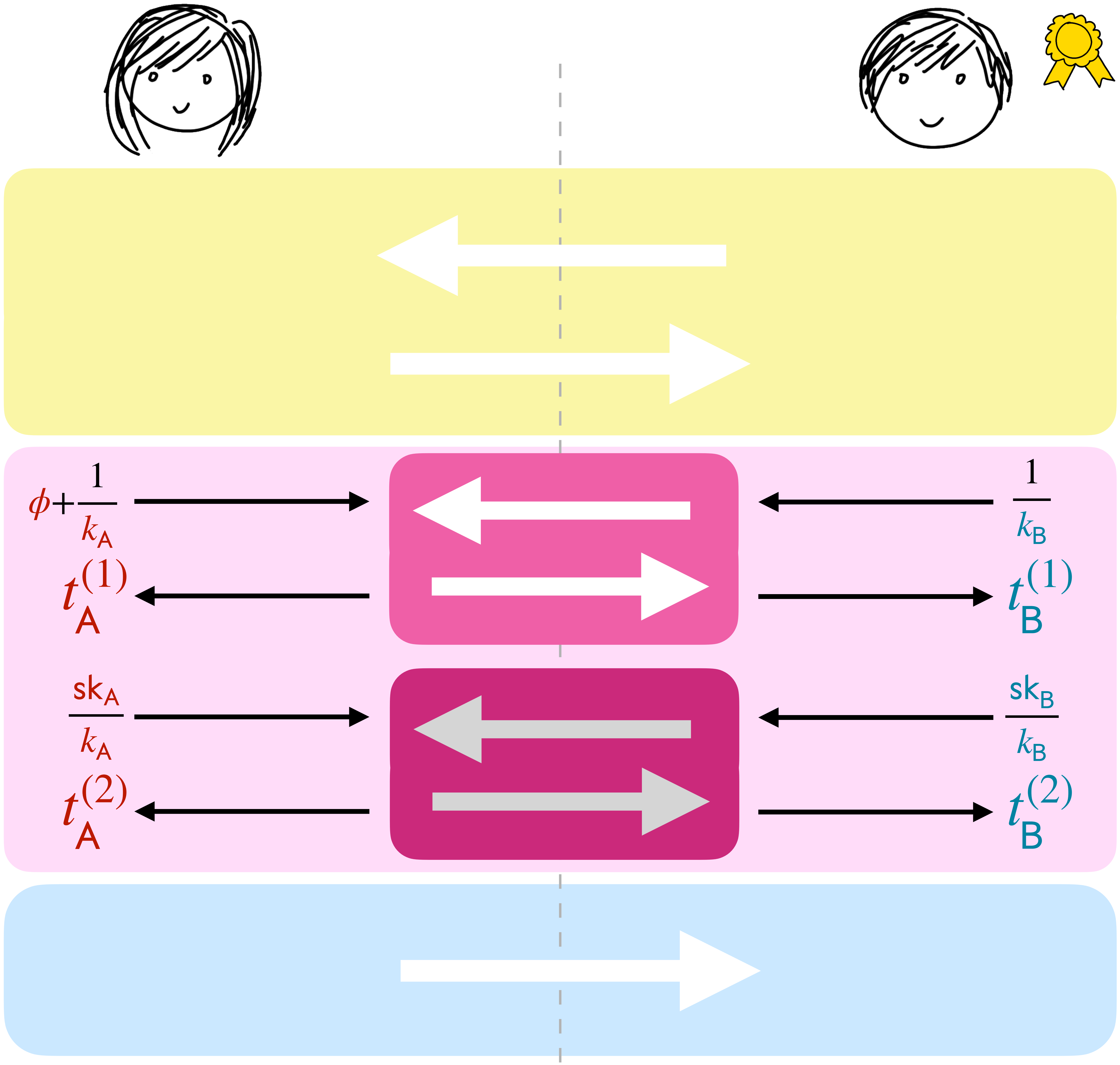


$\phi + \frac{1}{k_\mathsf{A}}$ Mul $\frac{1}{k_\mathsf{B}}$

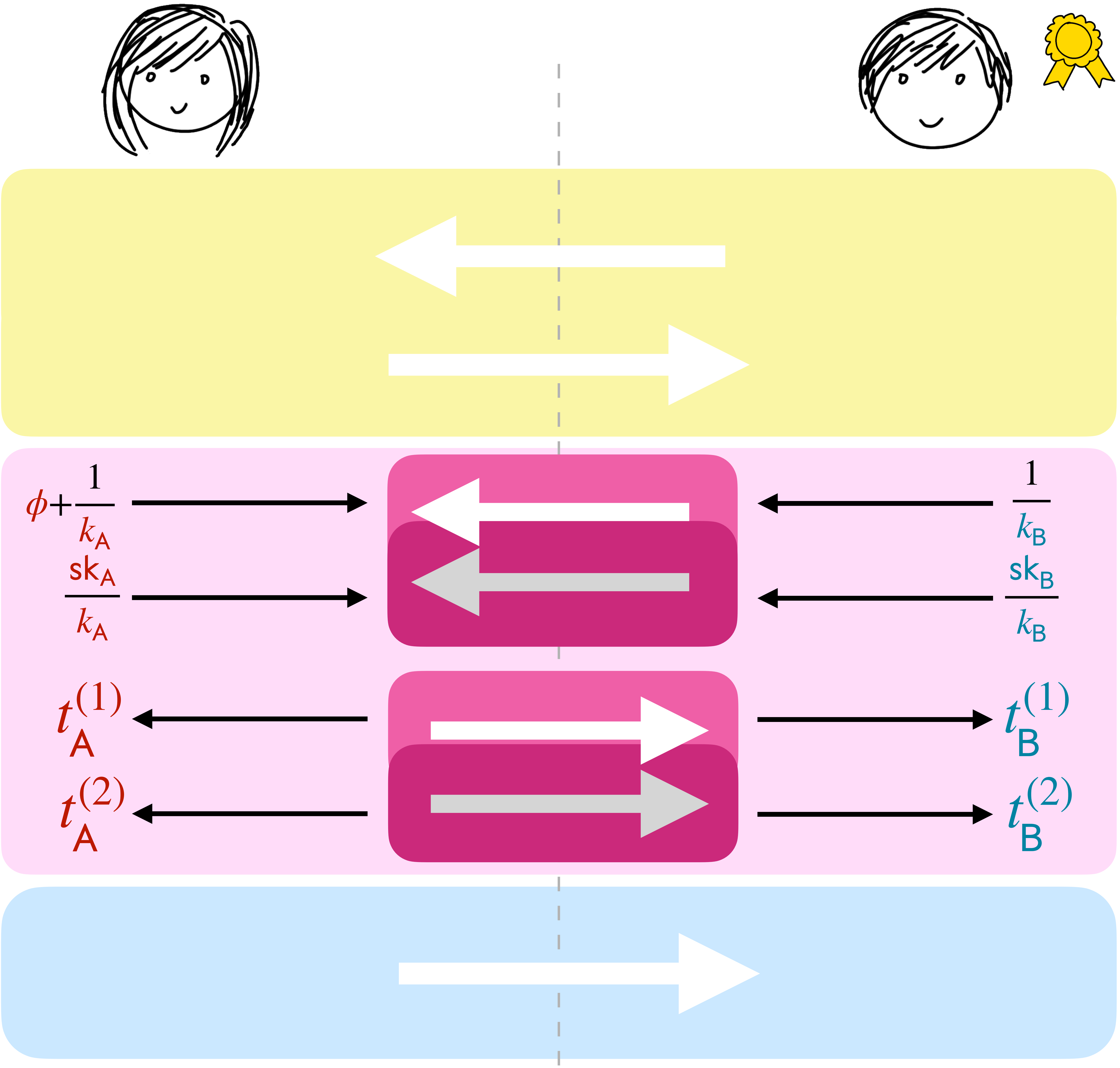
$t_\mathsf{A}^{(1)}$ $t_\mathsf{B}^{(1)}$

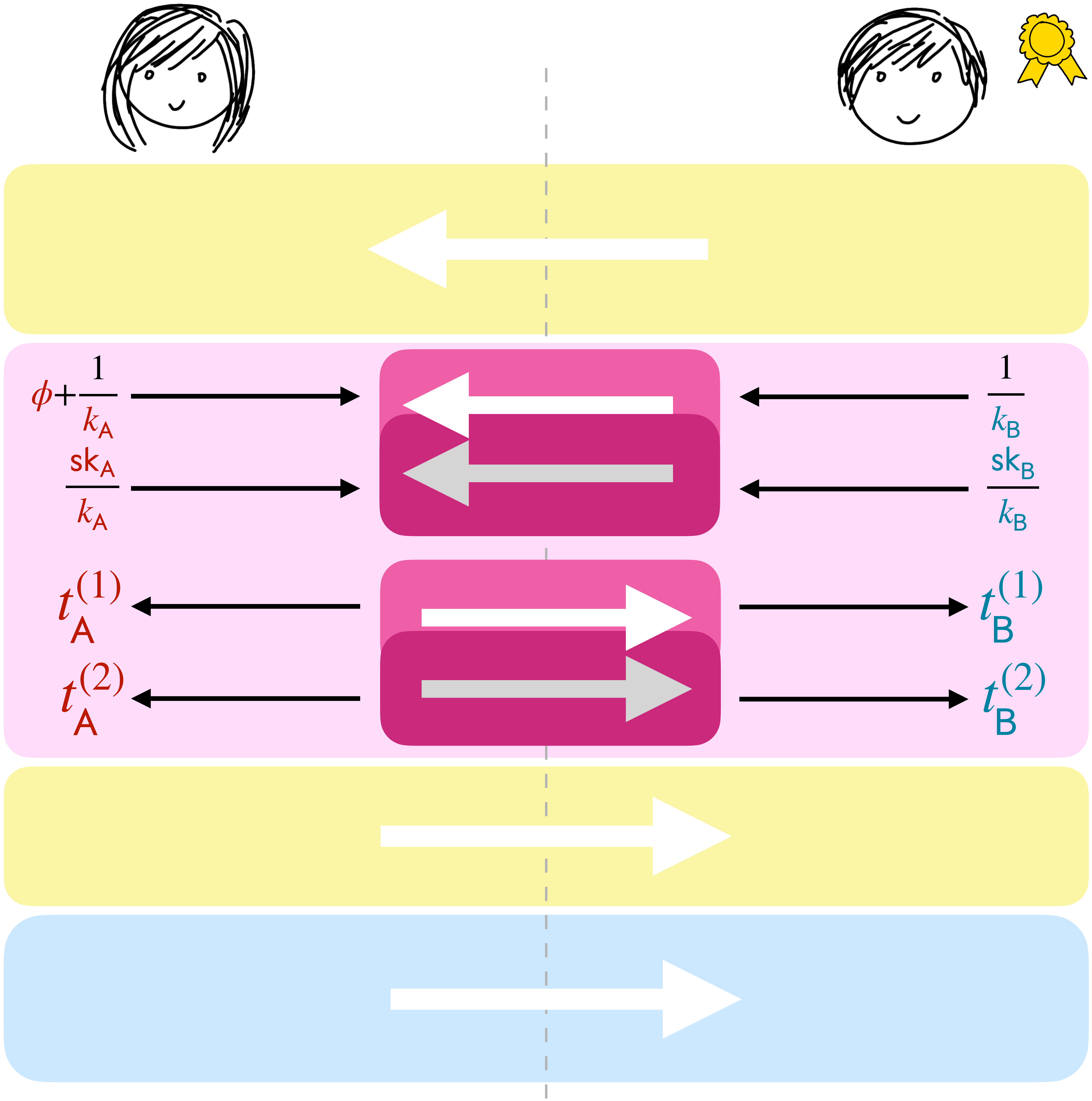
$\frac{\mathsf{sk}_\mathsf{A}}{k_\mathsf{A}}$ Mul $\frac{\mathsf{sk}_\mathsf{B}}{k_\mathsf{B}}$

$t_\mathsf{A}^{(2)}$ $t_\mathsf{B}^{(2)}$



$\phi+\frac{1}{k_A}$
$\frac{1}{k_B}$
$t_A^{(1)}$
$t_B^{(1)}$
$\frac{sk_A}{k_A}$
$\frac{sk_B}{k_B}$
$t_A^{(2)}$
$t_B^{(2)}$


$\phi + \frac{1}{k_A}$
$\frac{sk_A}{k_A}$
$\frac{1}{k_B}$
$\frac{sk_B}{k_B}$
$t_A^{(1)}$
$t_A^{(2)}$
$t_B^{(1)}$
$t_B^{(2)}$



$\phi+\frac{1}{k_\mathsf{A}}$

$\frac{\mathsf{sk}_\mathsf{A}}{k_\mathsf{A}}$

$t_\mathsf{A}^{(1)}$

$t_\mathsf{A}^{(2)}$

$\frac{1}{k_\mathsf{B}}$

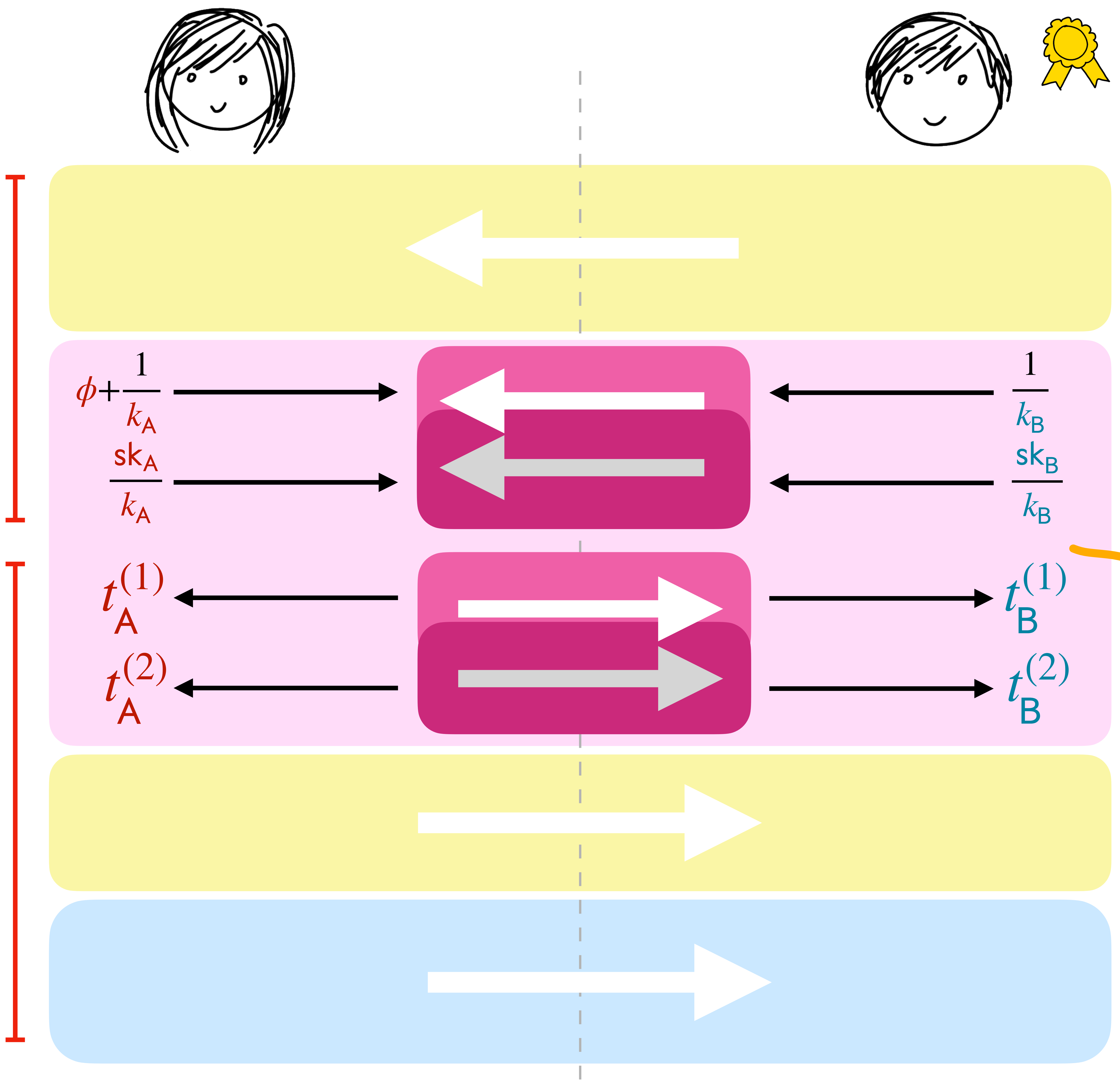
$\frac{\mathsf{sk}_\mathsf{B}}{k_\mathsf{B}}$

$t_\mathsf{B}^{(1)}$

$t_\mathsf{B}^{(2)}$

Message 1

Message 2



$\phi+\frac{1}{k_\mathrm{A}}$

$\frac{\mathrm{sk}_\mathrm{A}}{k_\mathrm{A}}$

$\frac{1}{k_\mathrm{B}}$

$\frac{\mathrm{sk}_\mathrm{B}}{k_\mathrm{B}}$

$t_\mathrm{A}^{(1)}$

$t_\mathrm{A}^{(2)}$

$t_\mathrm{B}^{(1)}$

$t_\mathrm{B}^{(2)}$

Two message protocol!

Special Case: 2-of-n

- **Key differences:**
  - Instance key *k* multiplicative (Diffie-Hellman ex.)
  - Alice has ‘final say’ for nonce *R*
  - Check messages serve as encryption keys
    - i.e. Instead of verifying $\Gamma_A + \Gamma_B = \phi$, Alice sends $\mathrm{Enc}_{\Gamma_A}(\sigma_A)$ to Bob to conditionally reveal her signature share σ_A

Conclusion

Conclusion

- **Efficient full-threshold ECDSA with fully distributed keygen**

Conclusion

- **Efficient full-threshold ECDSA with fully distributed keygen**

- **Paradigm**: ‘produce candidate shares, verify by exponent check’ costs **5 exponentiations** (+ many hashes) to sign, **no ZK** online

Conclusion

- **Efficient full-threshold ECDSA with fully distributed keygen**

- **Paradigm**: ‘produce candidate shares, verify by exponent check’ costs **5 exponentiations** (+ many hashes) to sign, **no ZK** online

- **Instantiation**: Cryptographic assumptions native to ECDSA itself (**CDH** in the same curve)

Conclusion

- **Efficient full-threshold ECDSA with fully distributed keygen**

- **Paradigm**: ‘produce candidate shares, verify by exponent check’ costs **5 exponentiations** (+ many hashes) to sign, **no ZK** online

- **Instantiation**: Cryptographic assumptions native to ECDSA itself (**CDH** in the same curve)

- Lightweight computation **but communication well within practical range (<100*t* KB/party)**

Conclusion

- **Efficient full-threshold ECDSA with fully distributed keygen**
- **Paradigm**: ‘produce candidate shares, verify by exponent check’ costs **5 exponentiations** (+ many hashes) to sign, **no ZK** online
- **Instantiation**: Cryptographic assumptions native to ECDSA itself (**CDH** in the same curve)
- Lightweight computation **but communication well within practical range (<100*t* KB/party)**
- **Wall-clock times**: Practical in realistic scenarios

Thank you!

eprint.iacr.org/2019/523