A Multiparty Computation Approach to Threshold ECDSA

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

Northeastern University

Based on work in papers from IEEE S&P 2018 and IEEE S&P 2019

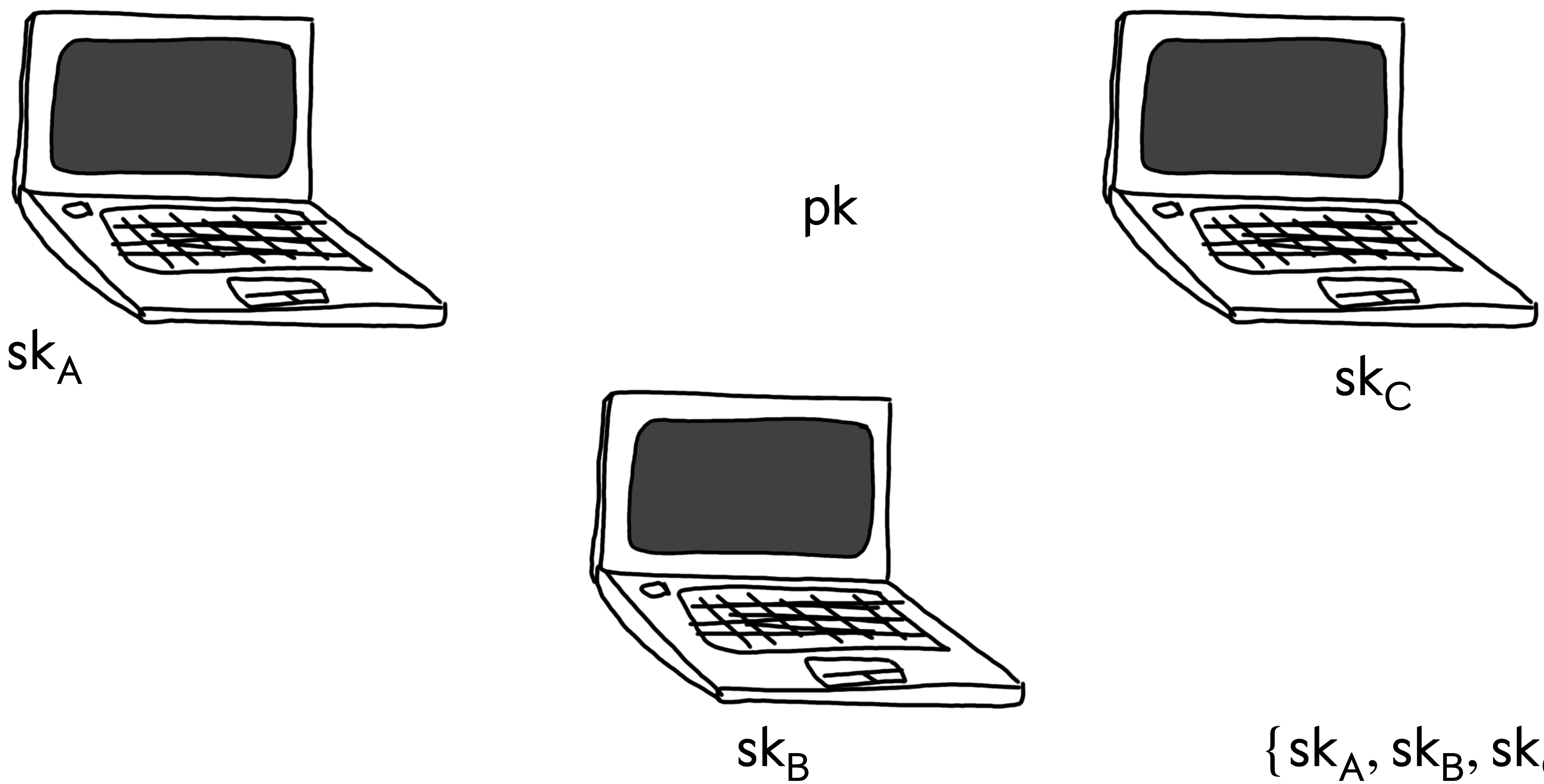
Threshold Signature

$$\mathsf{sk}\ , \mathsf{pk} \leftarrow \mathsf{Gen}(1^\kappa)$$

Threshold Signature

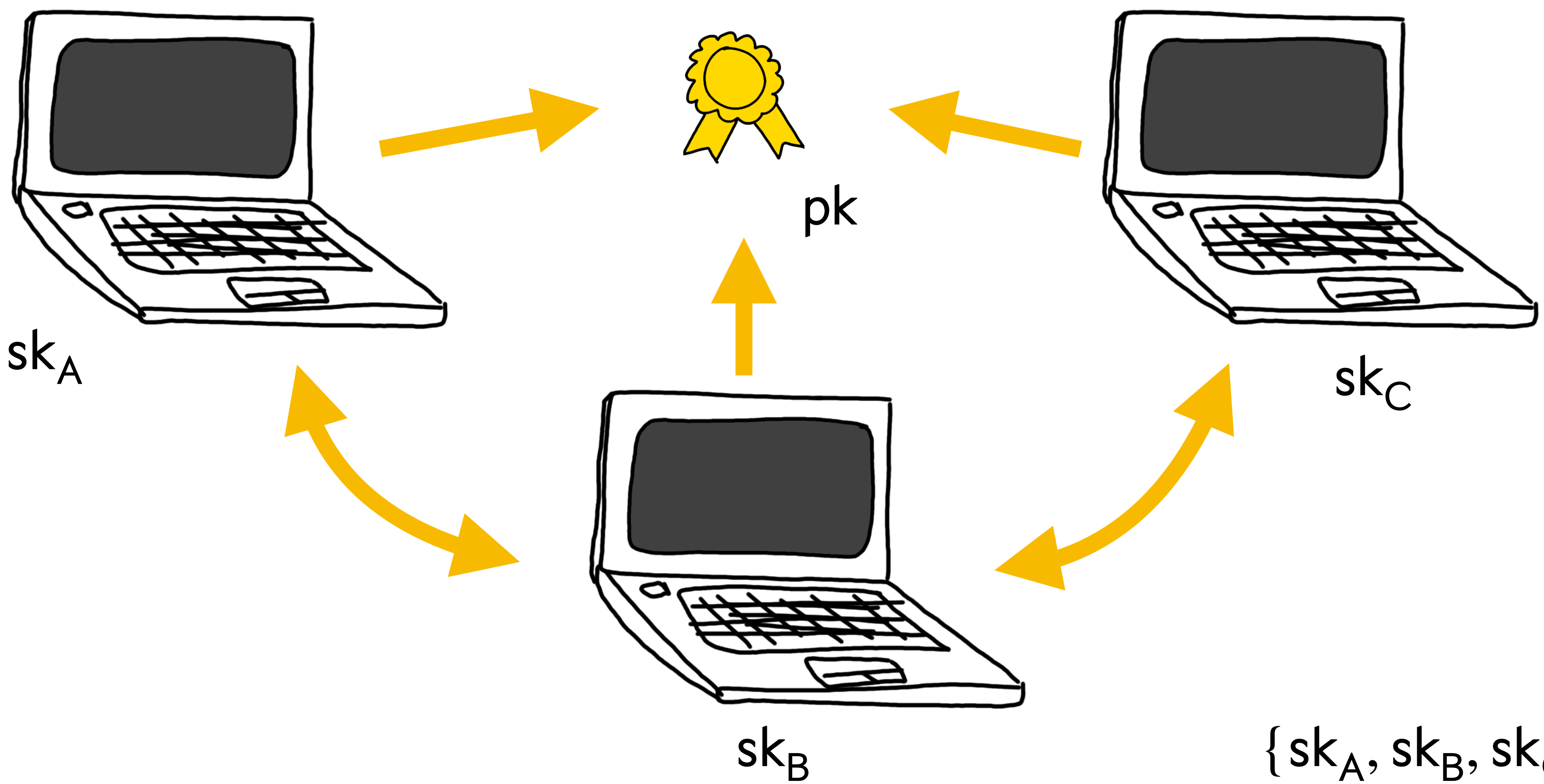
$$\{\mathsf{sk}_{\mathsf{A}}, \mathsf{sk}_{\mathsf{B}}, \mathsf{sk}_{C}\} \leftarrow \mathrm{Share}(\mathsf{sk}), \mathsf{pk} \leftarrow \mathsf{Gen}(1^{\kappa})$$

Threshold Signature



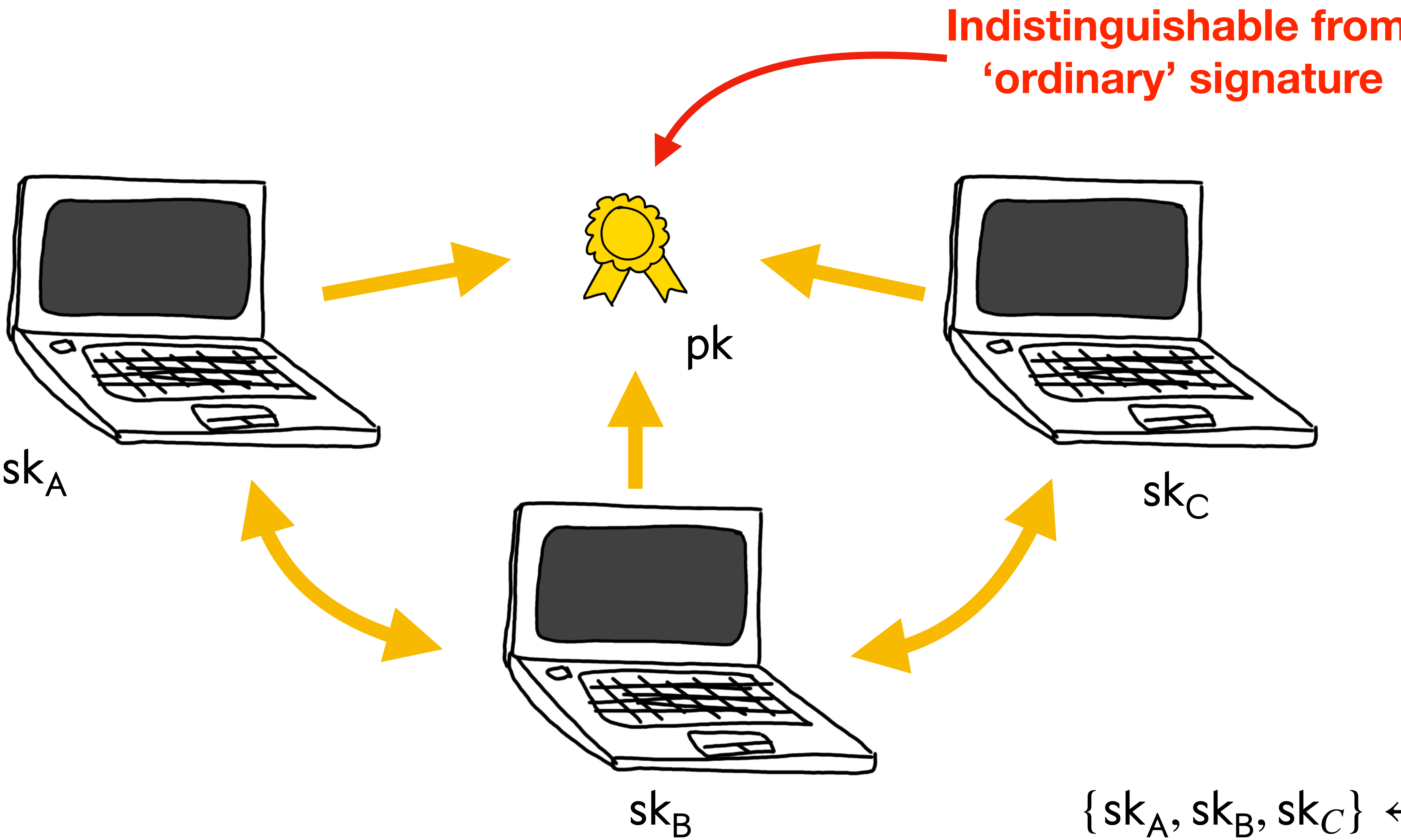
$$\{\mathsf{sk}_A, \mathsf{sk}_B, \mathsf{sk}_C\} \leftarrow \text{Share}(\mathsf{sk})$$

Threshold Signature



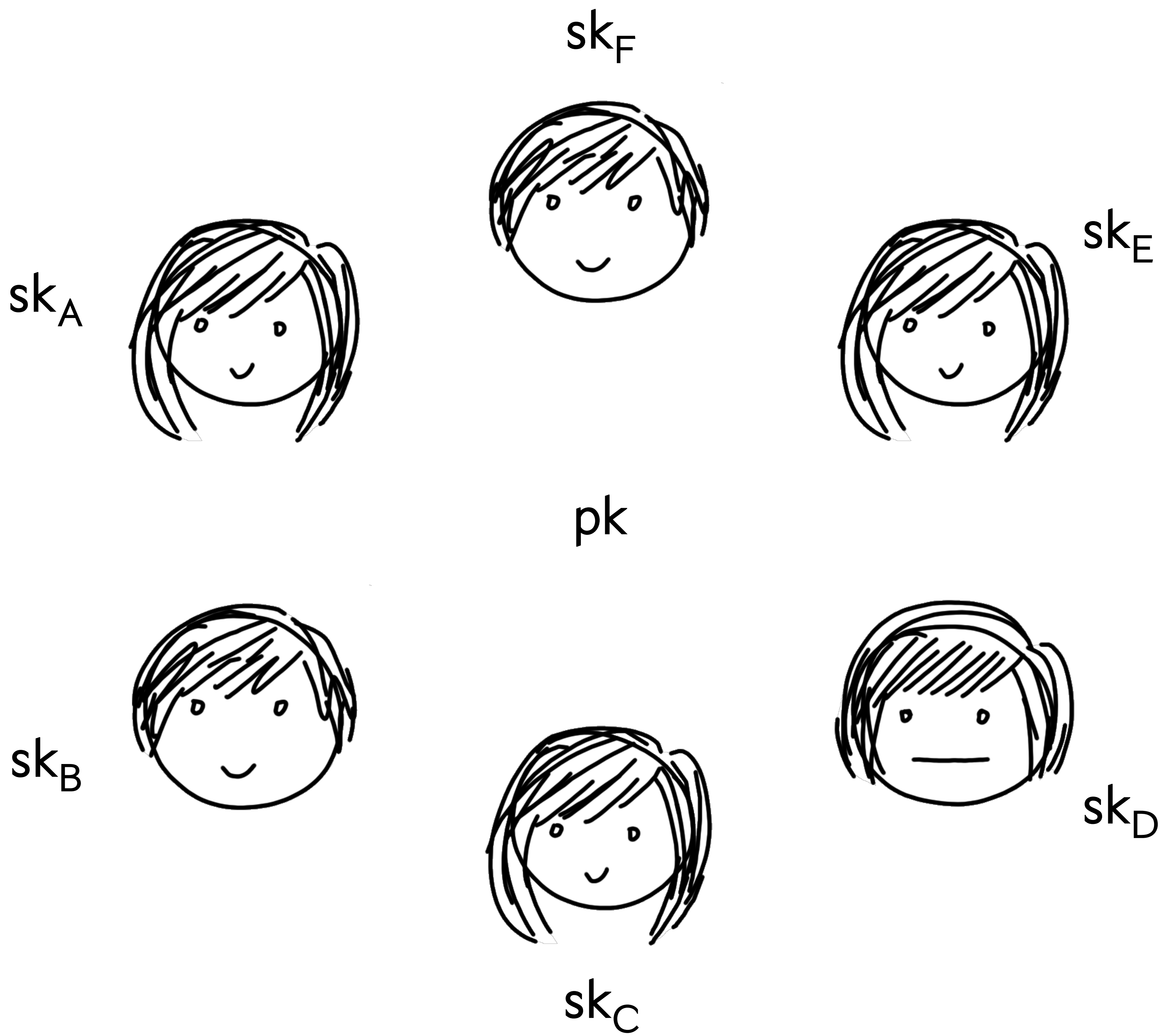
$\{\mathsf{sk}_\mathsf{A}, \mathsf{sk}_\mathsf{B}, \mathsf{sk}_C\} \leftarrow \mathrm{Share}(\mathsf{sk})$

Threshold Signature

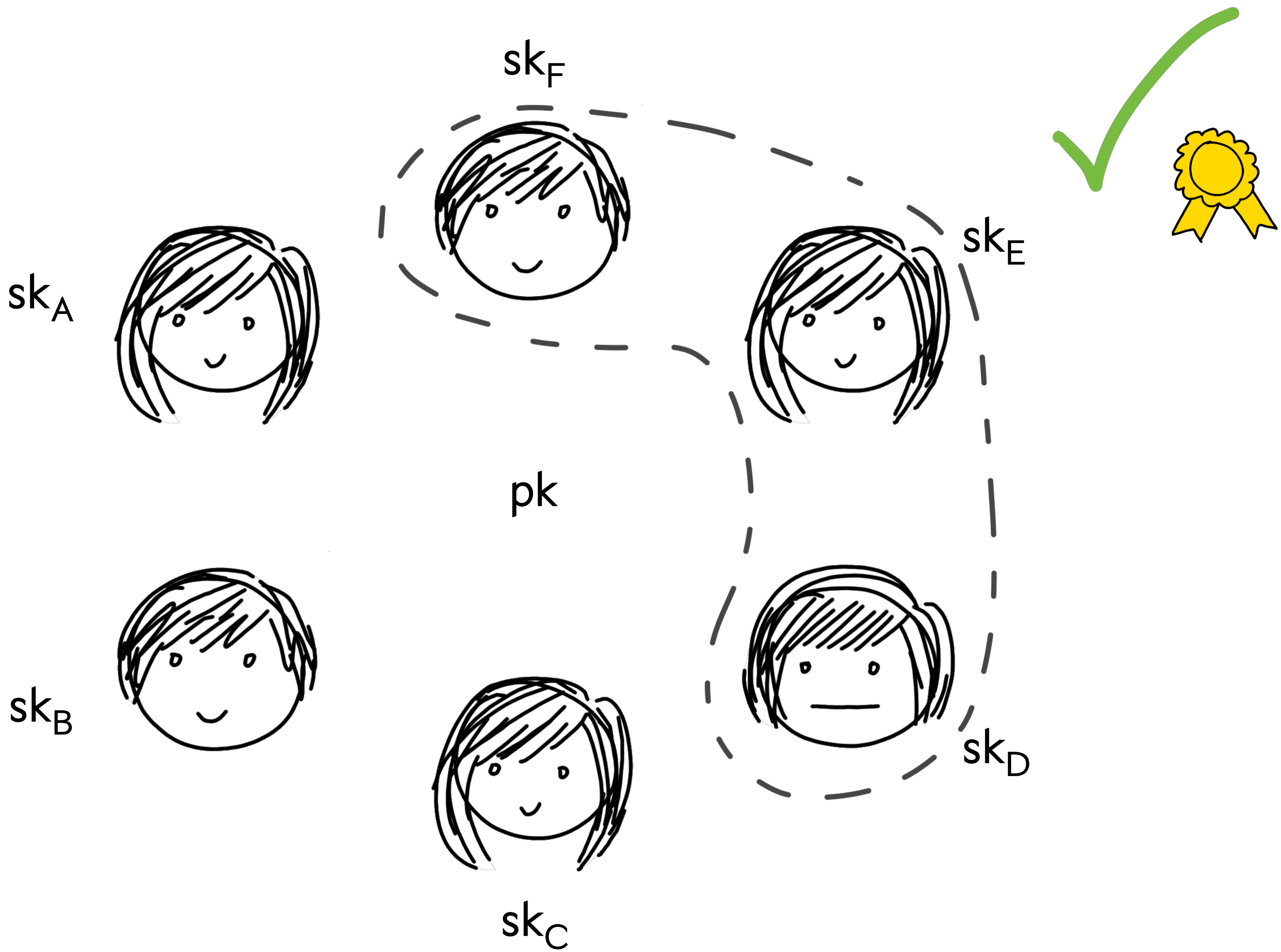


$\{\mathsf{sk}_A, \mathsf{sk}_B, \mathsf{sk}_C\} \leftarrow \mathrm{Share}(\mathsf{sk})$

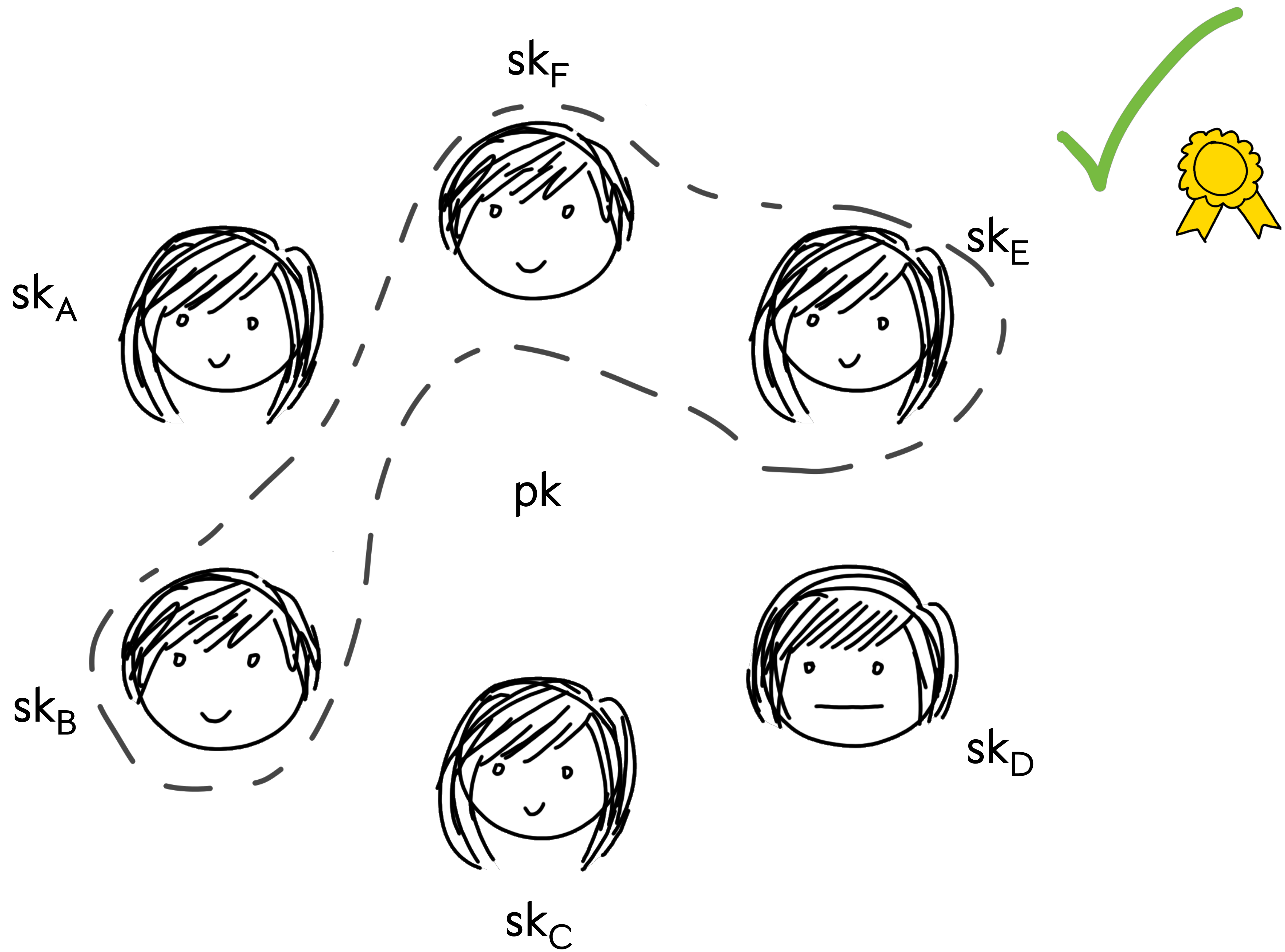
3-of-n Signature Scheme



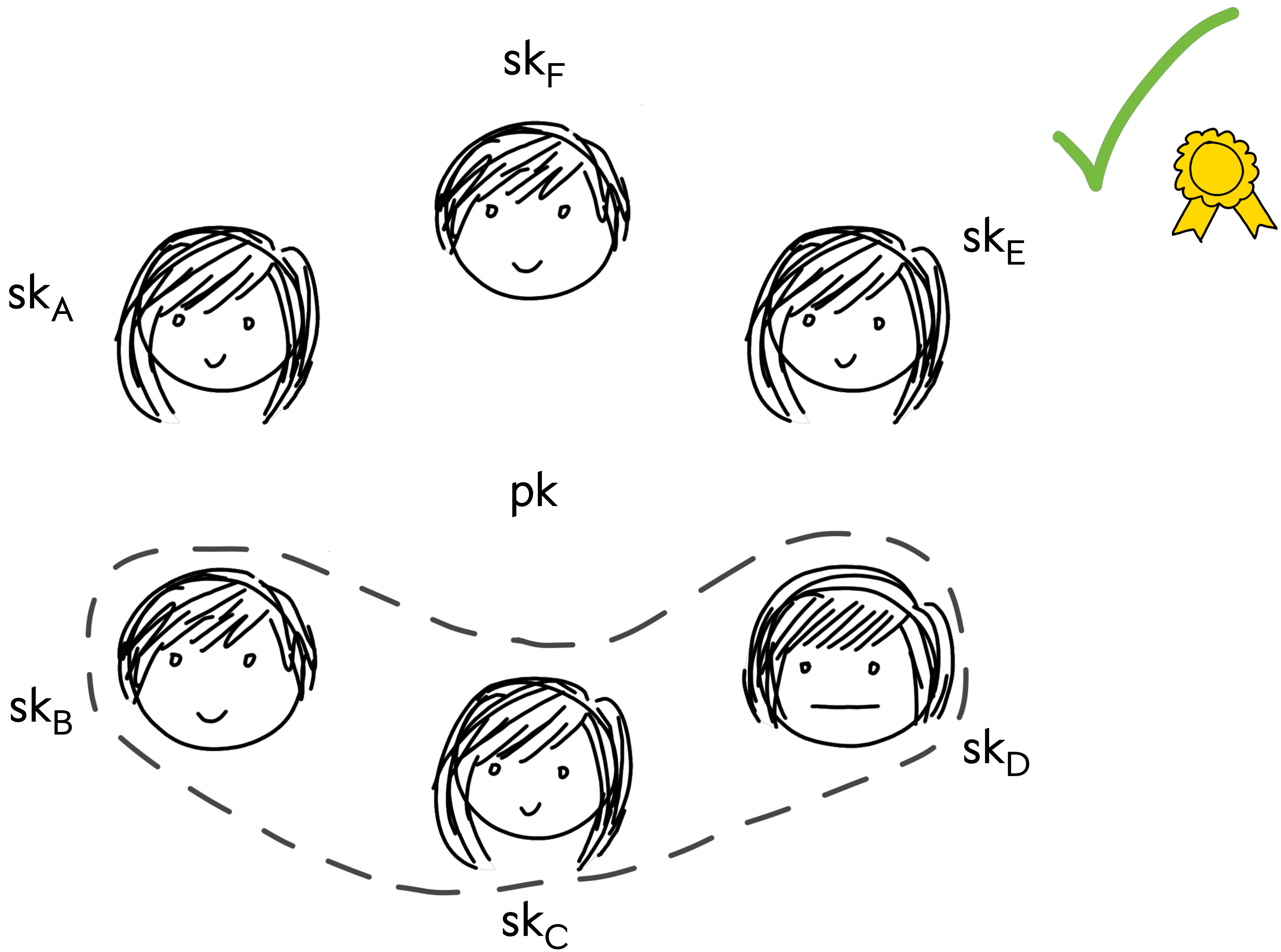
3-of-n Signature Scheme



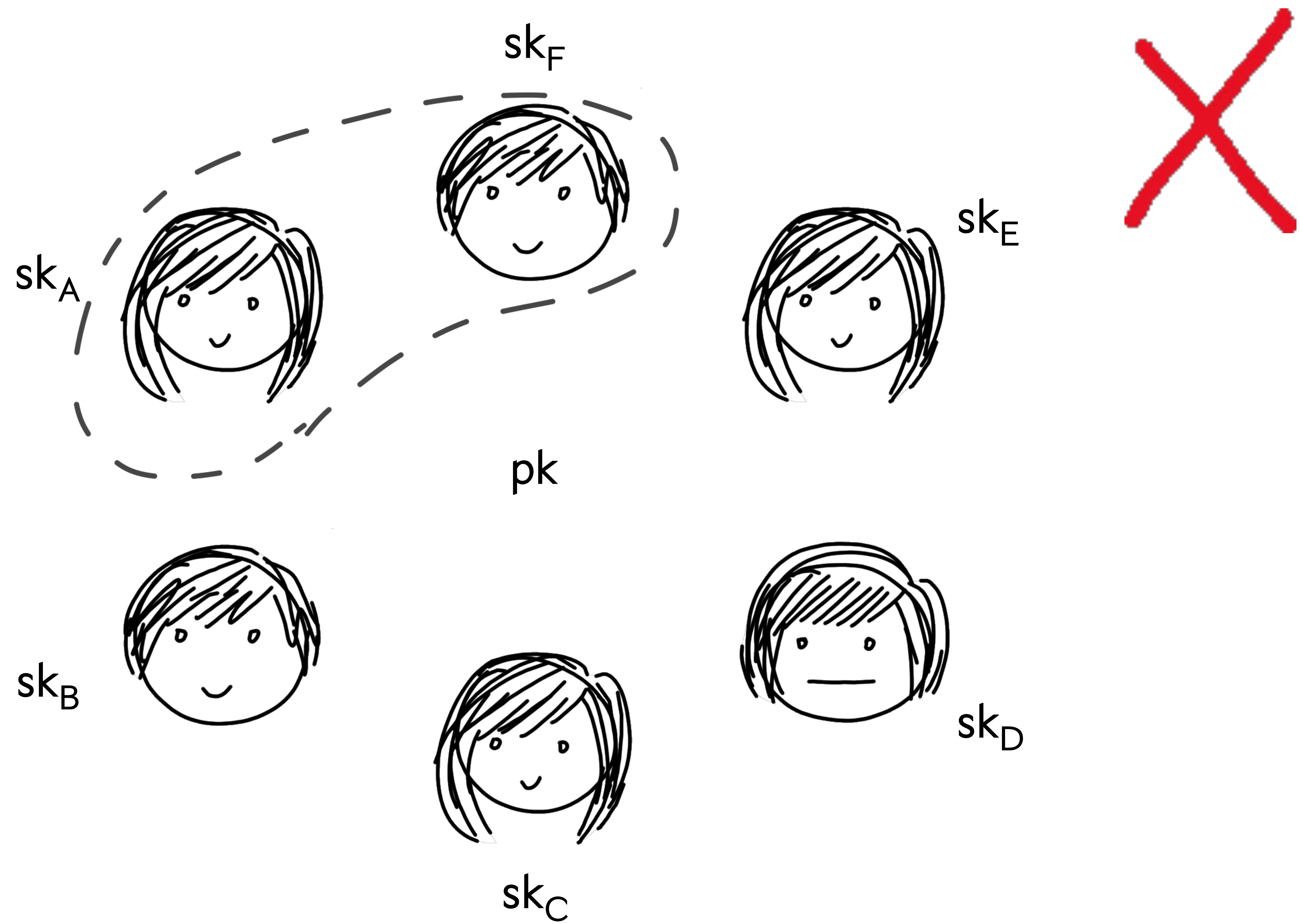
3-of-n Signature Scheme



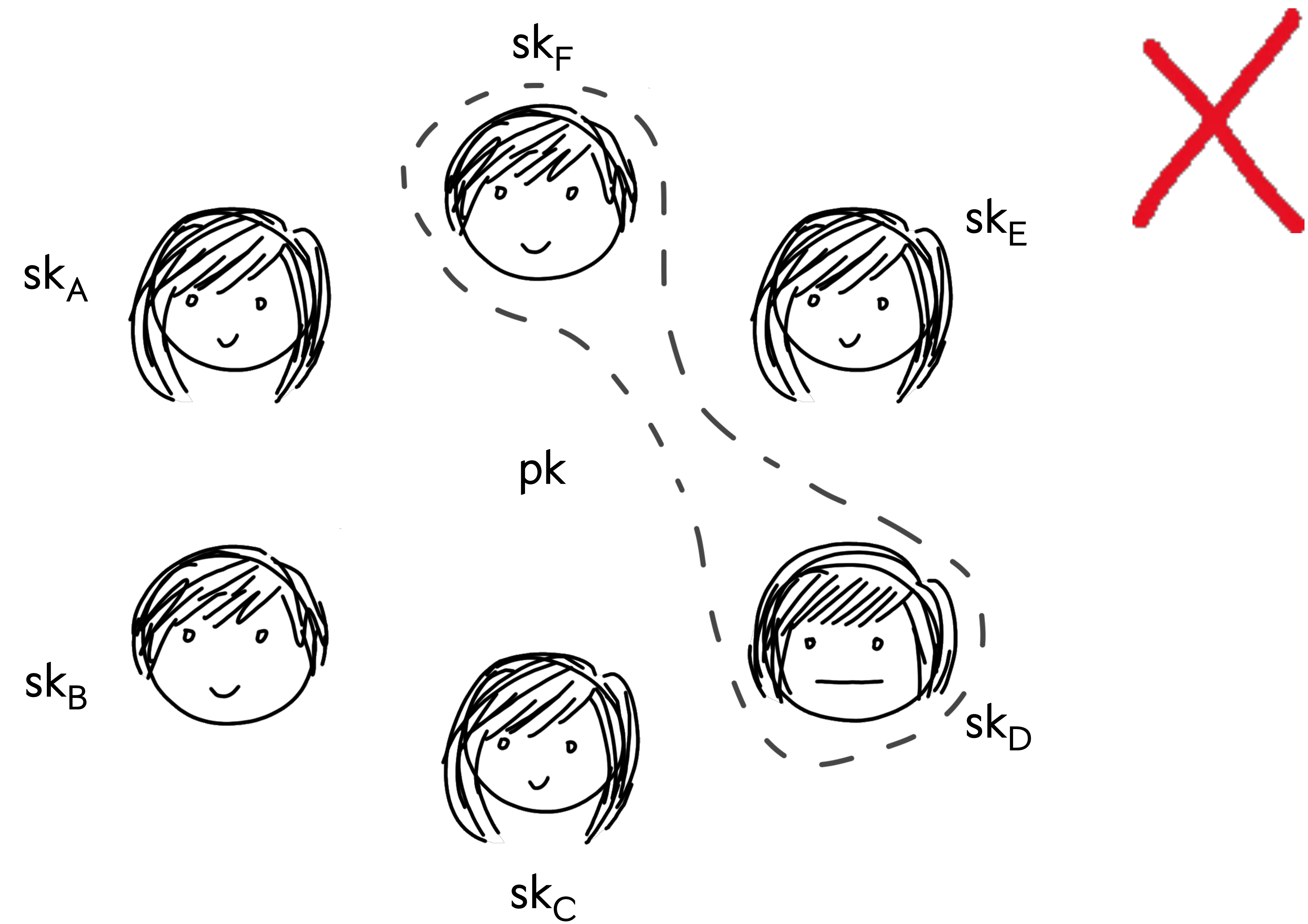
3-of-n Signature Scheme



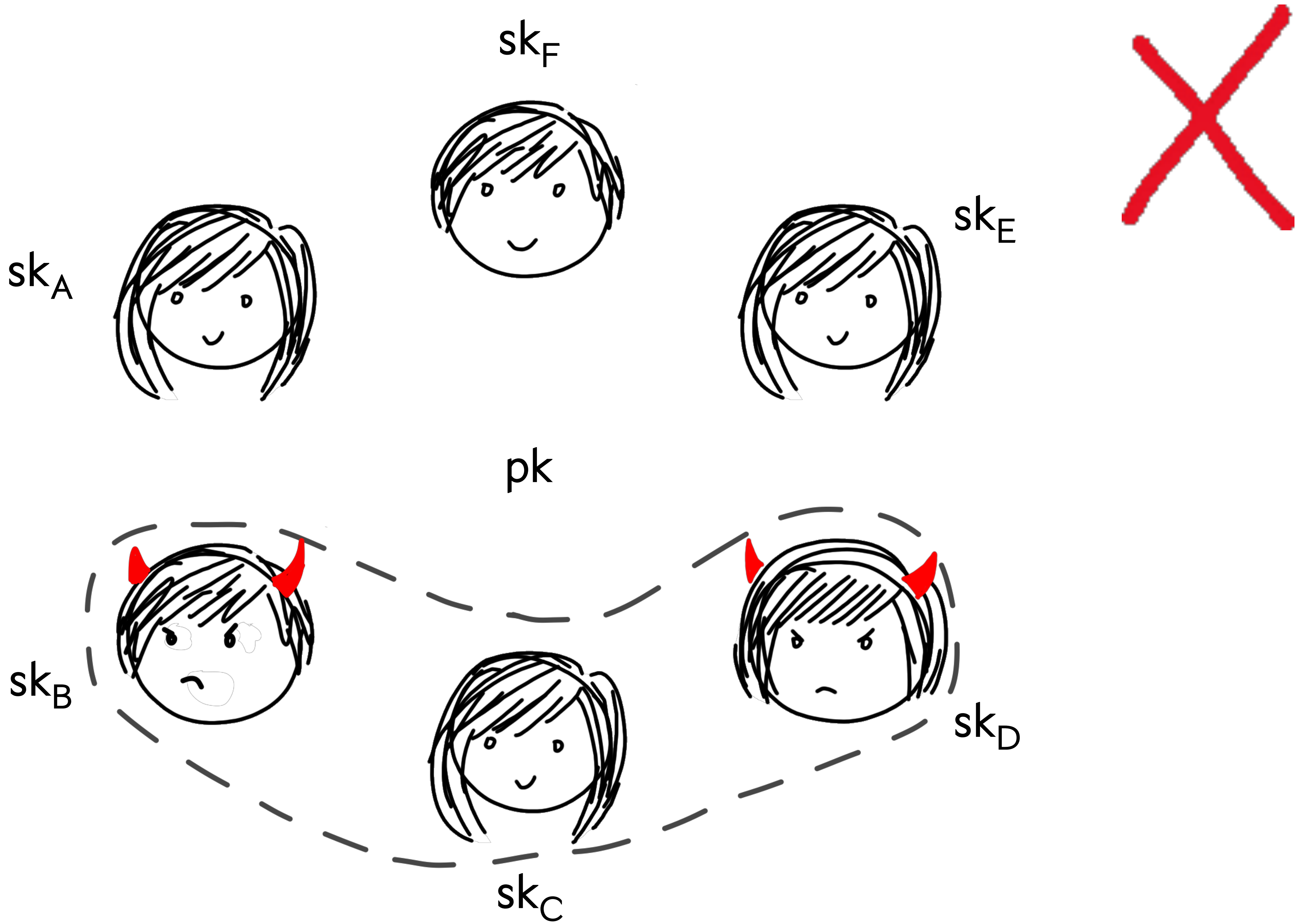
3-of-n Signature Scheme



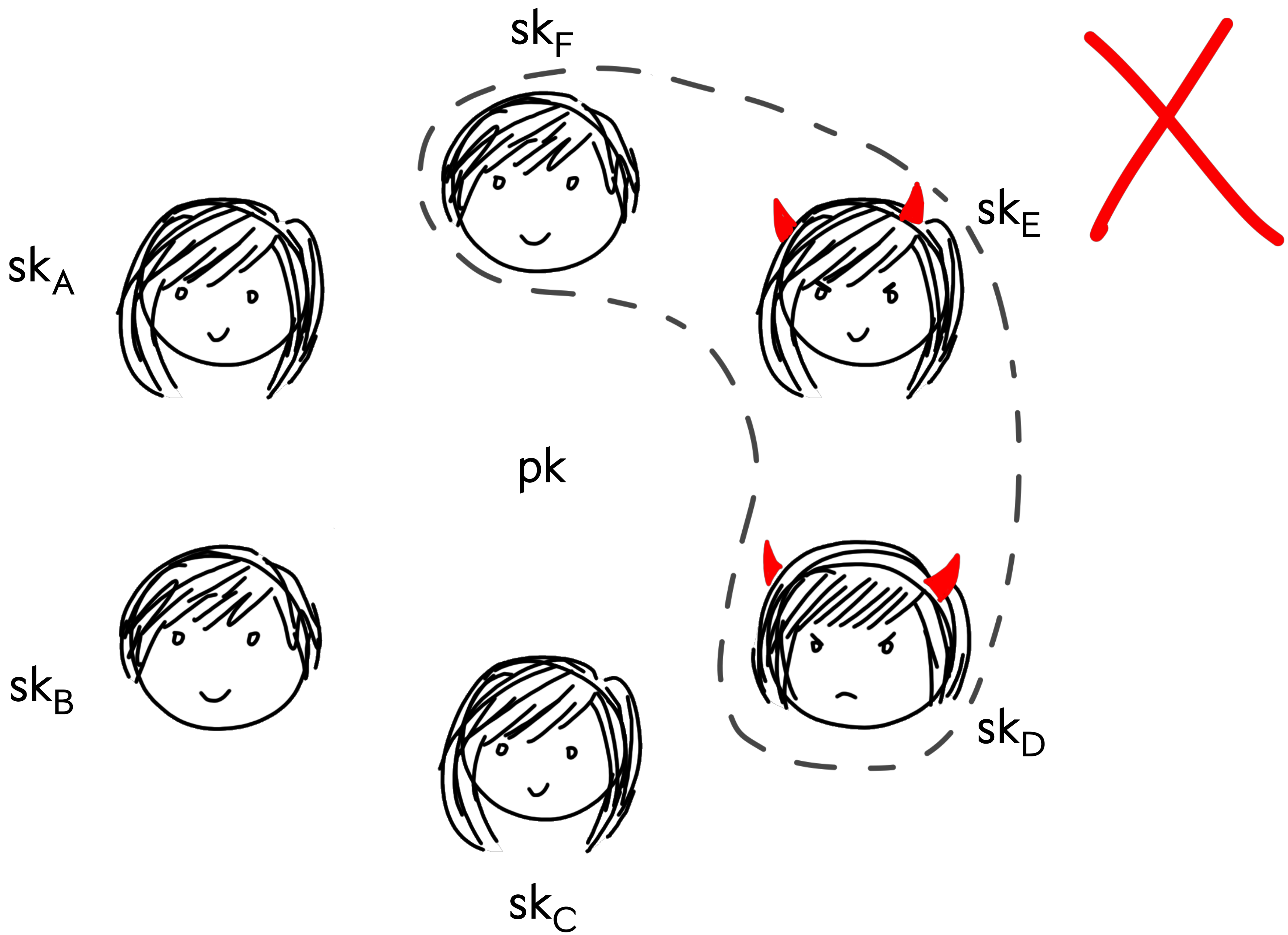
3-of-n Signature Scheme



3-of-n Signature Scheme



3-of-n Signature Scheme



Full Threshold

- Scheme can be instantiated with any $t \leq n$
- Adversary can corrupt up to t-1 parties

Notation

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

Schnorr Signatures

$$\begin{aligned}&\text{SchnorrSign}(\mathsf{sk}, m):\\&\quad k \leftarrow \mathbb{Z}_q\end{aligned}$$

Schnorr Signatures

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

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

$$R = k \cdot G$$

Schnorr Signatures

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

Schnorr Signatures

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

Schnorr Signatures

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

Schnorr Signatures

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

Linear function of *k*, sk

Threshold friendly w. linear secret sharing

ECDSA

ECDSA

- Devised by David Kravitz, standardized by NIST

ECDSA

- Devised by David Kravitz, standardized by NIST
- Widespread adoption across the internet

ECDSA

- Devised by David Kravitz, standardized by NIST
- Widespread adoption across the internet
- Used by TLS, DNSSec, SSH, Bitcoin, Ethereum, etc.

ECDSA

- Devised by David Kravitz, standardized by NIST
- Widespread adoption across the internet
- Used by TLS, DNSSec, SSH, Bitcoin, Ethereum, etc.
- Unfortunately not ‘threshold friendly’

Schnorr vs. ECDSA

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

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

Schnorr vs. ECDSA

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

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

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

$$k \leftarrow \mathbb{Z}_q$$
$$R = k \cdot G$$
$$e = H(m)$$
$$s = \frac{e}{k} + \frac{\mathsf{sk} \cdot r_x}{k}$$

Schnorr vs. ECDSA

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

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

$$R = k \cdot G$$

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

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

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

$$\text{output } \sigma$$

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

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

$$R = k \cdot G$$

$$e = H(m)$$

$$s = \frac{e}{k}$$

x – **coordinate of** R

Schnorr vs. ECDSA

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

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

$$R = k \cdot G$$

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

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

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

$$\text{output } \sigma$$

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

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

$$R = k \cdot G$$

$$e = H(m)$$

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

Bottleneck for threshold setting

Schnorr vs. ECDSA

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

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

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

- **This work**: Full-Threshold 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

- **This work**: Full-Threshold ECDSA under native assumptions
  - Low computation, practical bandwidth (100s of KB)

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

- **This work**: Full-Threshold ECDSA under native assumptions
  - Low computation, practical bandwidth (100s of KB)
  - Benchmarks: order of magnitude better wall-clock time

Our Model

Our Model

- **Universal Composability** [Canetti '01] (static adv., local RO)

Our Model

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

Our Model

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

Our Model

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

- **Functionality (trusted third party *emulated* by protocol)**:
  - Store secret key
  - Compute ECDSA signature when enough parties ask

- **Assumption**: Computational Diffie-Hellman problem is hard in the same group used by ECDSA

Our Model

- **Universal Composability** [Canetti ’01] (static adv., local RO)
- **Functionality (trusted third party *emulated* by protocol)**:
  - Store secret key
  - Compute ECDSA signature when enough parties ask
- **Assumption**: Computational Diffie-Hellman problem is hard in the same group used by ECDSA
- **Network**: Synchronous, broadcast

Our Model

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

- **Functionality (trusted third party *emulated* by protocol)**:
  - Store secret key
  - Compute ECDSA signature when enough parties ask

- **Assumption**: Computational Diffie-Hellman problem is hard in the same group used by ECDSA

- **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]

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

Setup

- **Fully distributed**
- **MUL setup**: Pairwise among parties
- **Key generation**: Every party Shamir-shares a random secret

Setup

- **Fully distributed**
- **MUL setup**: Pairwise among parties
- **Key generation**: Every party Shamir-shares a random secret
  - Secret key is sum of parties' contributions

Setup

- **Fully distributed**
- **MUL setup**: Pairwise among parties
- **Key generation**: 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

Obtaining Candidate Shares

- **Building Block**: Two party MUL with full security

Obtaining Candidate Shares

- **Building Block**: Two party MUL with full security
- **One approach** [DKLs19]:

Obtaining Candidate Shares

- **Building Block**: Two party MUL with full security
- **One approach** [DKLs19]:
  - Each party starts with multiplicative shares of k and $1/k$

Obtaining Candidate Shares

- **Building Block**: Two party MUL with full security
- **One approach** [DKLs19]:
  - 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
- **One approach** [DKLs19]:
  - 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 [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]) => Standard GMW
  3. Check relations in exponent
  4. Reconstruct sig = [$1/k$]$\cdot H(m)$+[sk/k]

Two-party MUL

Two-party MUL

- **Need:**

Two-party MUL

- **Need**:
  - Efficient single-use (not amortized) multiplication

Two-party MUL

- **Need**:
  - Efficient single-use (not amortized) multiplication
  - Assumptions in the same curve as ECDSA

Two-party MUL

- **Need**:
  - Efficient single-use (not amortized) multiplication
  - Assumptions in the same curve as ECDSA
- **[Gilboa ’99]**: semi-honest MUL based on OT

Two-party MUL

- **Need**:
  - Efficient single-use (not amortized) multiplication
  - Assumptions in the same curve as ECDSA
- **[Gilboa ’99]**: semi-honest MUL based on OT
- We harden to **full malicious security** in the RO model

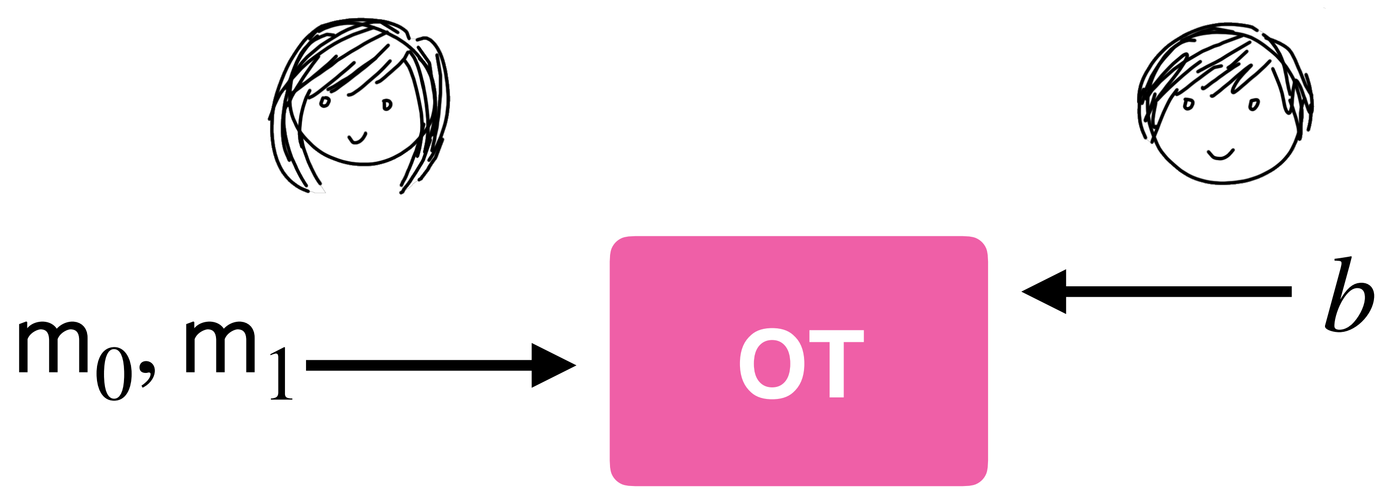
Oblivious Transfer

m_0, m_1

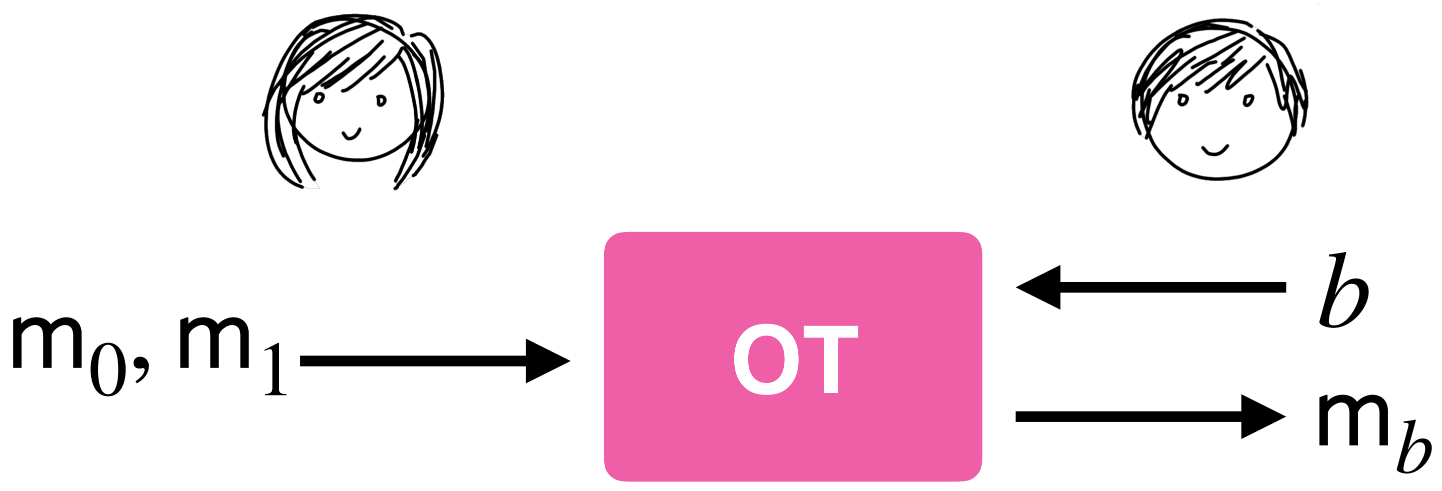
OT

b

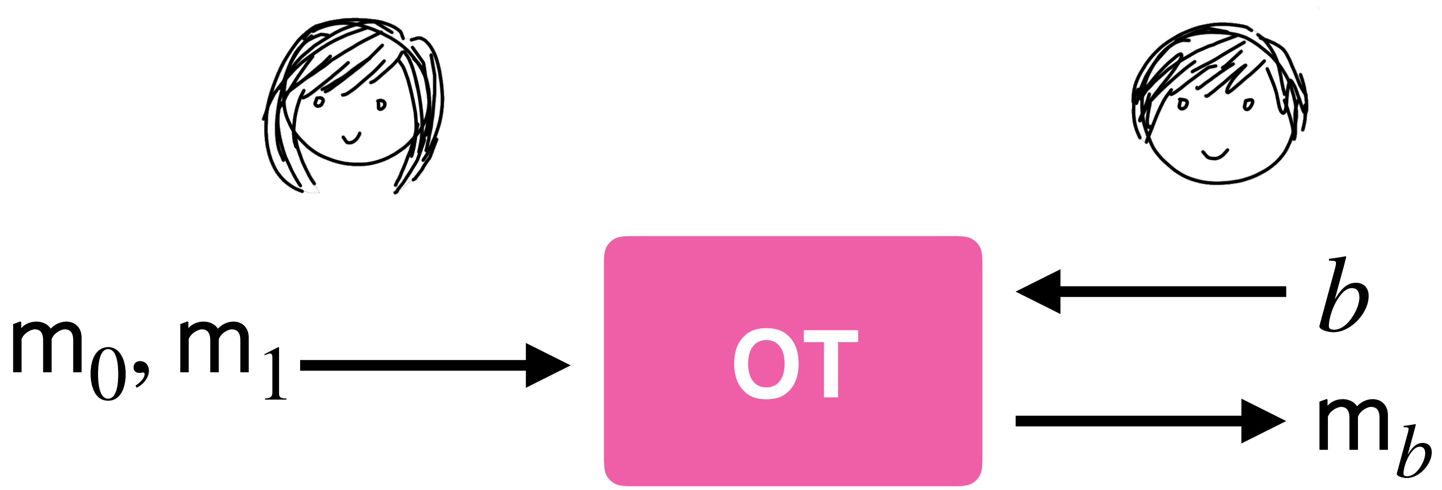
Oblivious Transfer



Oblivious Transfer

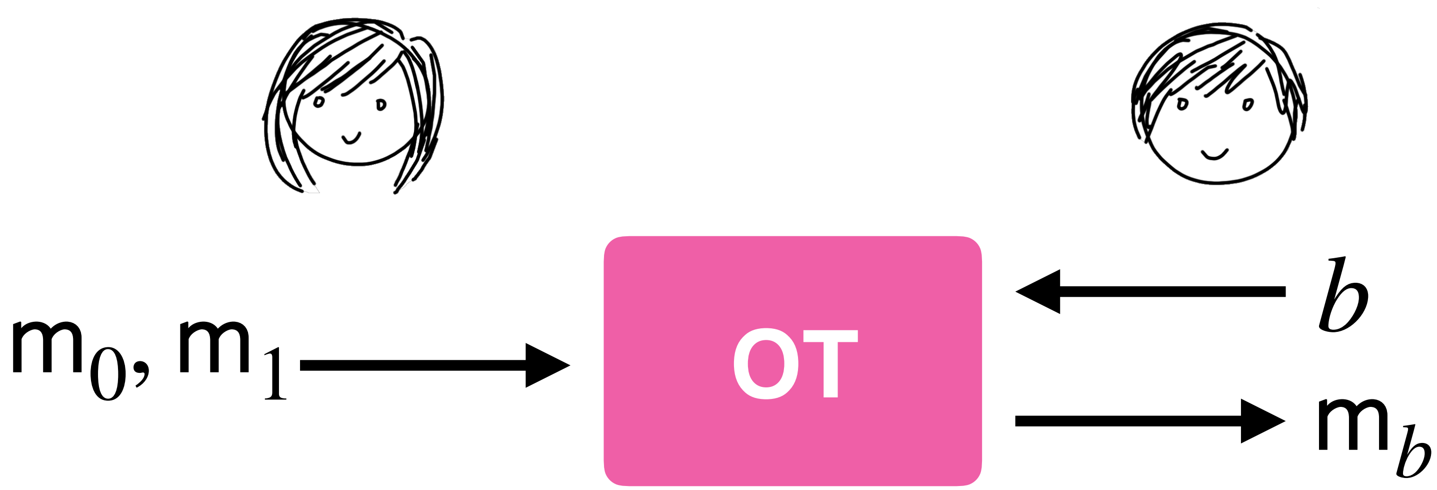


Oblivious Transfer



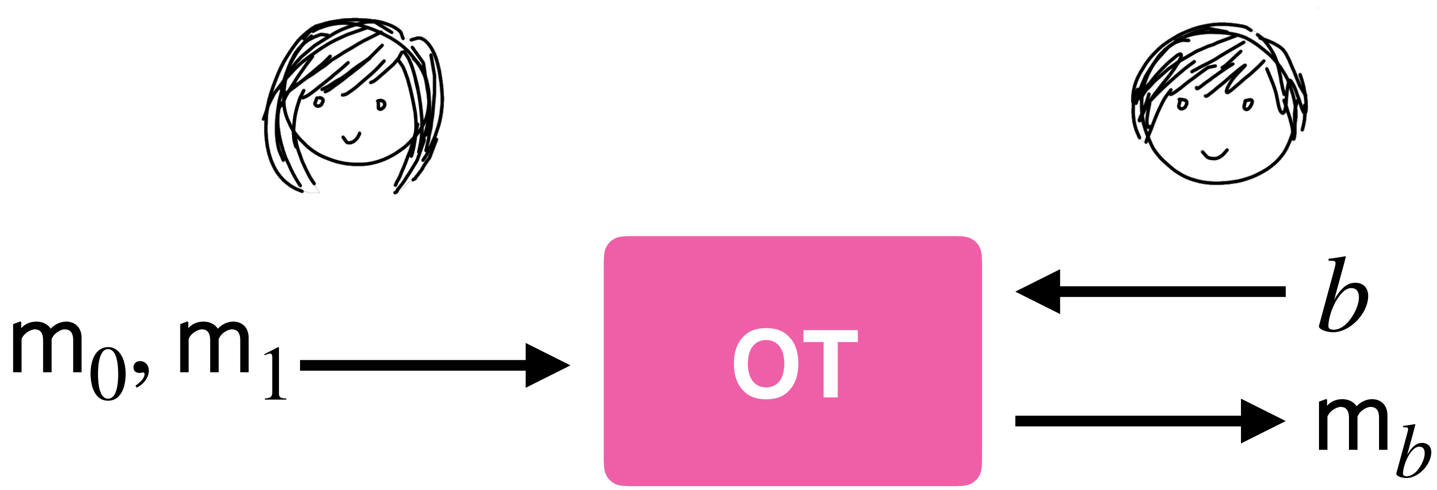
- **Instantiation:** “Verified” Simplest Oblivious Transfer [Chou&Orlandi15]

Oblivious Transfer



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

Oblivious Transfer



- **Instantiation:** "Verified" Simplest Oblivious Transfer [Chou&Orlandi15]
- UC-secure (RO model) assuming CDH in the same group as ECDSA
- **OT Extension**: [Keller Orsini Scholl '15]

Malicious OT-MUL

Malicious OT-MUL

- Gil99 already secure against malicious Bob

Malicious OT-MUL

- Gil99 already secure against malicious Bob
- Security against malicious Alice:

Malicious OT-MUL

- Gil99 already secure against malicious Bob
- Security against malicious Alice:
  - **Selective Failure**: Bob uses high-entropy encoding of input

Malicious OT-MUL

- Gil99 already secure against malicious Bob
- Security against malicious Alice:
  - **Selective Failure**: Bob uses high-entropy encoding of input
  - **Input consistency**: Alice is challenged to reveal a linear combination of her (masked) inputs

Malicious OT-MUL

- Gil99 already secure against malicious Bob
- Security against malicious Alice:
  - **Selective Failure**: Bob uses high-entropy encoding of input
  - **Input consistency**: Alice is challenged to reveal a linear combination of her (masked) inputs
- **Minimally interactive**: two messages

Malicious OT-MUL

- Gil99 already secure against malicious Bob
- Security against malicious Alice:
  - **Selective Failure**: Bob uses high-entropy encoding of input
  - **Input consistency**: Alice is challenged to reveal a linear combination of her (masked) inputs
- **Minimally interactive**: two messages
- **Overhead**: ~6x, room for improvement

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]

Check in Exponent

Check in Exponent

- **Three** sharings [*k*],[1/*k*],[sk/*k*] to verify consistency

Check in Exponent

- **Three** sharings [k],[$1/k$],[sk/k] to verify consistency
- **Technique**: Each equation is verified in the exponent, using ‘auxiliary’ information that’s already available

Check in Exponent

- **Three** sharings [k],[$1/k$],[sk/k] to verify consistency

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

- **Cost**: $5+t$ exponentiations, 5 group elements per party

Check in Exponent

- There are **three** relations that have to be verified

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

Check in Exponent

- There are **three** relations that have to be verified

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

Check in Exponent

- There are **three** relations that have to be verified

$$
\begin{tikzcd}
{[k]} \arrow[r, bend left, "R"] & \left[\dfrac{1}{k}\right] & \left[\dfrac{\mathsf{sk}}{k}\right]
\end{tikzcd}
$$

Check in Exponent

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

- **Auxiliary information**: '*k*' in the exponent, ie. $R=k \cdot G$

- **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

Adversary's contribution

Attempt at a solution:

Honest Party's contribution

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

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

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

Check in Exponent

Adversary's contribution

Honest Party's contribution

Attempt at a solution:

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

Check in Exponent

Adversary's contribution

Attempt at a solution:

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 $$R = k_A k_h \cdot G$$

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

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

Easy for Adv. to offset

Check in Exponent

Check in Exponent

- Each party commits ϕ_i before MUL

Check in Exponent

- Each party commits ϕ_i before MUL
- Define $\phi = \prod_{i \in [n]} \phi_i$

Check in Exponent

- Each party commits ϕ_i before MUL
- Define $\phi = \prod_{i \in [n]} \phi_i$
- Randomize inversion: compute $\left[\frac{\phi}{k}\right]$ instead of $\left[\frac{1}{k}\right]$

Check in Exponent

- Each party commits ϕ_i before MUL
- Define $\phi = \prod_{i \in [n]} \phi_i$
- Randomize inversion: compute $\left[\dfrac{\phi}{k}\right]$ instead of $\left[\dfrac{1}{k}\right]$
- Reveal ϕ only after *every* other value is committed

Check in Exponent

Adversary's contribution

Honest Party's contribution

Attempt at a solution:

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

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

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

Check in Exponent

Adversary's contribution

Attempt at a solution:

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 $R = k_A k_h \cdot G$

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

Adversary's contribution

Attempt at a solution:

Honest Party's contribution

Public

$$R = k_A k_h \cdot G$$

Broadcast

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

Verify

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

Check in Exponent

Adversary's contribution

Attempt at a solution:

Honest Party's contribution

Public

$$R = k_A k_h \cdot G$$

Broadcast

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

Verify

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

Completely unpredictable

Check in Exponent

- There are **three** relations that have to be verified

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

R

Check in Exponent

- There are **three** relations that have to be verified

$$
\begin{tikzcd}
{[k]} \arrow[r, dash, bend left, "R"] & \left[\dfrac{1}{k}\right] \arrow[r, dash, bend left, "\mathrm{pk}"] & \left[\dfrac{\mathrm{sk}}{k}\right]
\end{tikzcd}
$$

Check in Exponent

- There are **three** relations that have to be verified

$$
\begin{tikzcd}
{[k]} \arrow[r, bend left, "R"] \arrow[rr, bend right, "{R, \mathsf{pk}}"'] & \left[\dfrac{1}{k}\right] \arrow[r, bend left, "\mathsf{pk}"] & \left[\dfrac{\mathsf{sk}}{k}\right]
\end{tikzcd}
$$

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

Dominant Costs

Dominant Costs

- **Setup:**

Dominant Costs

- **Setup**:
  - OTE Setup (128 OTs) pairwise, broadcast PoK (DLog)

Dominant Costs

- **Setup**:
  - OTE Setup (128 OTs) pairwise, broadcast PoK (DLog)
  - 5 rounds

Dominant Costs

- **Setup**:
  - OTE Setup (128 OTs) pairwise, broadcast PoK (DLog)
  - 5 rounds
  - Concretely ~21*n* KB (transmitted), ~520*n* exp/party

Dominant Costs

- **Setup**:
  - OTE Setup (128 OTs) pairwise, broadcast PoK (DLog)
  - 5 rounds
  - Concretely ~21n KB (transmitted), ~520n exp/party
- **Signing**:

Dominant Costs

- **Setup**:
  - OTE Setup (128 OTs) pairwise, broadcast PoK (DLog)
  - 5 rounds
  - Concretely ~21*n* KB (transmitted), ~520*n* exp/party
- **Signing**:
  - log(*t*)+6 rounds, constant round version in progress

Dominant Costs

- **Setup**:
  - OTE Setup (128 OTs) pairwise, broadcast PoK (DLog)
  - 5 rounds
  - Concretely ~21*n* KB (transmitted), ~520*n* exp/party
- **Signing**:
  - log(*t*)+6 rounds, constant round version in progress
  - Concretely ~65*t* KB (transmitted), 5+*t* exp/party

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

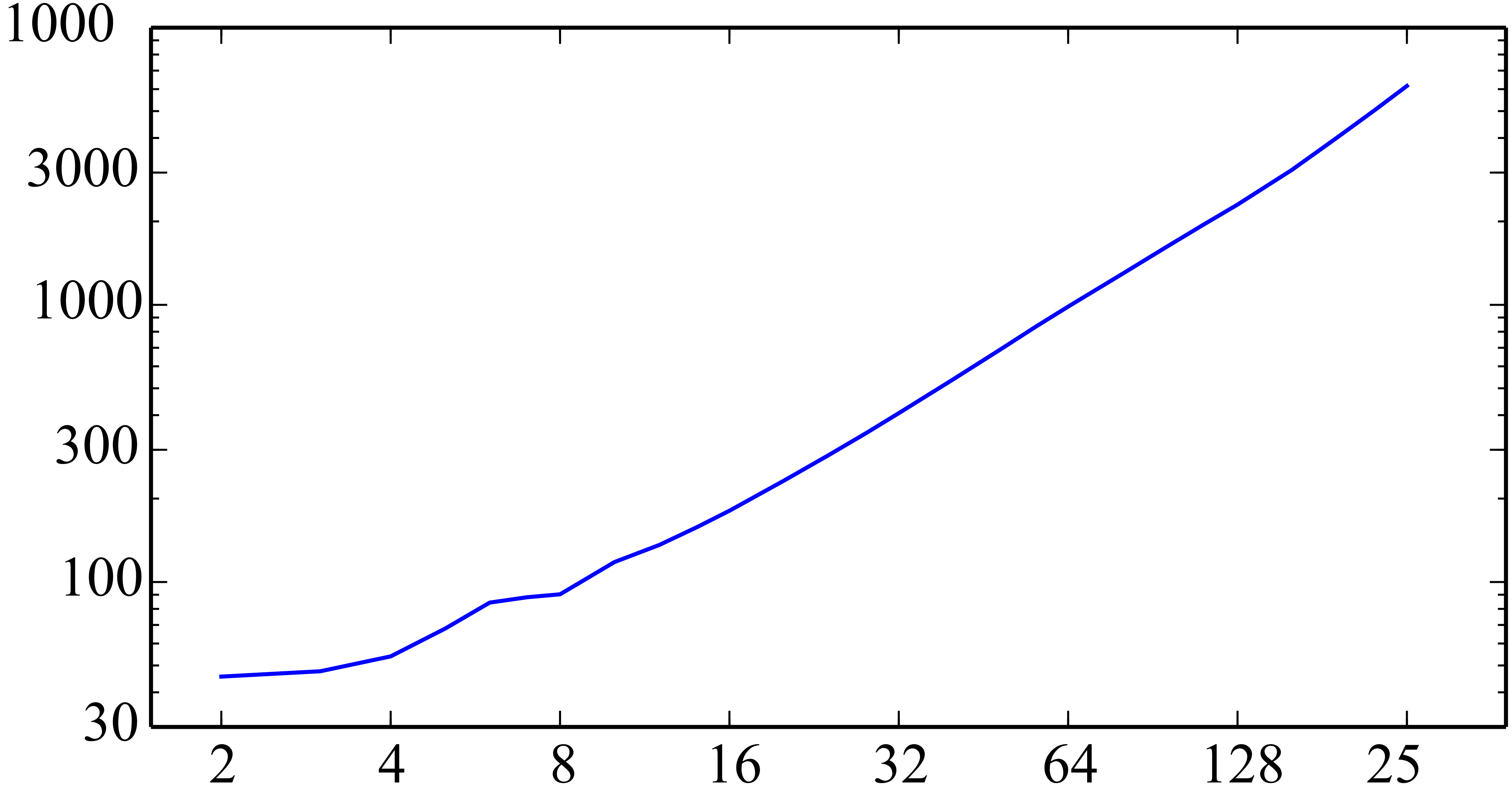
Benchmarks

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

Benchmarks

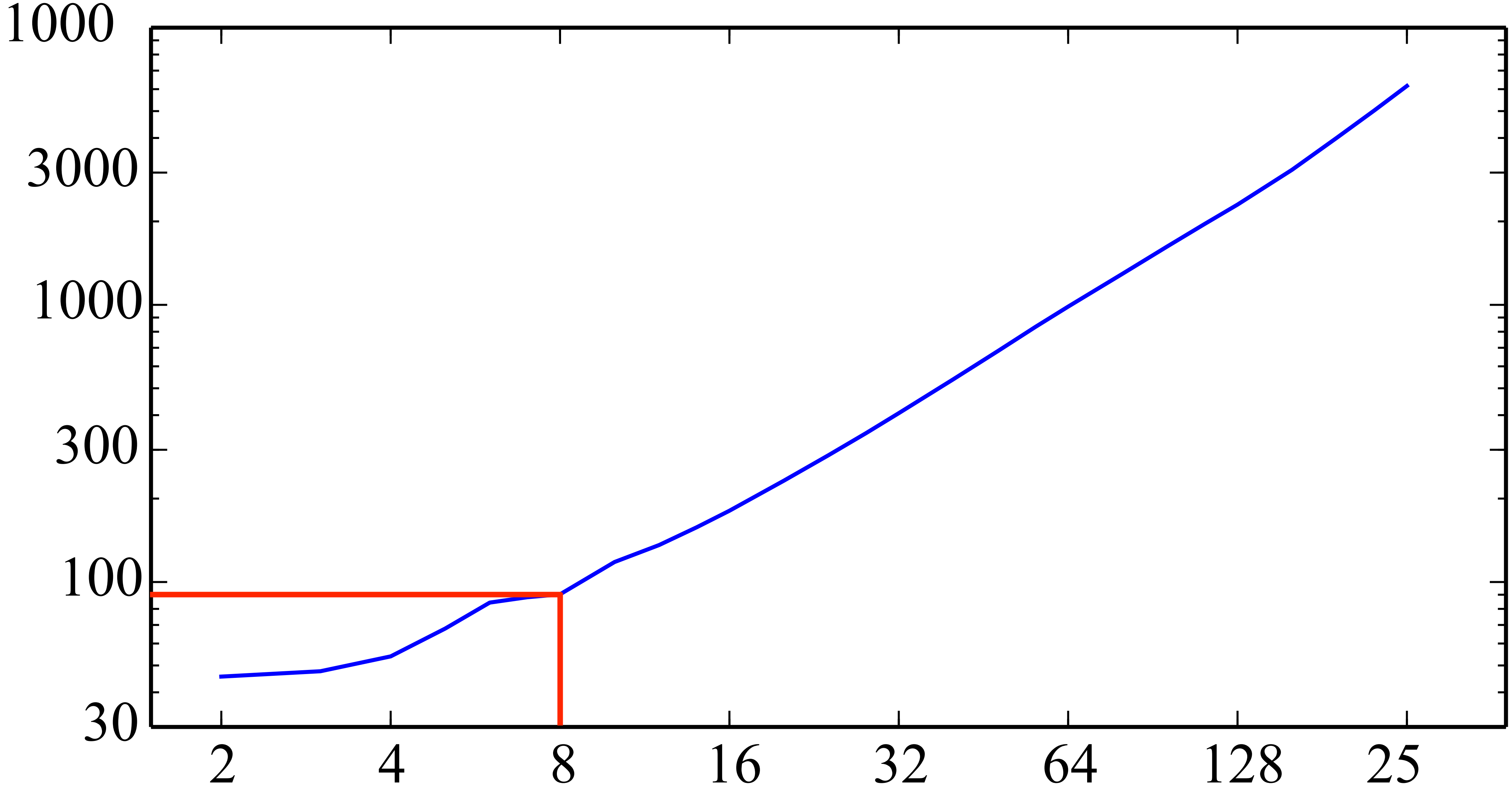
- Implementation in **Rust**
- Ran benchmarks on Google Cloud
- One node per party
- **LAN** and **WAN** tests (up to **16 zones**)
- **Low Power Friendliness**: Raspberry Pi benchmark (~60ms for 2-of-2)

LAN Setup



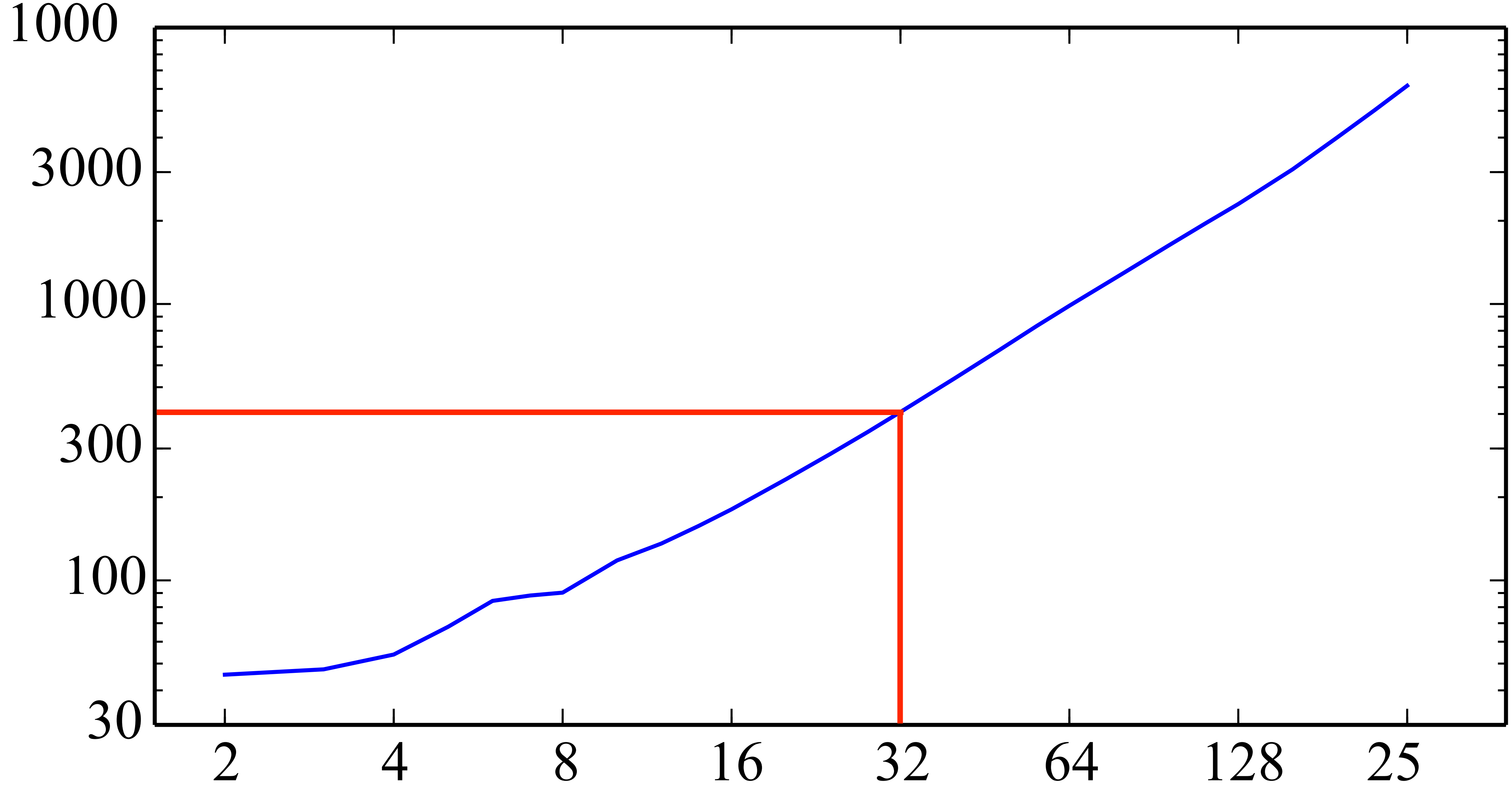
Broadcast PoK (DLog), **Pairwise**: 128 OTs

LAN Setup



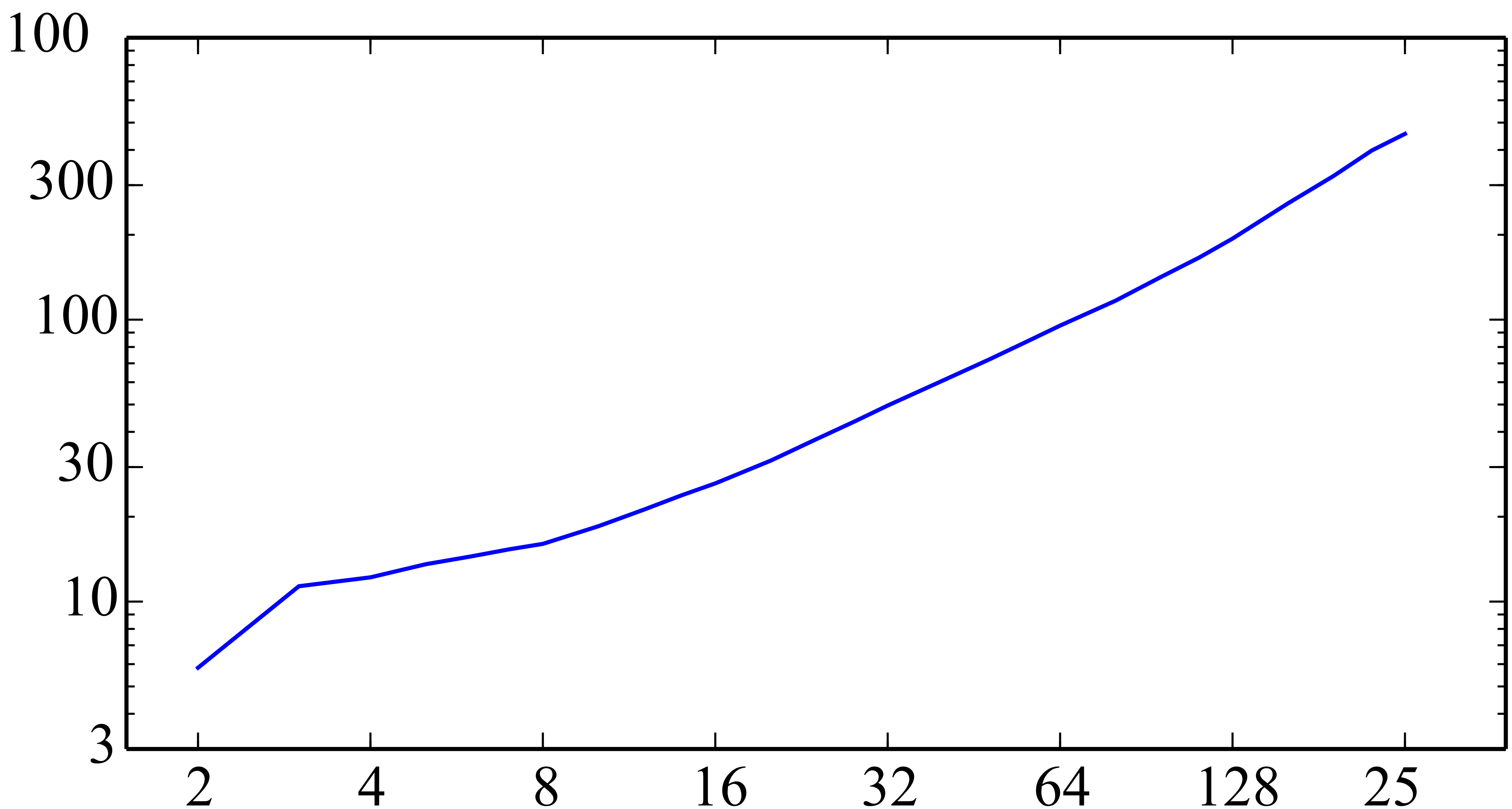
Broadcast PoK (DLog), **Pairwise**: 128 OTs

LAN Setup

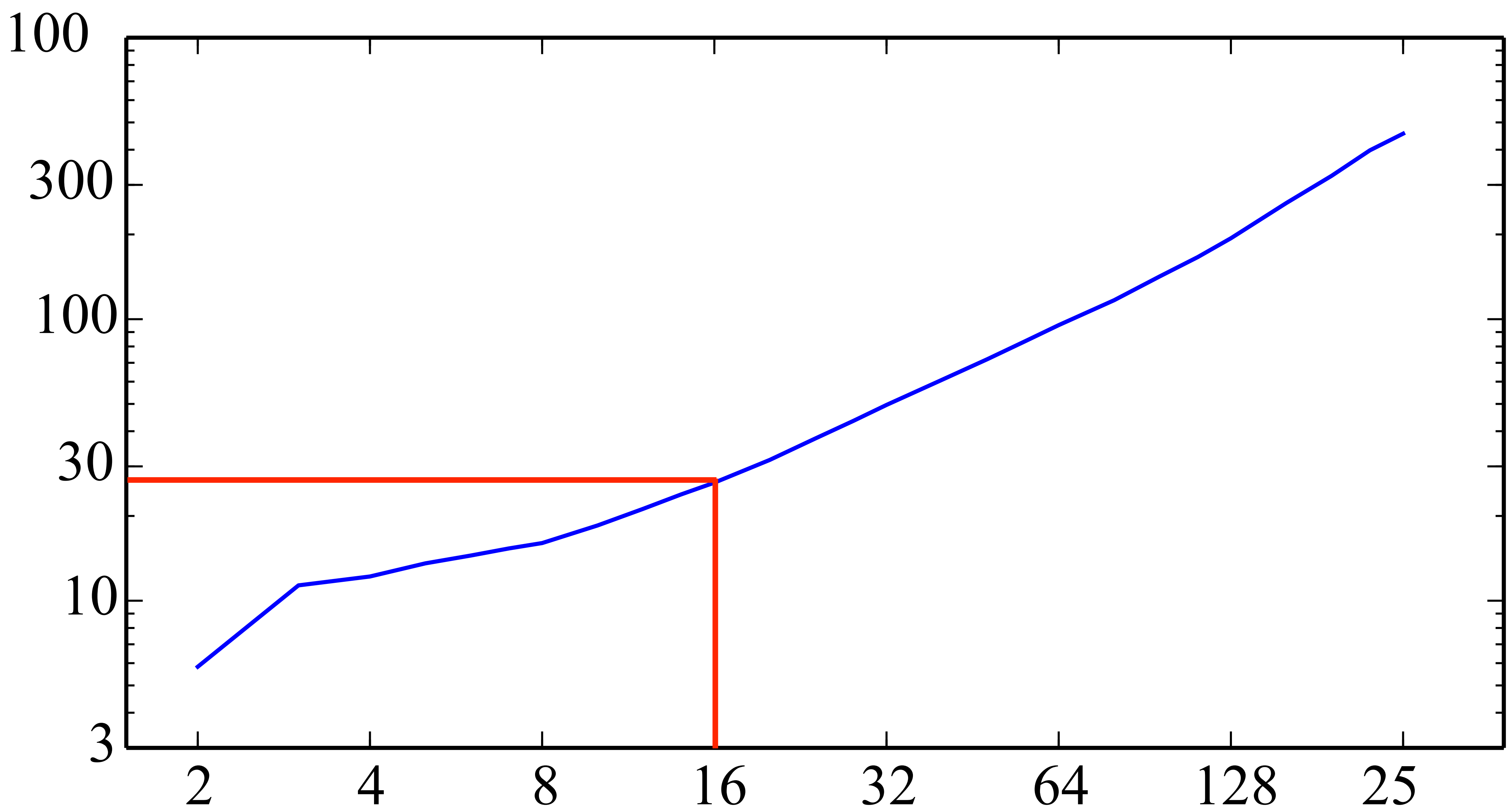


Broadcast PoK (DLog), **Pairwise**: 128 OTs

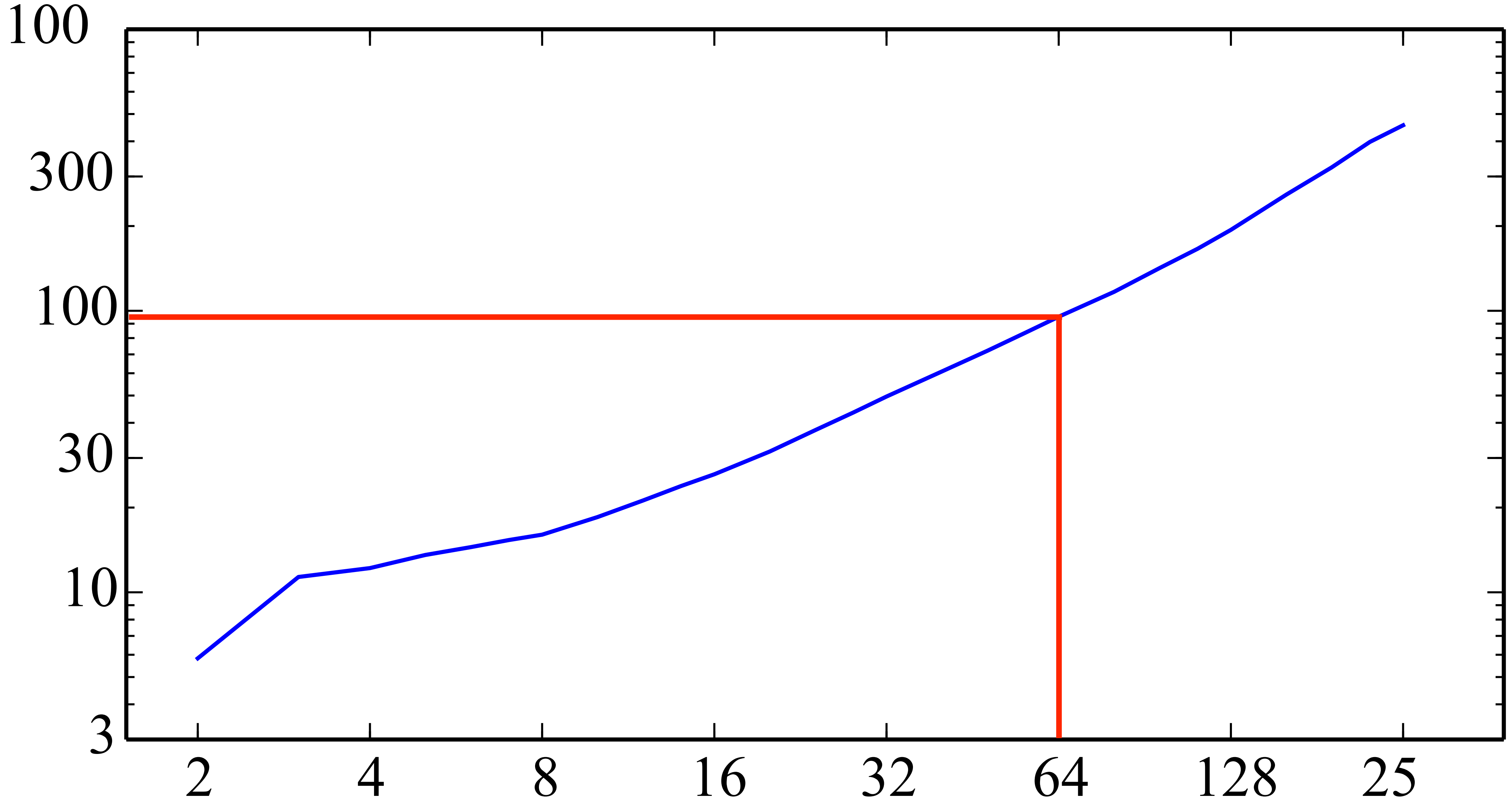
LAN Signing



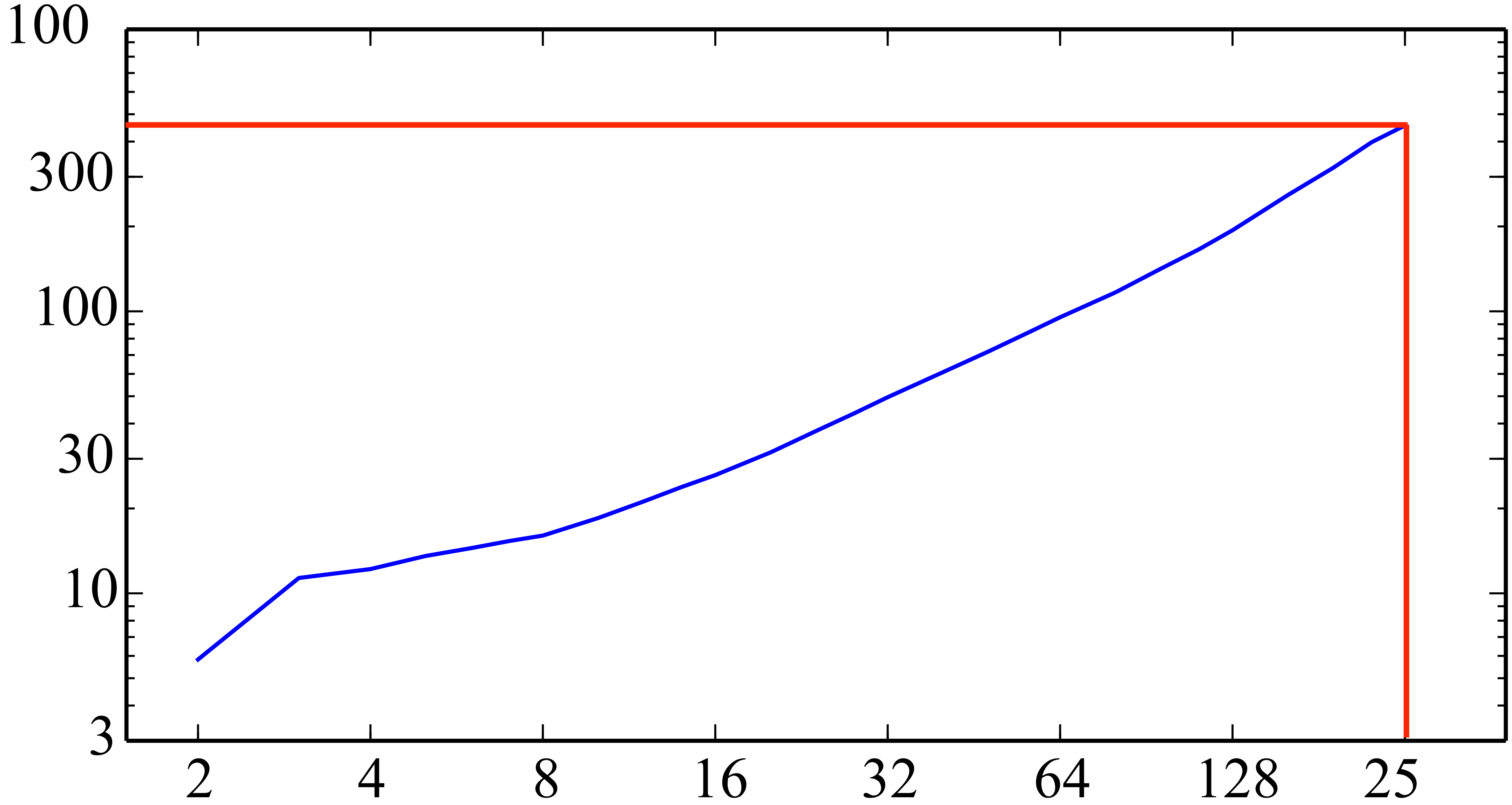
LAN Signing



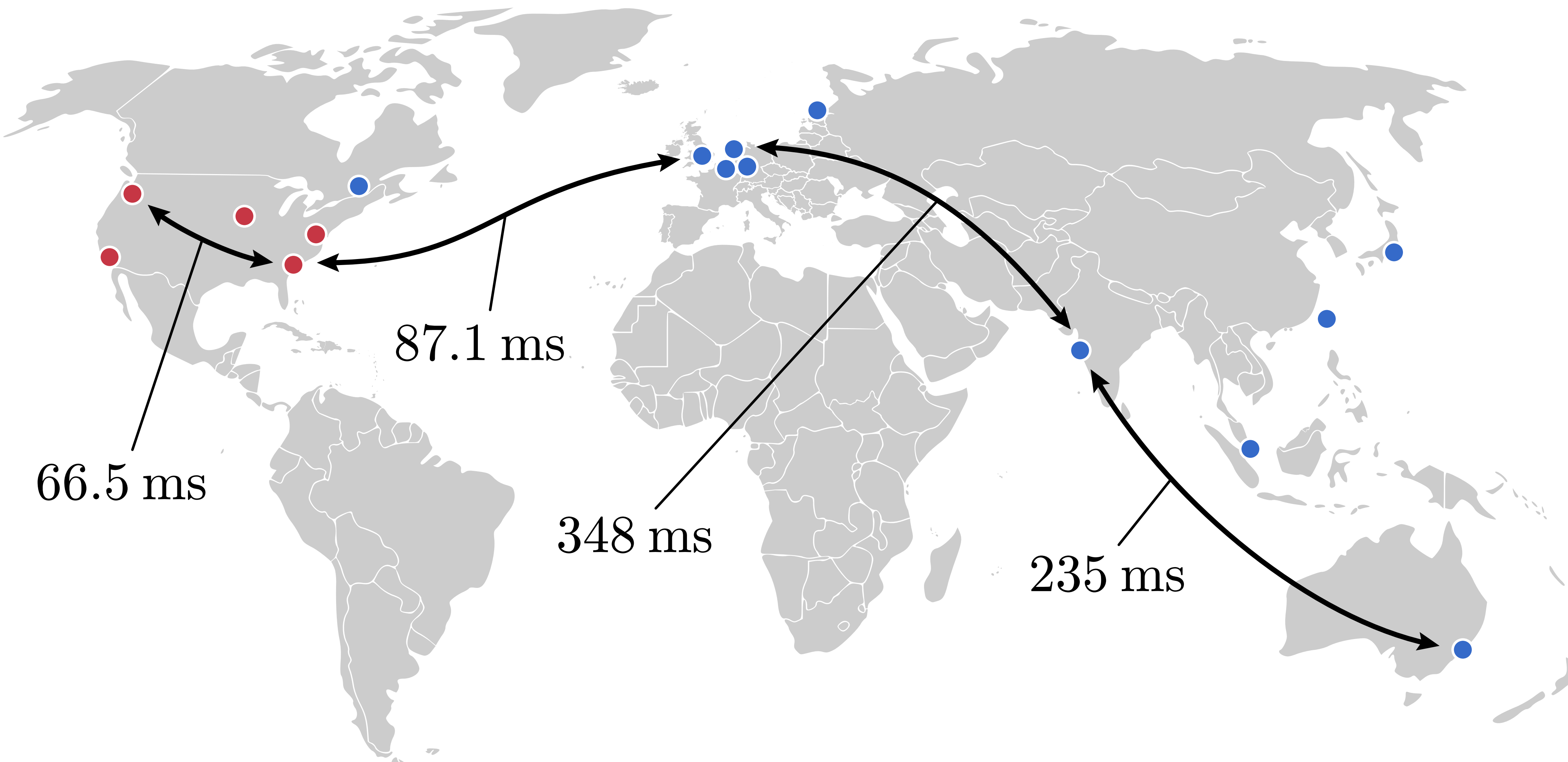
LAN Signing



LAN Signing



WAN Node Locations



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

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	~11000	~28000
BGG17	~650	~1500	–	–
GGN16	205	1136	–	–
Lindell17	36.8	–	2435	–
DKLs18	**3.8**	–	**43.4**	**177**

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

How much of a bottleneck is communication?

How much of a bottleneck is communication?

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

How much of a bottleneck is communication?

- **Mobile applications (human-initiated)**:
  - eg. t=4, ~260KB (2.08Mb) transmitted per party

How much of a bottleneck is communication?

- **Mobile applications (human-initiated)**:
  - eg. t=4, ~260KB (2.08Mb) transmitted per party
  - Well within LTE envelope (10Mbps) for responsivity

How much of a bottleneck is communication?

How much of a bottleneck is communication?

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

How much of a bottleneck is communication?

- **Large-scale automated distributed signing:**
  - Threshold 2:

How much of a bottleneck is communication?

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

How much of a bottleneck is communication?

- **Large-scale automated distributed signing:**
  - Threshold 2:
    - 3.8ms/sig => ~263 sig/second
    - Bandwidth required: ~490Mb

How much of a bottleneck is communication?

- **Large-scale automated distributed signing:**
  - Threshold 2:
    - 3.8ms/sig => ~263 sig/second
    - Bandwidth required: ~490Mb
  - Threshold 20:

How much of a bottleneck is communication?

- **Large-scale automated distributed signing:**
  - Threshold 2:
    - 3.8ms/sig => ~263 sig/second
    - Bandwidth required: ~490Mb
  - Threshold 20:
    - 31.6ms/sig => ~31 sig/second

How much of a bottleneck is communication?

- **Large-scale automated distributed signing:**
  - Threshold 2:
    - 3.8ms/sig => ~263 sig/second
    - Bandwidth required: ~490Mb
  - Threshold 20:
    - 31.6ms/sig => ~31 sig/second
    - Bandwidth required: ~200Mb

Future Directions

Future Directions

- **Constant Round Version** based on [BB89] (in progress)

Future Directions

- **Constant Round Version** based on [BB89] (in progress)
- MUL based on Additively Homomorphic Encryption for restricted bandwidth setting (when lax on assumptions)

Future Directions

- **Constant Round Version** based on [BB89] (in progress)
- MUL based on Additively Homomorphic Encryption for restricted bandwidth setting (when lax on assumptions)
- Efficient addition/removal parties

Future Directions

- **Constant Round Version** based on [BB89] (in progress)
- MUL based on Additively Homomorphic Encryption for restricted bandwidth setting (when lax on assumptions)
- Efficient addition/removal parties
- Identifying cheaters

Future Directions

- **Constant Round Version** based on [BB89] (in progress)
- MUL based on Additively Homomorphic Encryption for restricted bandwidth setting (when lax on assumptions)
- Efficient addition/removal parties
- Identifying cheaters
- Extensive benchmarking in more representative scenarios

Future Directions

- **Constant Round Version** based on [BB89] (in progress)
- MUL based on Additively Homomorphic Encryption for restricted bandwidth setting (when lax on assumptions)
- Efficient addition/removal parties
- Identifying cheaters
- Extensive benchmarking in more representative scenarios
  - eg. smartphones over LTE

Conclusion

Conclusion

- **Efficient full-threshold ECDSA with fully dist. keygen**

Conclusion

- **Efficient full-threshold ECDSA with fully dist. keygen**
- **Paradigm**: ‘produce candidate shares, verify by exponent check’ to minimize exponentiations

Conclusion

- **Efficient full-threshold ECDSA with fully dist. keygen**
- **Paradigm**: ‘produce candidate shares, verify by exponent check’ to minimize exponentiations
- **Instantiation**: Cryptographic assumptions native to ECDSA itself (**CDH** in the same curve)

Conclusion

- **Efficient full-threshold ECDSA with fully dist. keygen**
- **Paradigm**: ‘produce candidate shares, verify by exponent check’ to minimize exponentiations
- **Instantiation**: Cryptographic assumptions native to ECDSA itself (**CDH** in the same curve)
- Optimized computation **but communication well within practical range (65*t* KB/party)**

Conclusion

- **Efficient full-threshold ECDSA with fully dist. keygen**
- **Paradigm**: 'produce candidate shares, verify by exponent check' to minimize exponentiations
- **Instantiation**: Cryptographic assumptions native to ECDSA itself (**CDH** in the same curve)
- Optimized computation **but communication well within practical range (65*t* KB/party)**
- **Wall-clock times**: Practical in realistic scenarios

Thank you!

https://gitlab.com/neucrypt/mpecdsa

Component-Wise Comparison

- LNR18 can be instantiated with “ECDSA assumptions”
- Heaviest components of signing (per party):
  - **LNR18**: $2t \times 2$P-MUL + $(83 + 78 \cdot t)$ Exponentiations
  - **This work**: $4t \times 2$P-MUL + $(5+t)$ Exponentiations

Check in Exponent

2. [sk/*k*] and [*1*/*k*] are consistent with *pk*

$$\sum_{i \in [n]} \left[\frac{\phi}{k}\right]_i \cdot \mathsf{pk} - \left[\frac{\phi}{k} \cdot \mathsf{sk}\right]_i \cdot G = 0$$

Check in Exponent

3. [sk/*k*] and [*1*/*k*] are consistent with *R*

$$\sum_{i \in [n]} \left[\frac{\phi}{k} \cdot \mathsf{sk} \right]_i \cdot R = \phi \cdot \mathsf{pk}$$