Threshold ECDSA from ECDSA assumptions: *the multiparty case*

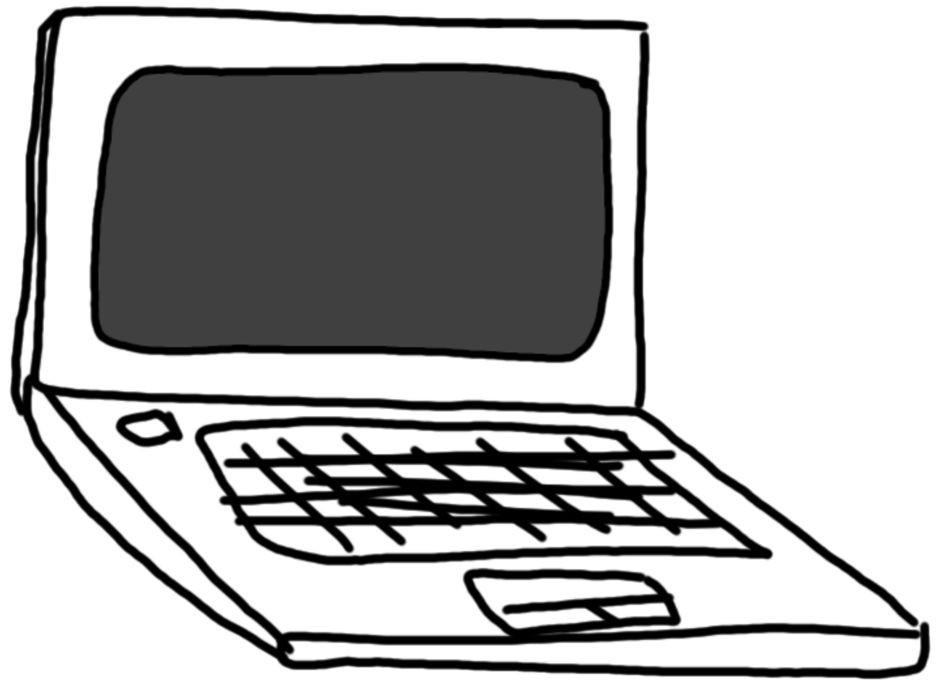
Jack **Doerner**, Yashvanth **Kondi**, Eysa **Lee**, and abhi **shelat**

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

Northeastern University

Traditional Signature

pk



sk

Traditional Signature

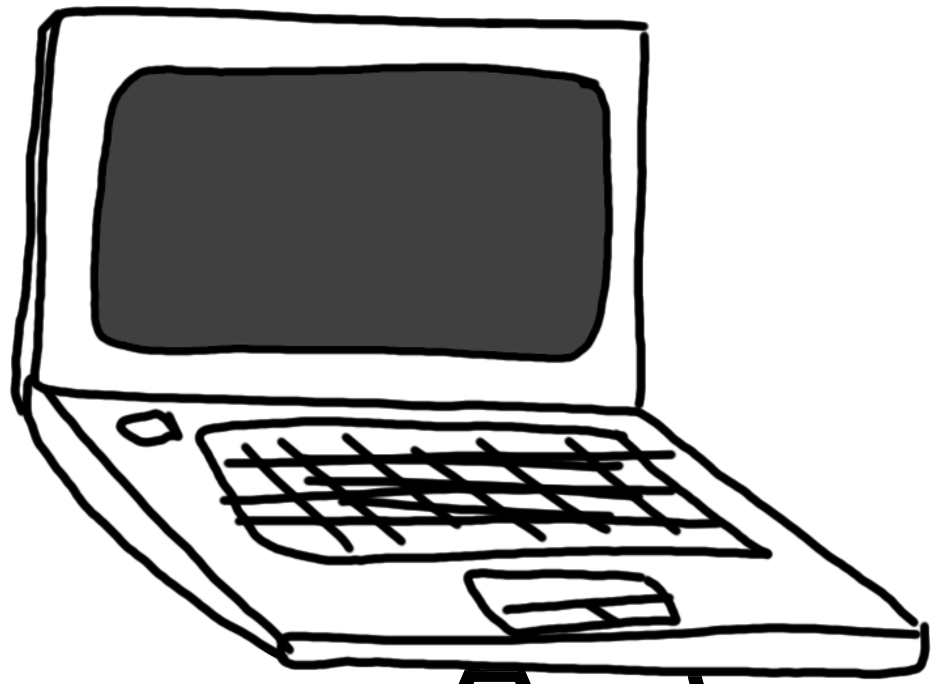
pk

sk

Threshold Signature

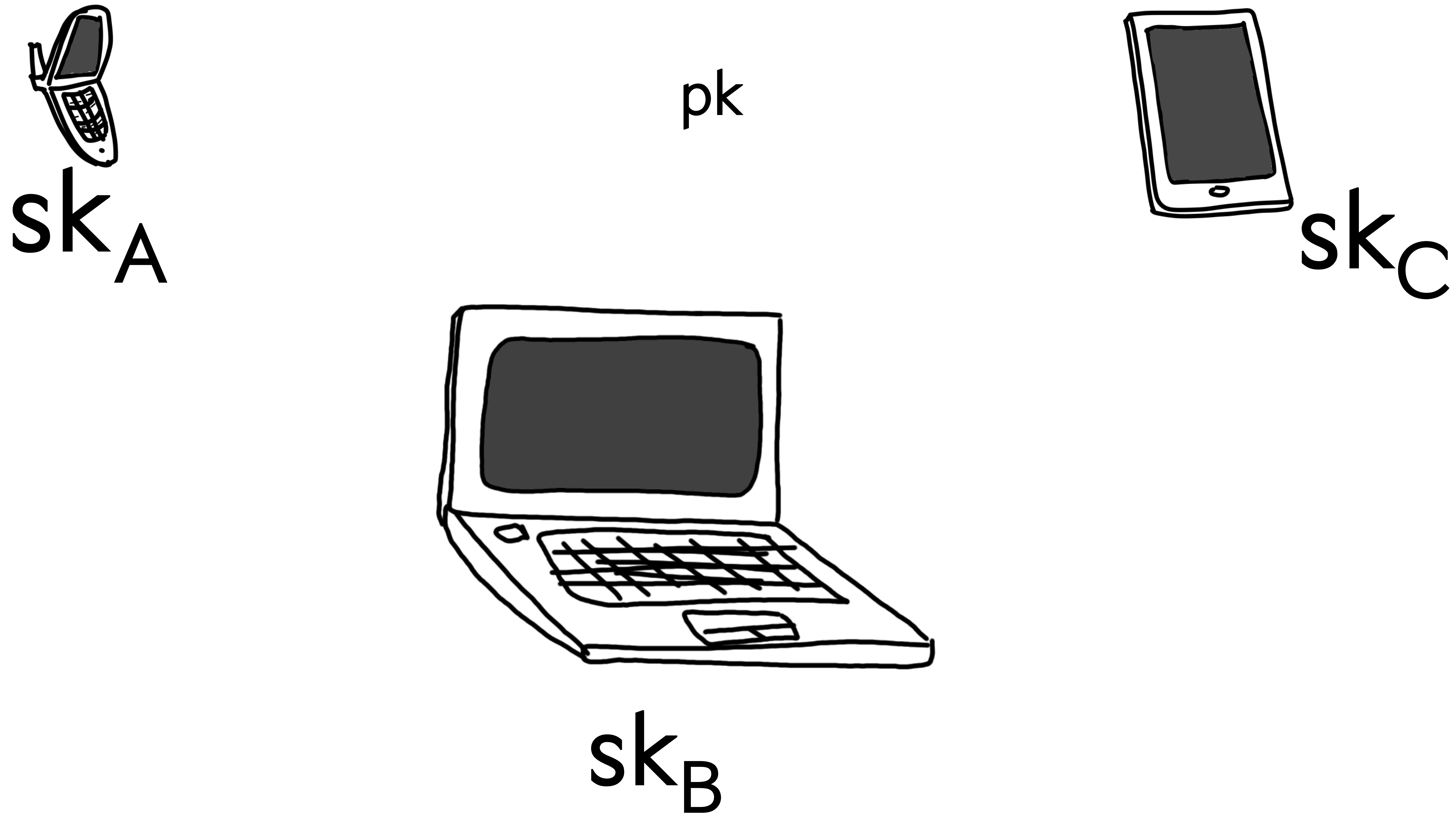
$$\{\mathsf{sk}_\mathsf{A}, \mathsf{sk}_\mathsf{B}, \mathsf{sk}_C\} \leftarrow \mathrm{Share}(\mathsf{sk})$$

pk



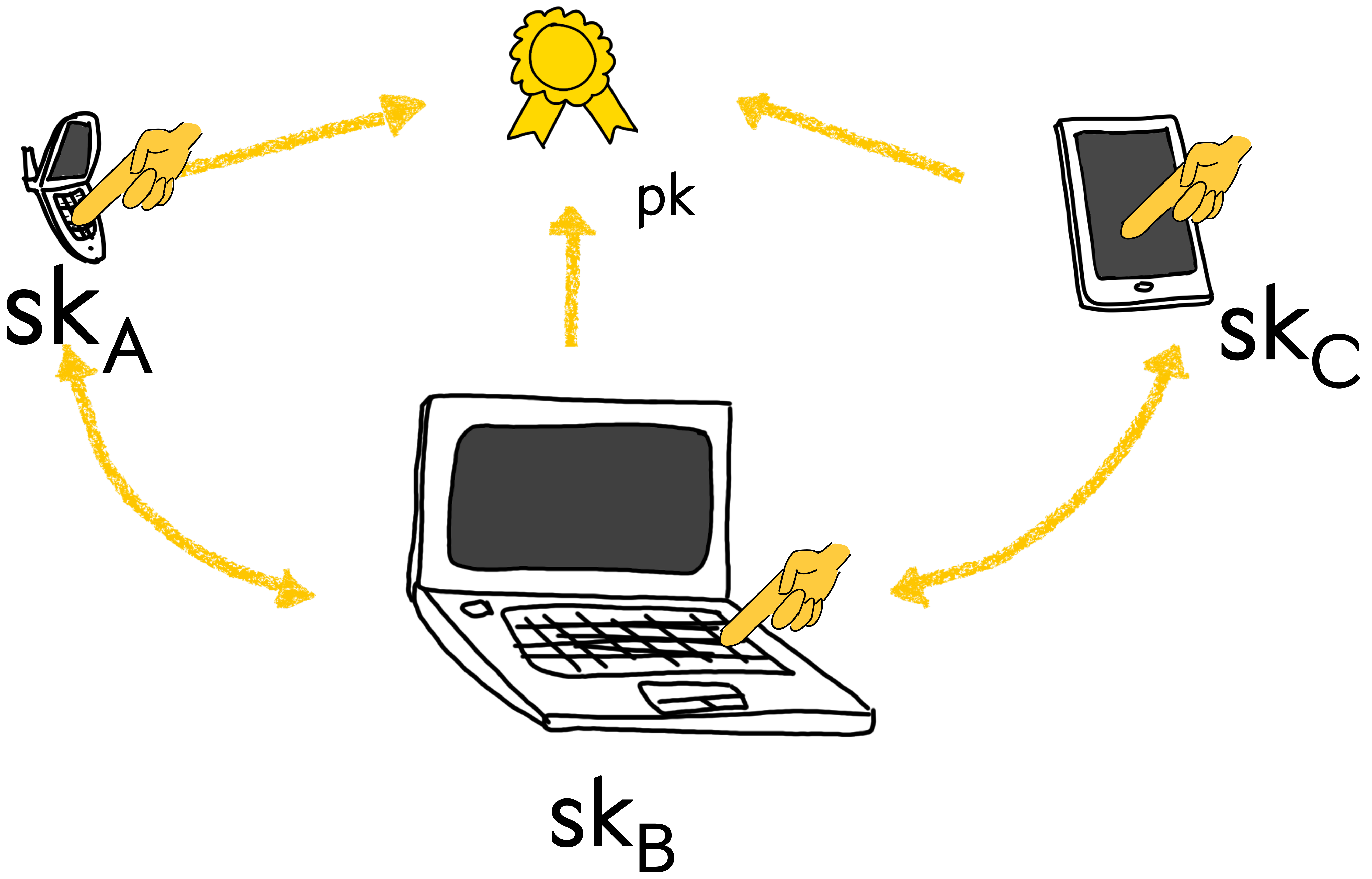
Threshold Signature

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



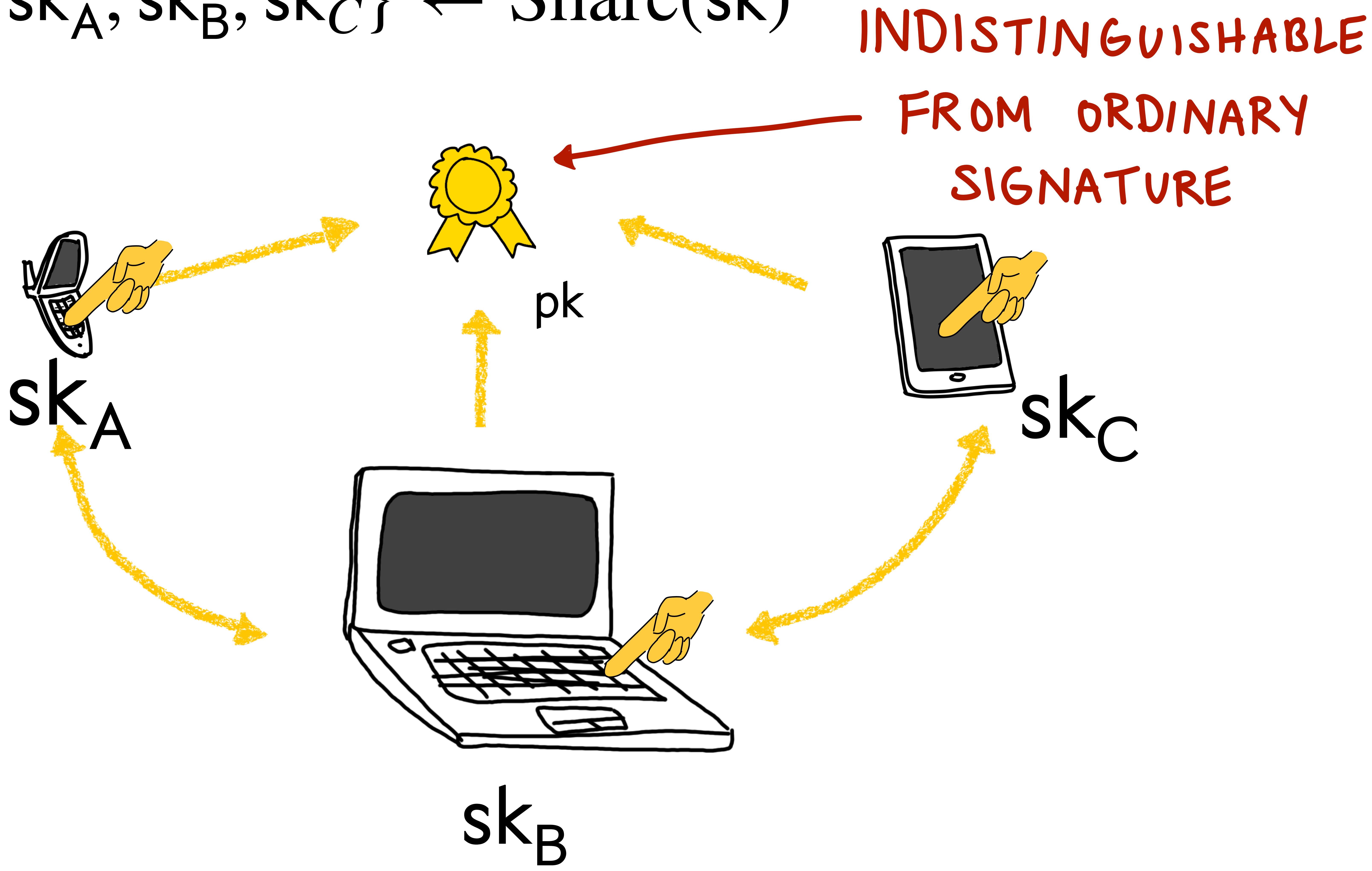
Threshold Signature

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

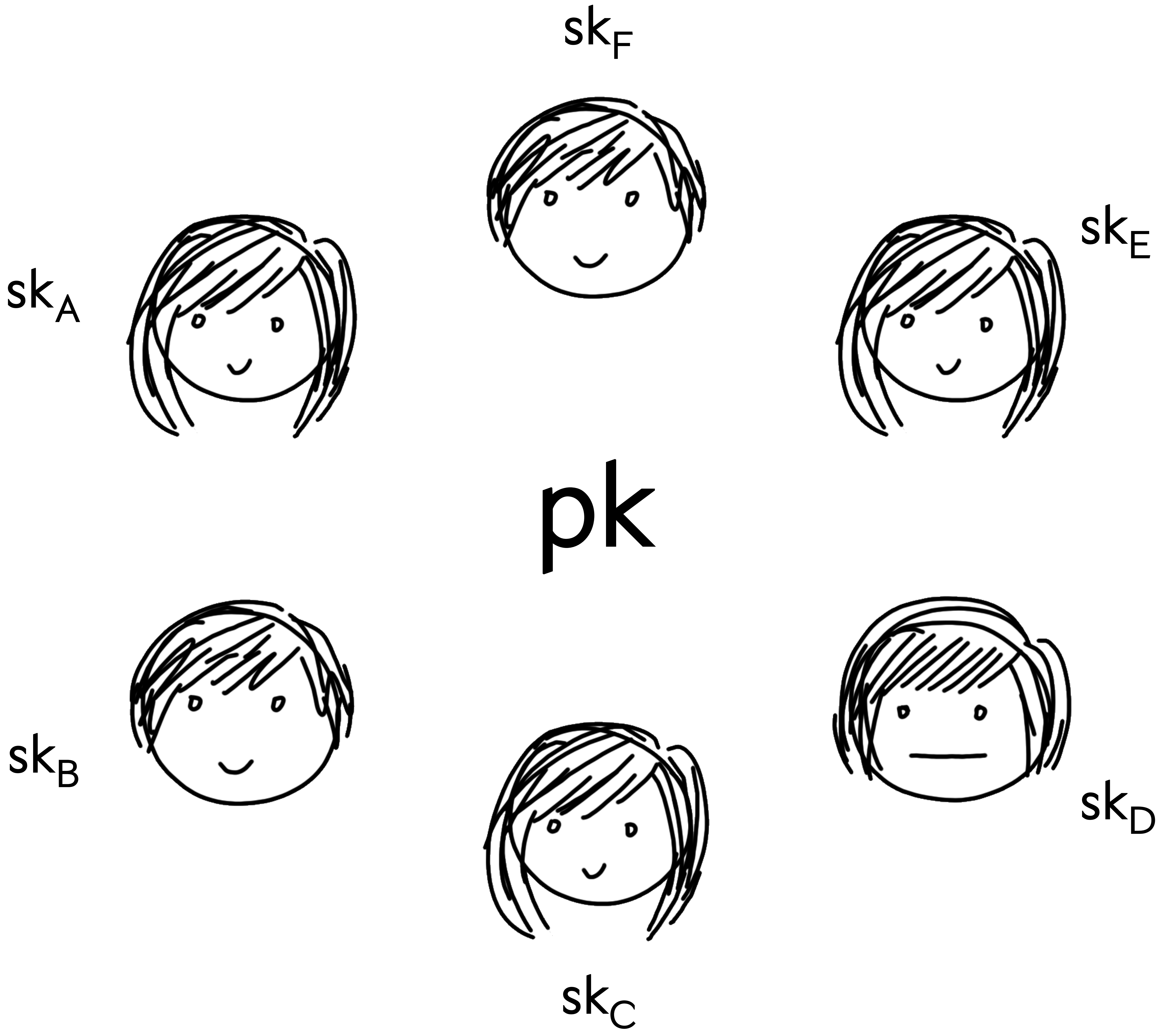


Threshold Signature

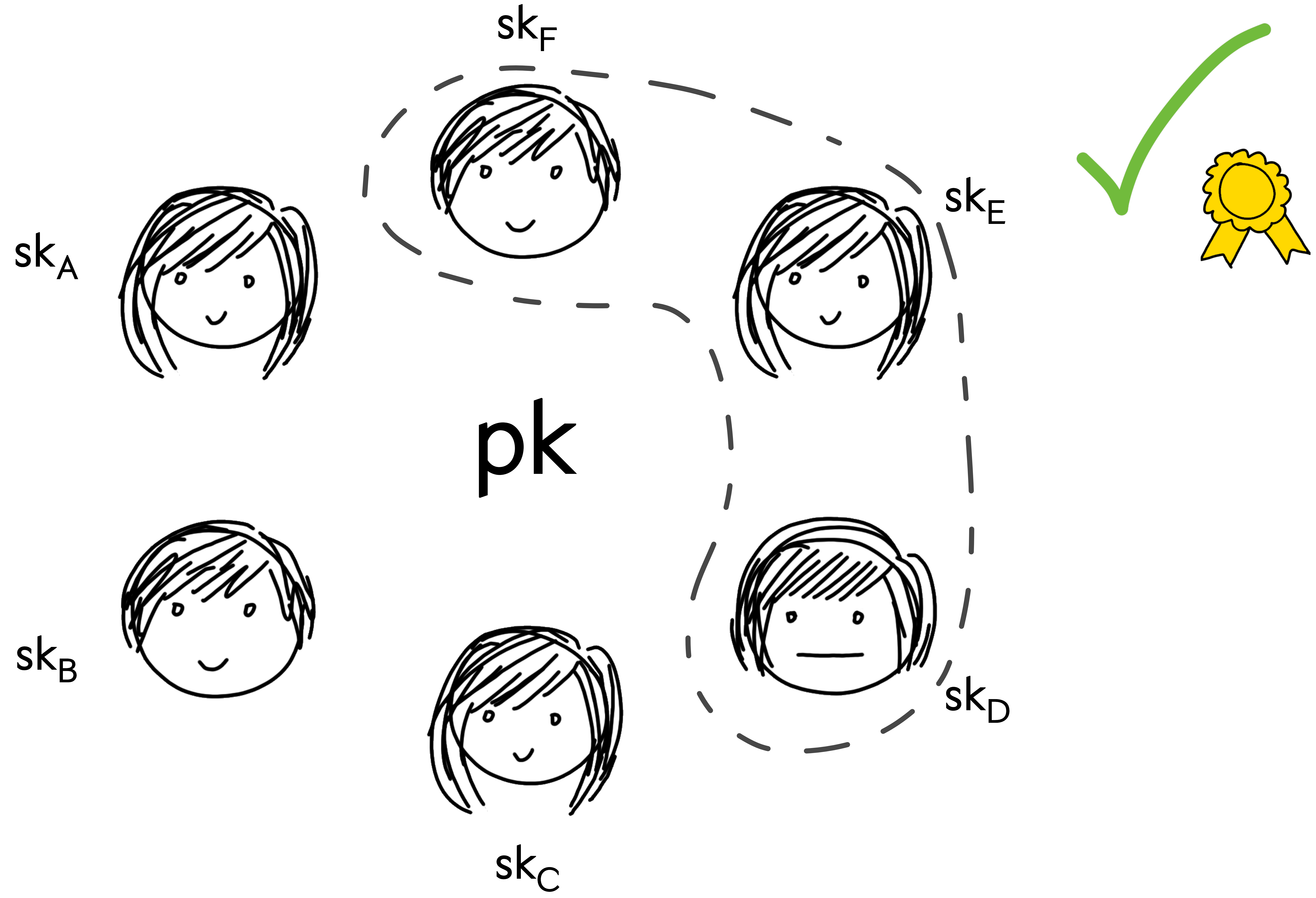
$$\{\mathrm{sk}_A, \mathrm{sk}_B, \mathrm{sk}_C\} \leftarrow \mathrm{Share}(\mathrm{sk})$$



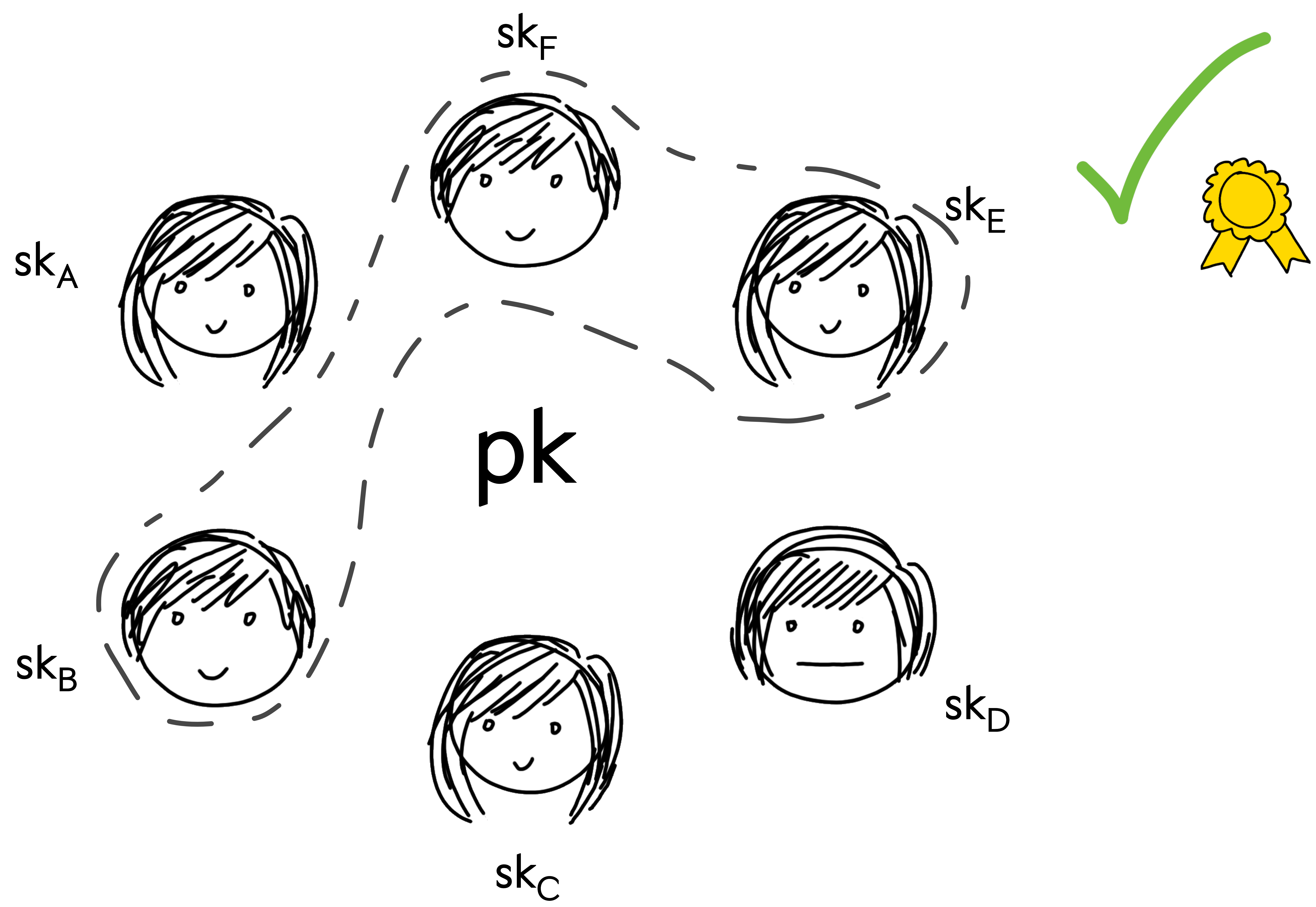
3-of-n Signature Scheme



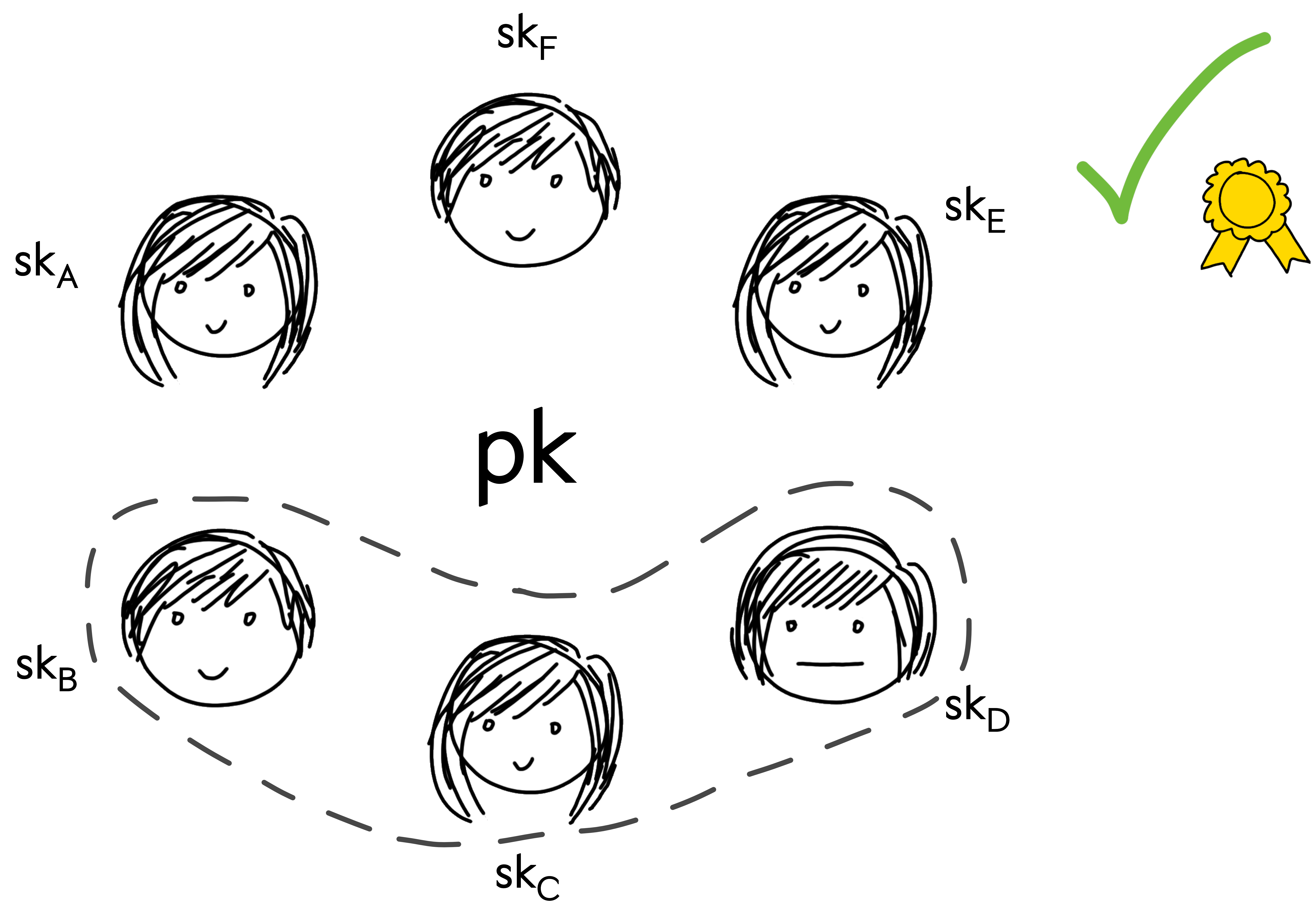
3-of-n Signature Scheme



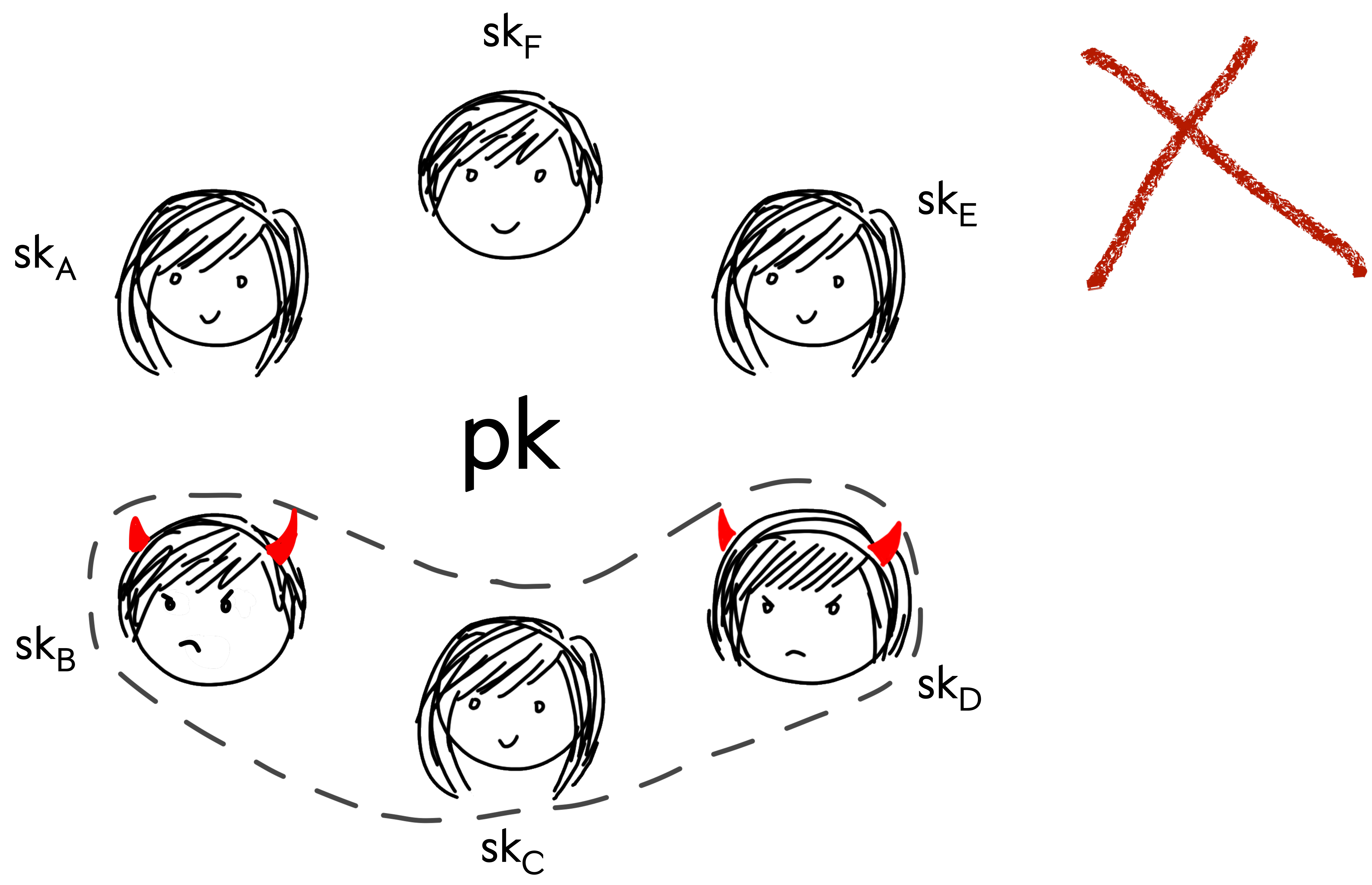
3-of-n Signature Scheme



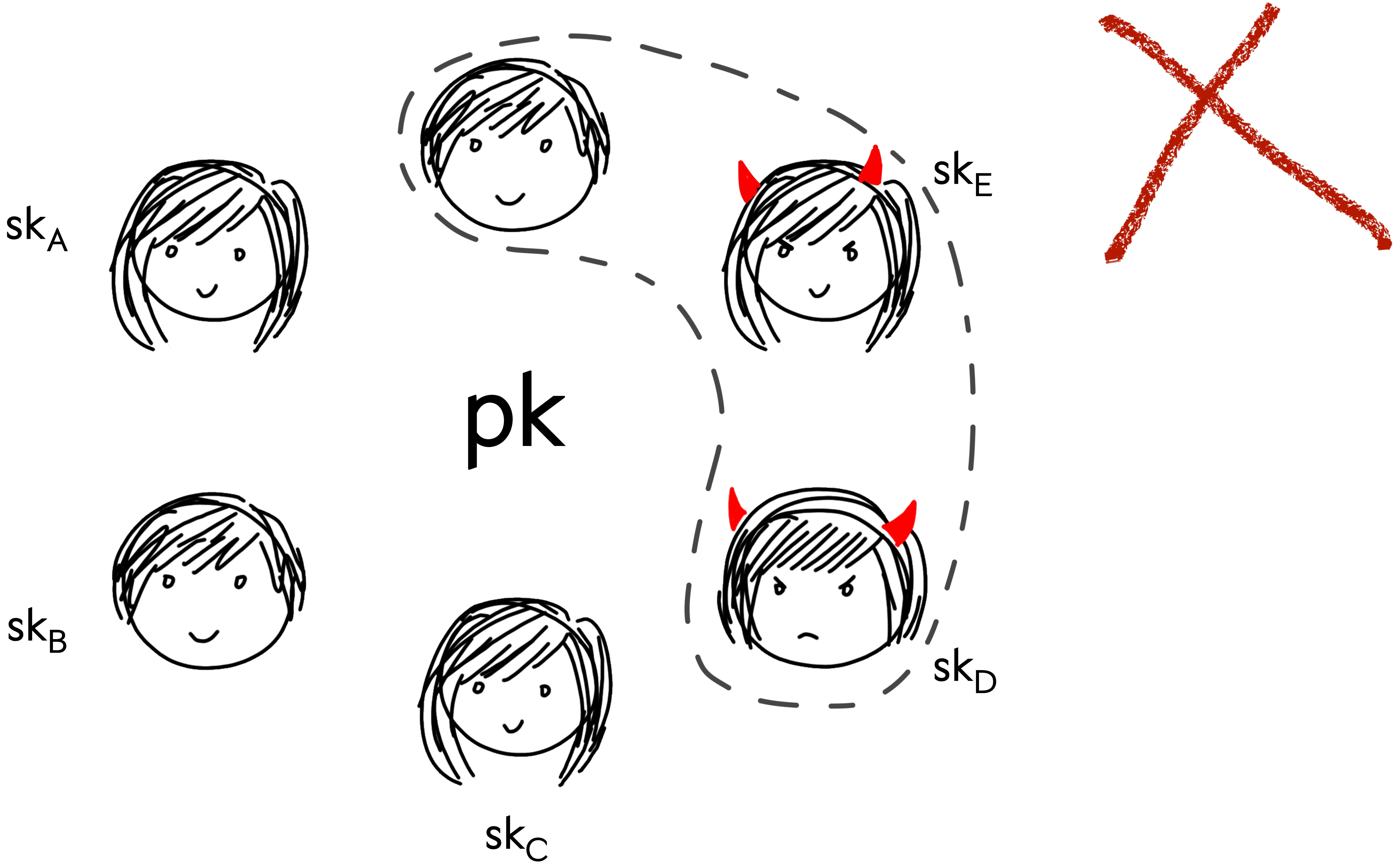
3-of-n Signature Scheme



3-of-n Signature Scheme



3-of-n Signature Scheme



Full Threshold

- Scheme can be instantiated with any $t <= n$
- Adversary corrupts up to t-1 parties

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

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

ECDSA Recap

$$R = k \cdot G$$

ECDSA Recap

$$R = k \cdot G$$

$$sign(m, \mathsf{sk}, k) =$$

ECDSA Recap

$$R = k \cdot G$$

$$sign(m, \mathsf{sk}, k) = H(m)$$

ECDSA Recap

$$R = k \cdot G$$

$$sign(m, \mathsf{sk}, k) = H(m) + \mathsf{sk} \cdot r_x$$

ECDSA Recap

x-coordinate of ***R***

$$R = k \cdot G$$

$$sign(m, \mathsf{sk}, k) = H(m) + \mathsf{sk} \cdot r_x$$

ECDSA Recap

x-coordinate of *R*

$$R = k \cdot G$$

$$sign(m, \mathsf{sk}, k) = \frac{H(m) + \mathsf{sk} \cdot r_x}{k}$$

ECDSA Recap

x-coordinate of ***R***

$$R = k \cdot G$$

$$sign(m, \mathsf{sk}, k) = \boxed{\frac{H(m) + \mathsf{sk} \cdot r_x}{k}}$$

Non-linearity makes ‘thresholdization’ difficult

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 last year** [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 last year** [DKLs18]: 2-of-n ECDSA under native assumptions
- **This work**: 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)
  - **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 the same curve

Tradeoffs

Tradeoffs

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

Tradeoffs

- Our work 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 most applications

Tradeoffs

- Our work 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 most 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., 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**: CDH is hard in the ECDSA curve

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**: CDH is hard in the ECDSA curve
- **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**: 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 (128 OTs)

Setup

- **Fully distributed**
- **MUL setup**: Pairwise among parties (128 OTs)
- **Key generation**: (Pedersen-style)

Setup

- **Fully distributed**
- **MUL setup**: Pairwise among parties (128 OTs)
- **Key generation**: (Pedersen-style)
  - Every party Shamir-shares a random secret

Setup

- **Fully distributed**
- **MUL setup**: Pairwise among parties (128 OTs)
- **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 (128 OTs)
- **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

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):

Obtaining Candidate Shares

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

  - 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):
  - 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]

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]

Major challenges from 2 to Multi-party

Major challenges from 2 to Multi-party

2-party check does not obviously generalize [LNR18]

Major challenges from 2 to Multi-party

2-party check does not obviously generalize [LNR18]

Can’t use Diffie-Hellman Exchange for *R*

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

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

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 $$R = k_A k_h \cdot G$$

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

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

$$R = k_A k_h \cdot G$$

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

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:

Adversary's contribution

Honest Party's contribution

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

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

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

Check in Exponent

Adversary's contribution

Honest Party's contribution

Attempt at a solution:

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

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

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

Check in Exponent

Adversary's contribution

Honest Party's contribution

Attempt at a solution:

Public

$$R = k_A k_h \cdot G$$

Broadcast

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

Verify

Check in Exponent

Attempt at a solution:

Adversary's contribution

Honest Party's contribution

Public

$$R = k_A k_h \cdot G$$

Broadcast

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

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

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

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

Check in Exponent

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

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

Check in Exponent

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

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

Check in Exponent

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

Each costs, per party:

$$
\begin{tikzcd}
{[k]} \arrow[r, bend left, no head, "R"] \arrow[rr, bend right, no head, "{R, \mathsf{pk}}"'] & \left[\frac{1}{k}\right] \arrow[r, bend left, no head, "\mathsf{pk}"] & \left[\frac{\mathsf{sk}}{k}\right]
\end{tikzcd}
$$

Check in Exponent

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

Each costs, per party:

-2 exponentiations

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

Check in Exponent

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

Each costs, per party:

-2 exponentiations

-2 field elements

$$\begin{tikzcd}[column sep=huge]
{[k]} \arrow[r, bend left, dash, "R"] \arrow[rr, bend right=40, dash, "{R, \mathrm{pk}}"'] & \left[\dfrac{1}{k}\right] \arrow[r, bend left, dash, "\mathrm{pk}"] & \left[\dfrac{\mathrm{sk}}{k}\right]
\end{tikzcd}$$

Check in Exponent

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

Each costs, per party:

-2 exponentiations

-2 field elements

Two broadcast rounds

R pk

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

R, pk

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

Setup

Signing

Dominant Costs

	Rounds
Setup	
Signing	

Dominant Costs

	Rounds	Public Key
Setup		
Signing		

Dominant Costs

	Rounds	Public Key	Bandwidth
Setup			
Signing			

Dominant Costs

	Rounds	Public Key	Bandwidth
Setup			
Signing			

Dominant Costs

	Rounds	Public Key	Bandwidth
Setup	5		
Signing			

Dominant Costs

	Rounds	Public Key	Bandwidth
Setup	5	$520n$	
Signing			

Dominant Costs

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

Dominant Costs

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

Dominant Costs

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

Dominant Costs

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

Dominant Costs

	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

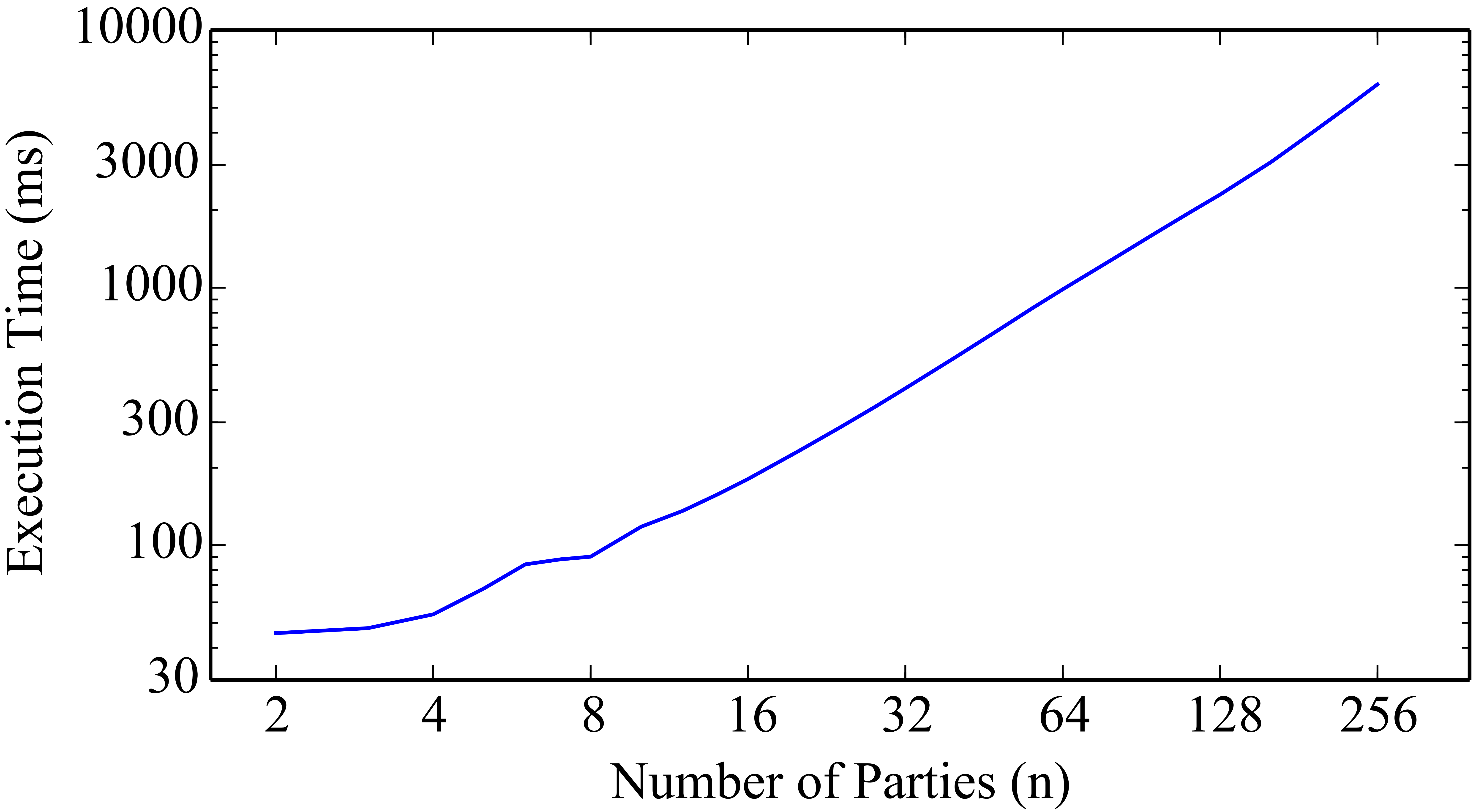
Benchmarks

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

Benchmarks

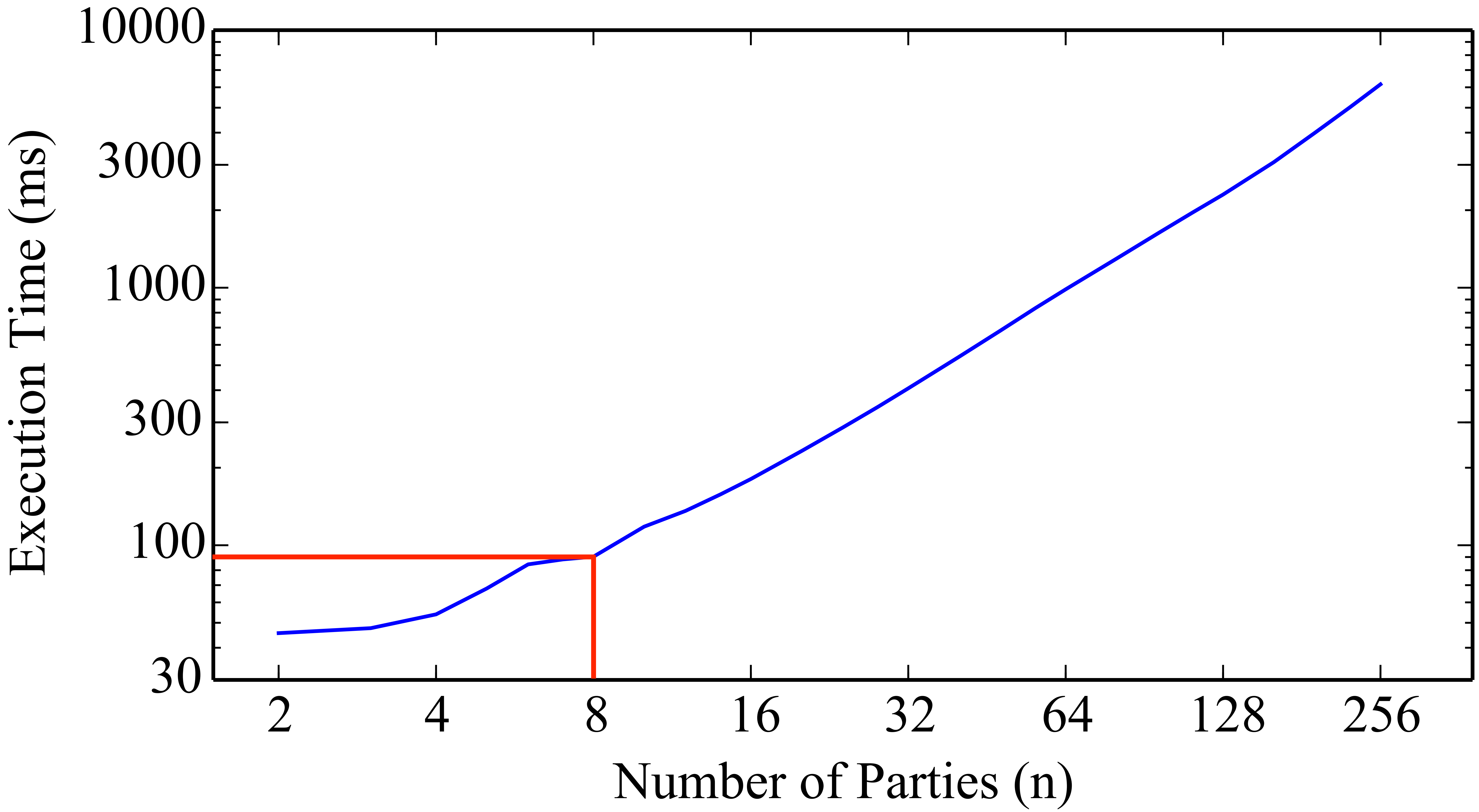
- Implementation in **Rust**
- Ran benchmarks on Google Cloud
- One node per party
- **LAN** and **WAN** tests (up to **16 zones**)
- **Low Power Friendliness**: Raspberry Pi (~93ms for 3-of-3)

LAN Setup



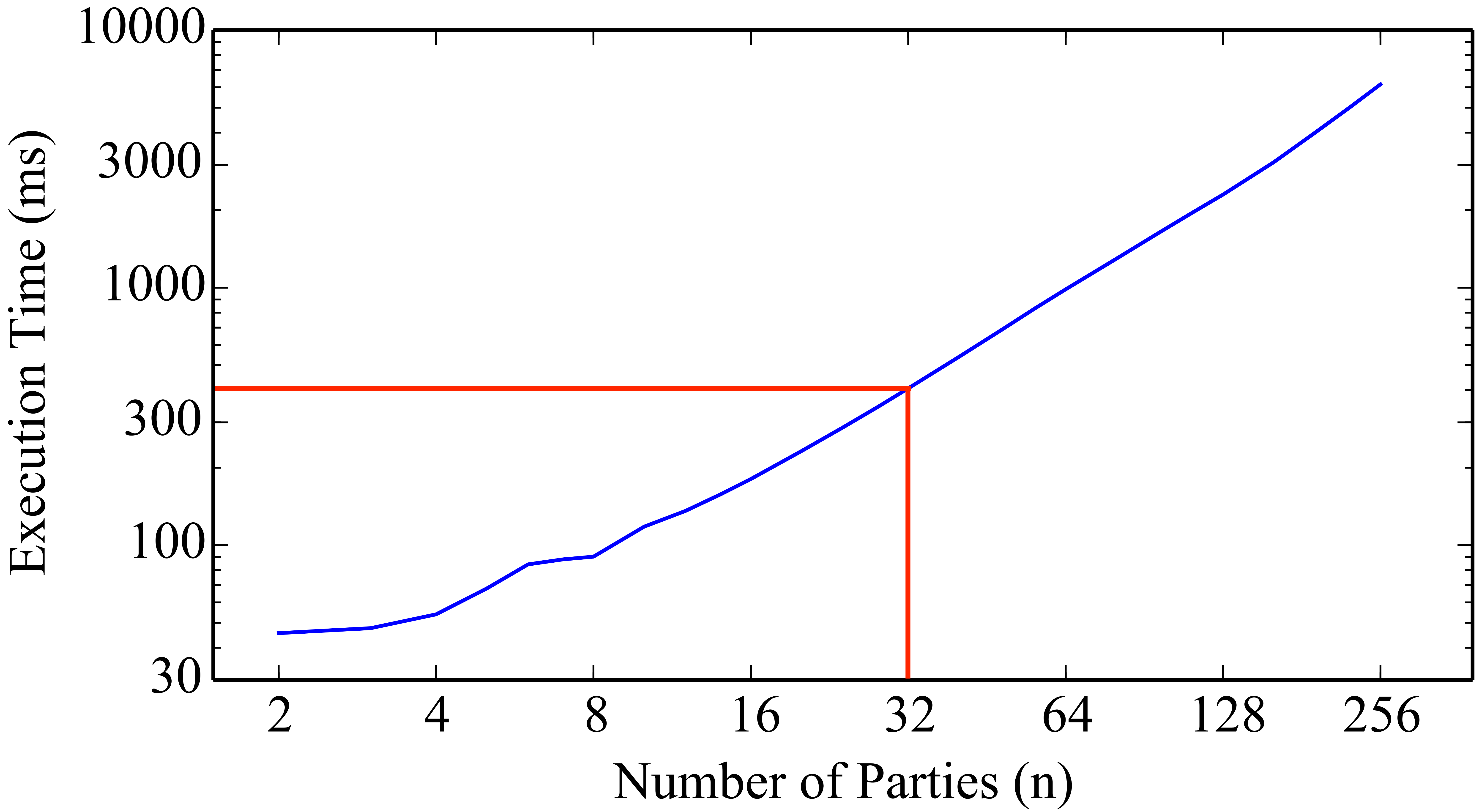
Broadcast PoK (DLog), **Pairwise**: 128 OTs

LAN Setup



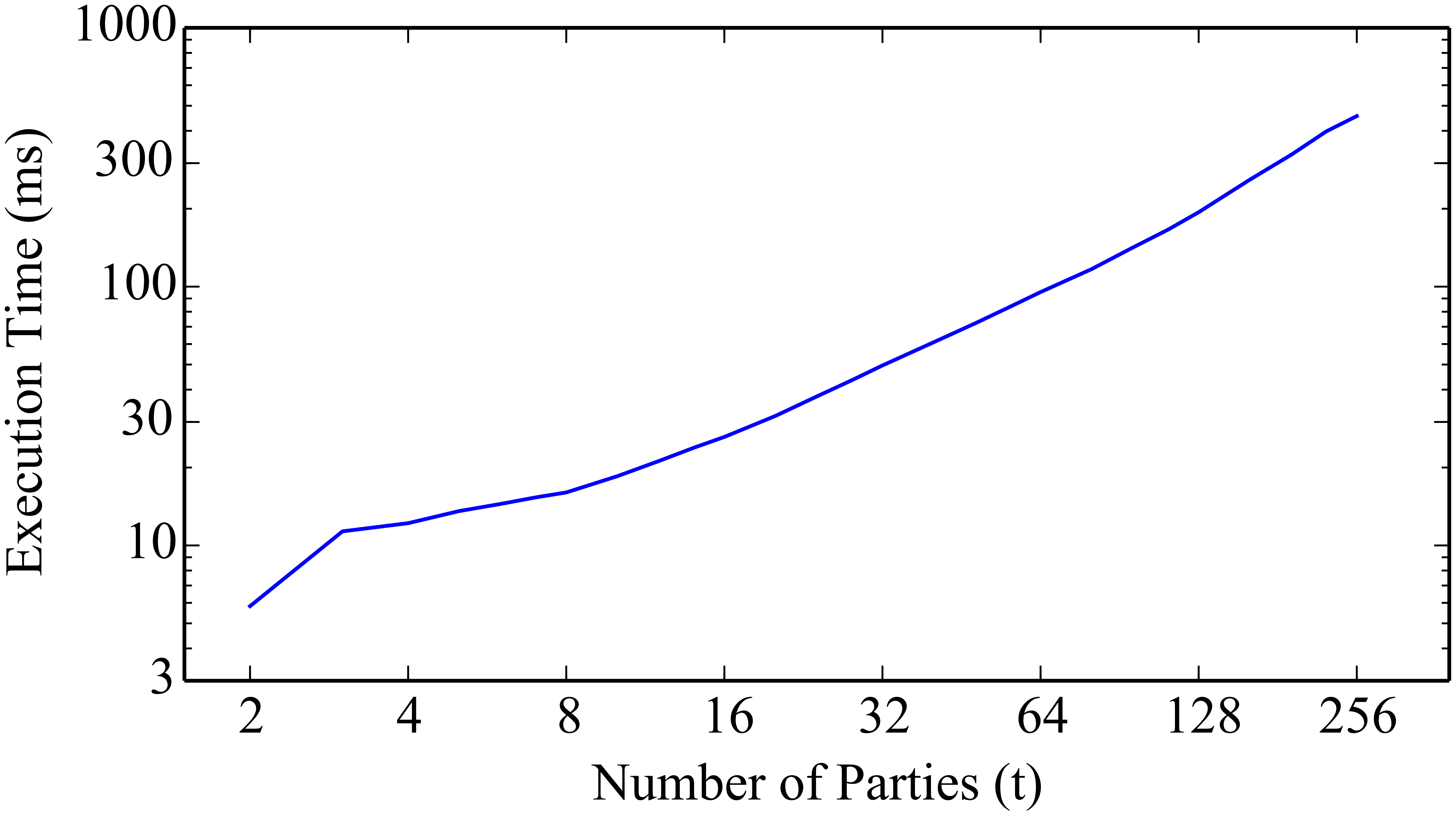
Broadcast PoK (DLog), **Pairwise**: 128 OTs

LAN Setup



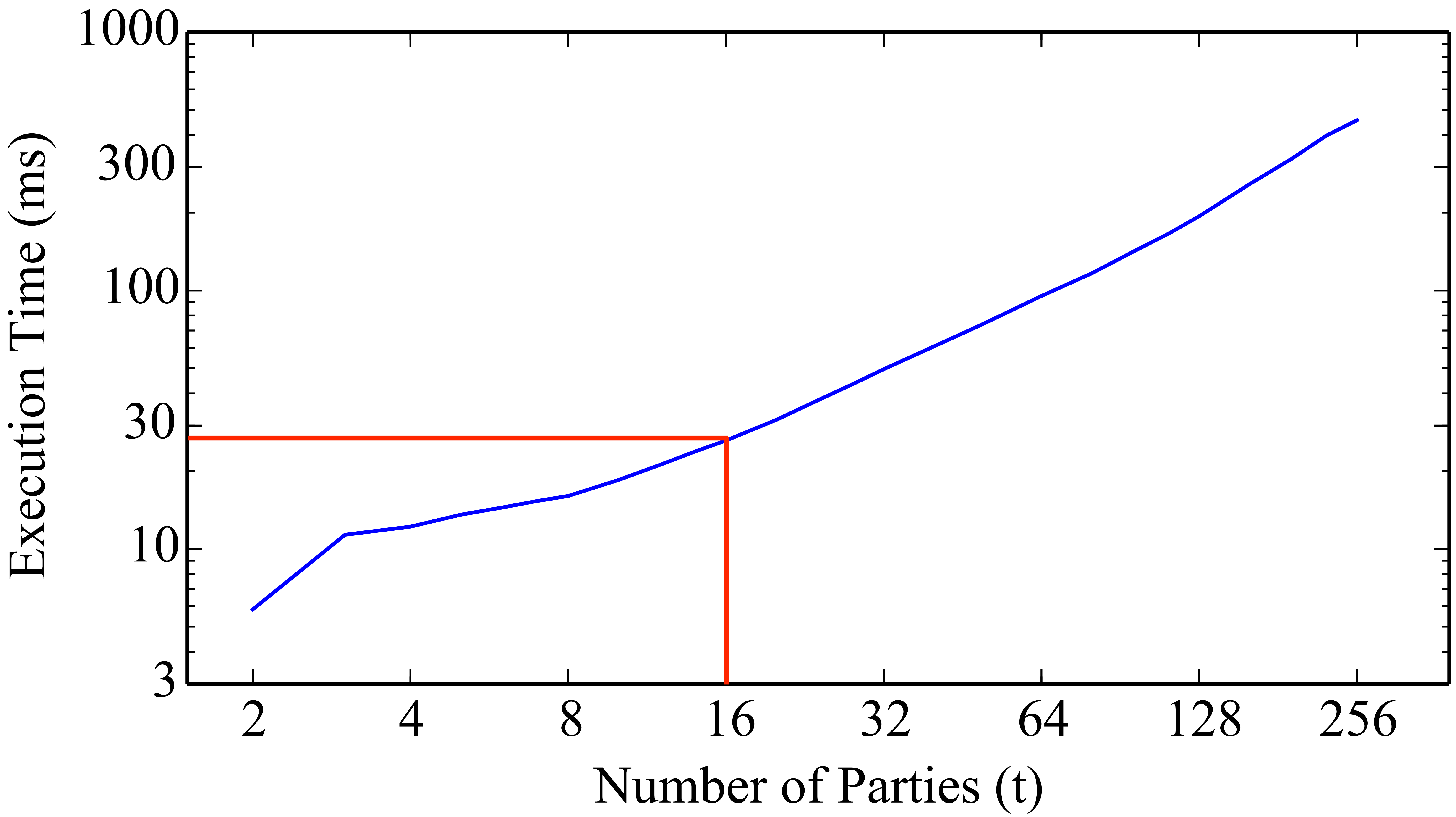
Broadcast PoK (DLog), **Pairwise**: 128 OTs

LAN Signing



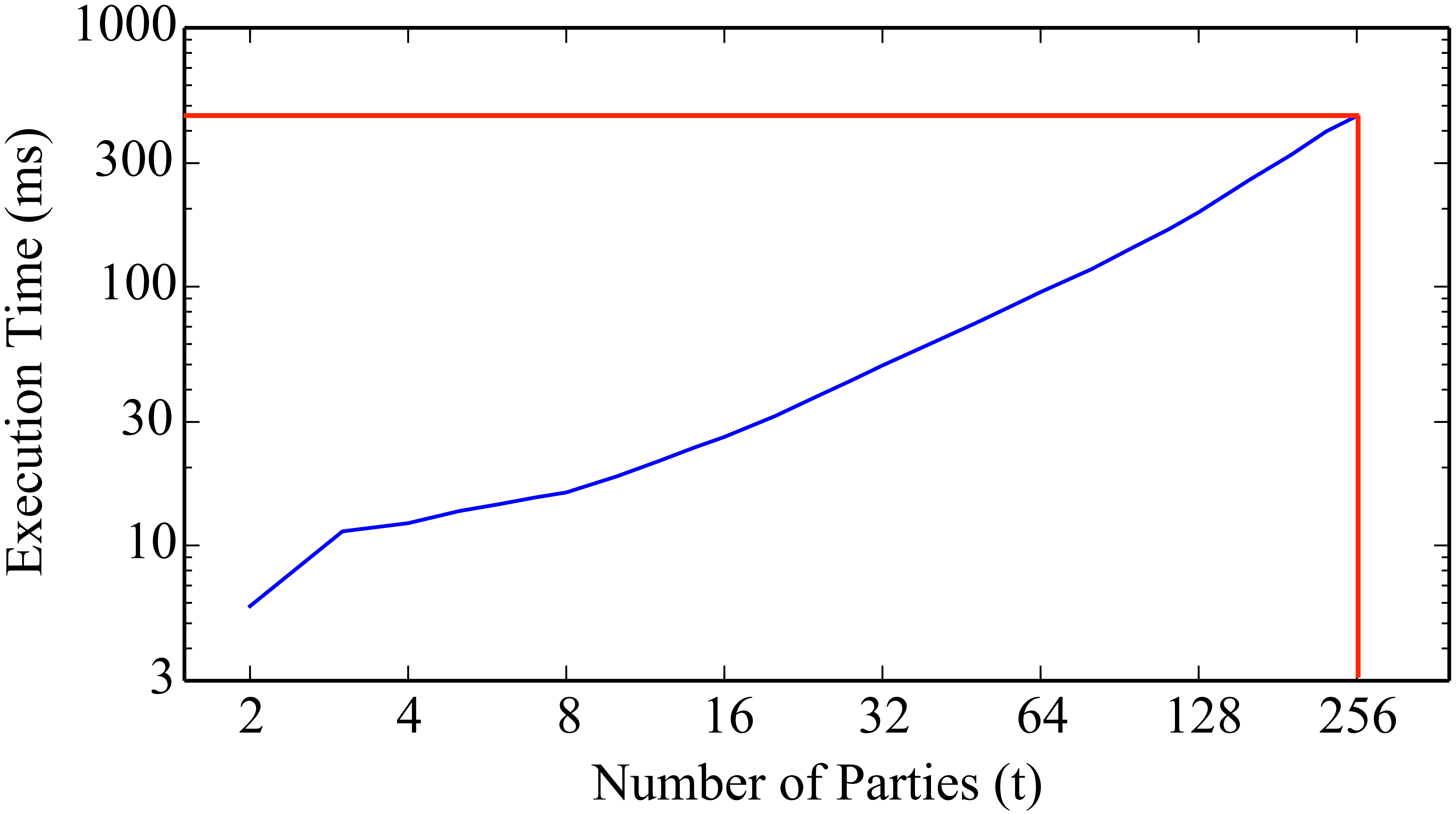
1000
300
100
30
10
3
Execution Time (ms)
2
4
8
16
32
64
128
256
Number of Parties (t)

LAN Signing



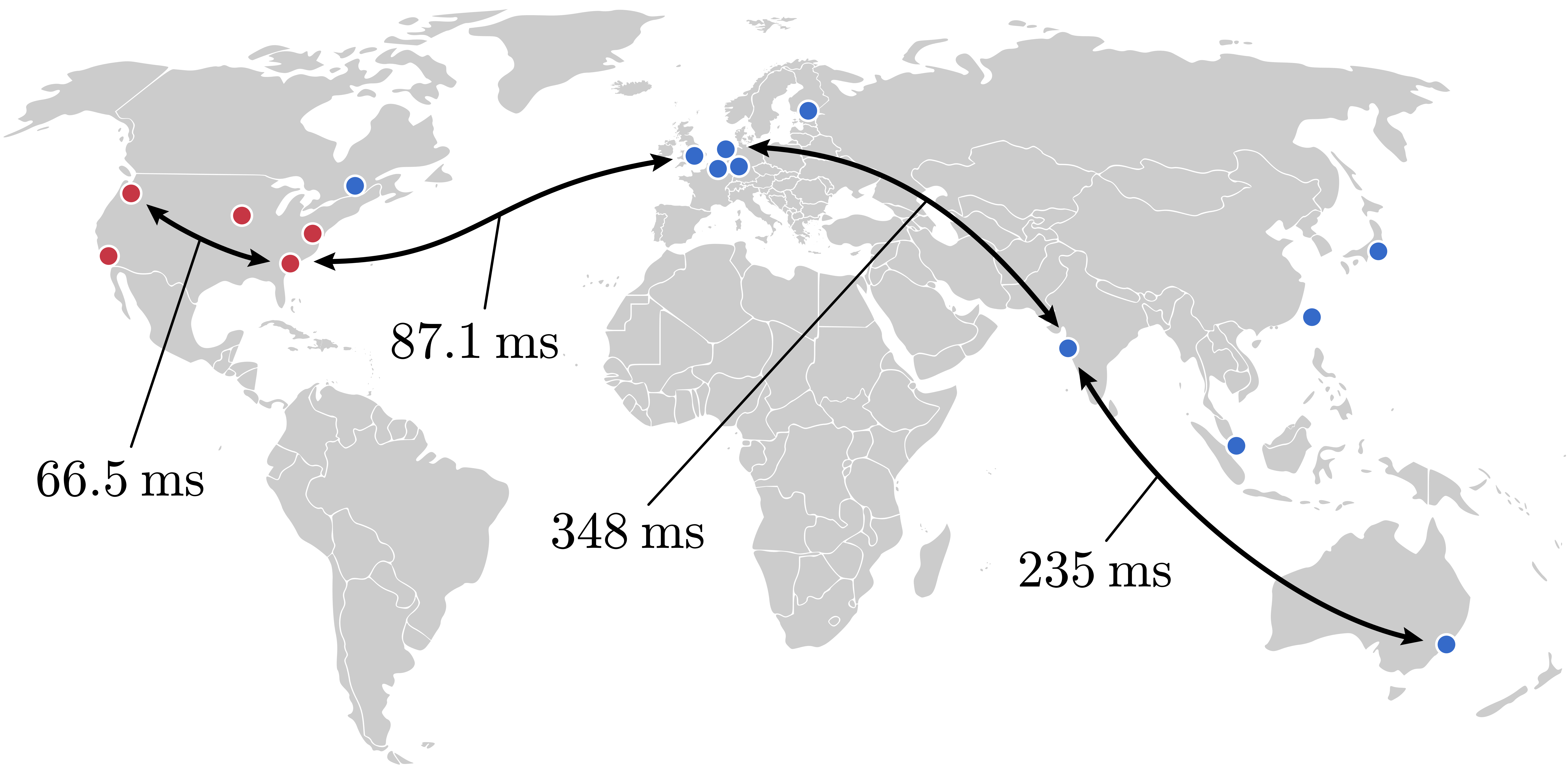
1000
300
100
30
10
3
Execution Time (ms)
2
4
8
16
32
64
128
256
Number of Parties (t)

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

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

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

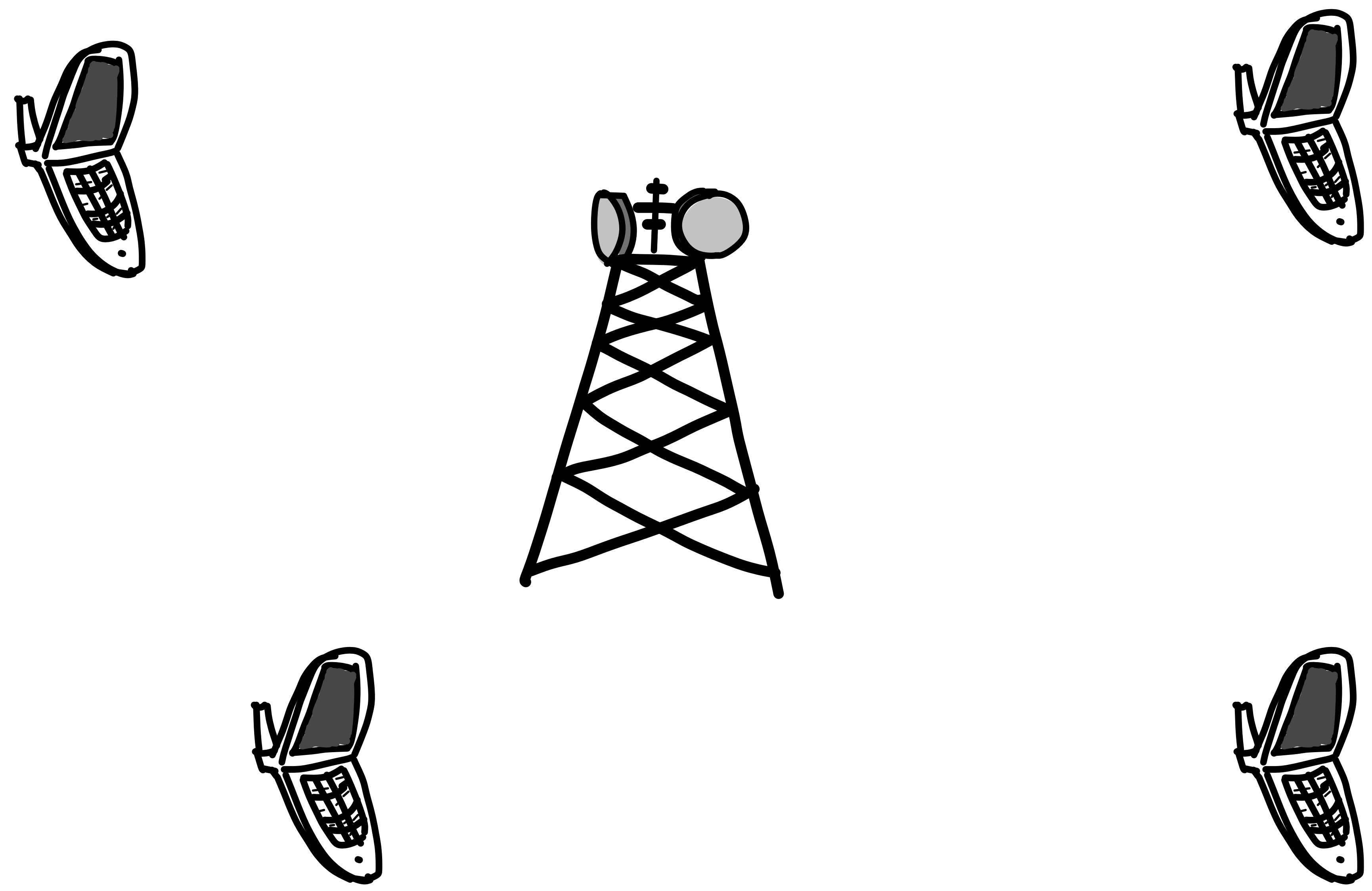
Comparison

All time figures in milliseconds

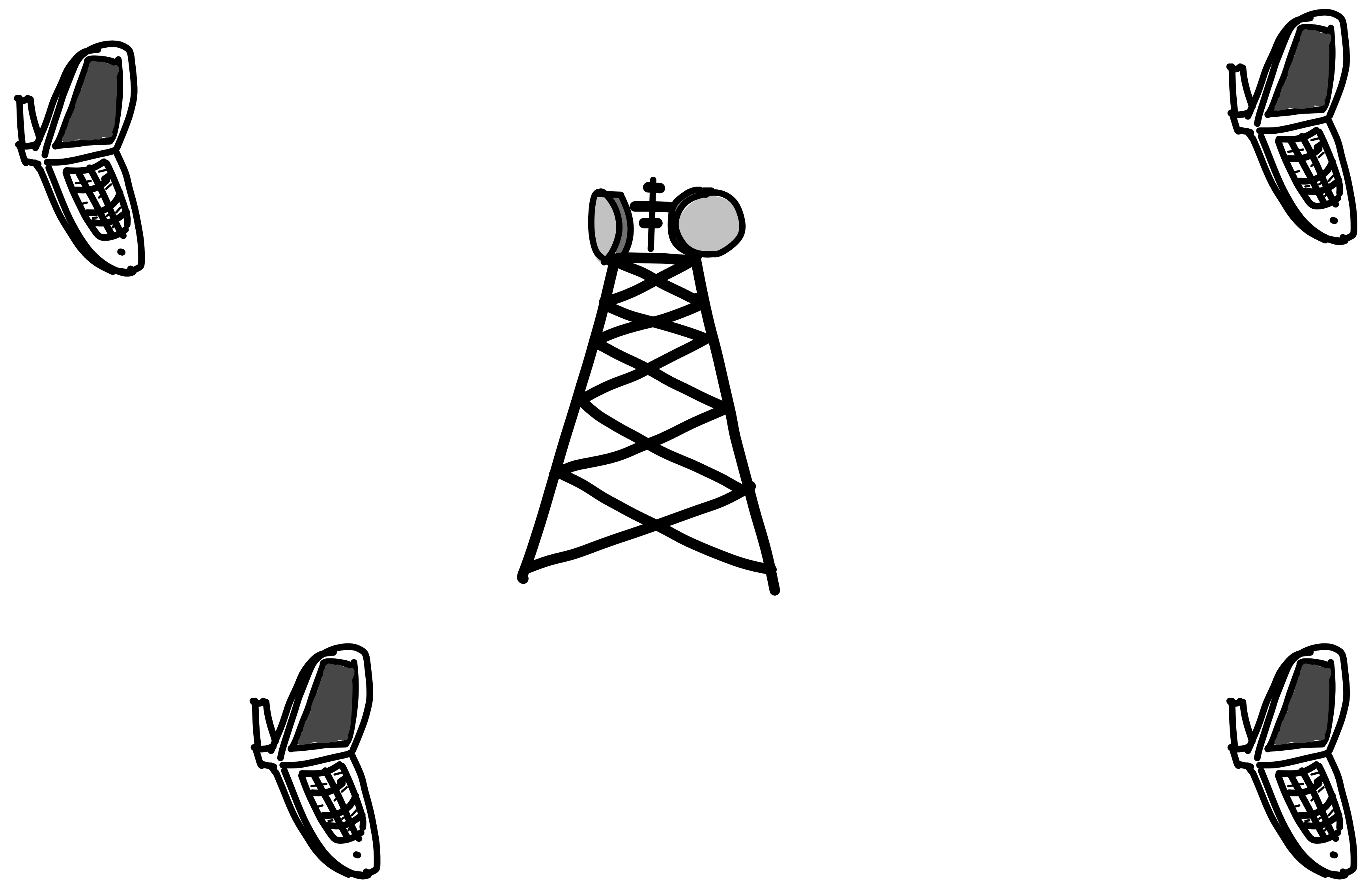
Protocol	Signing $t = 2$	Signing $t = 20$	Setup $n = 2$	Setup $n = 20$
This Work	**9.5**	**31.6**	**45.6**	**232**
GG18	77	509	–	–
LNR18	304	5194	$\sim$11000	$\sim$28000

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):**

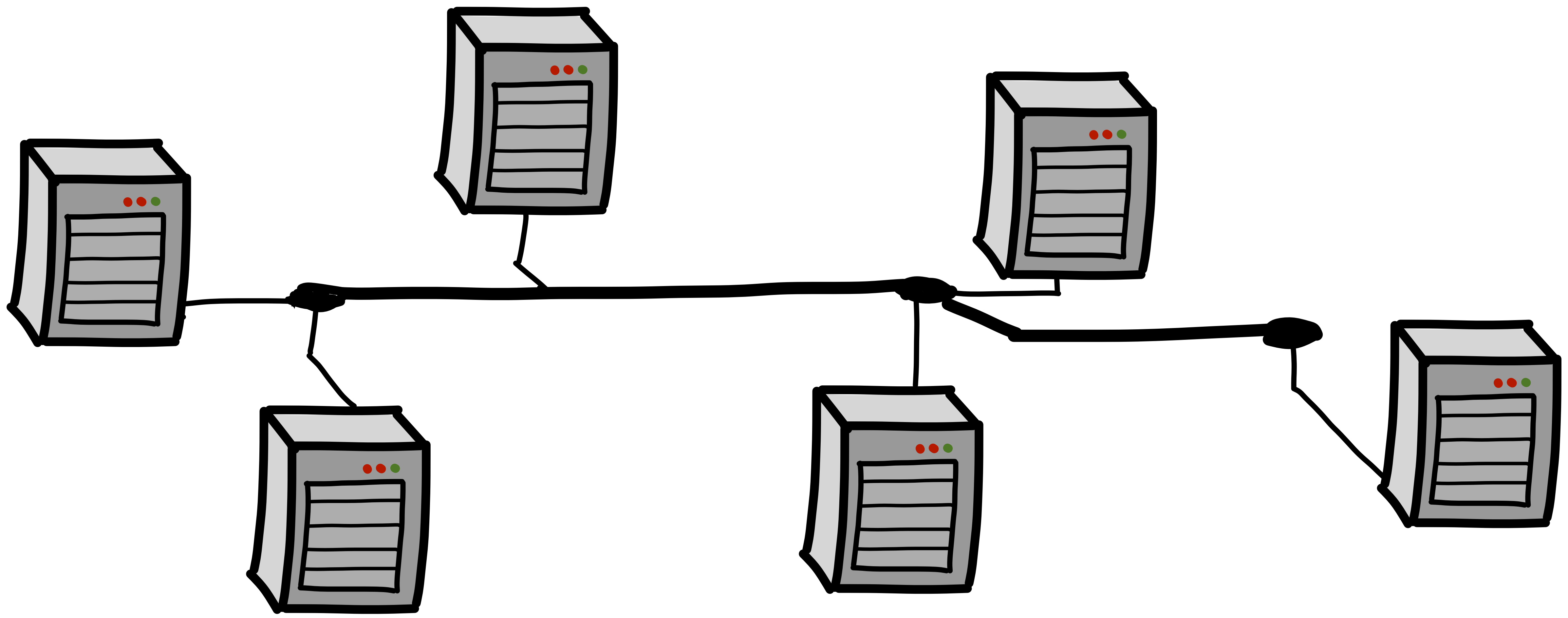
Is communication the bottleneck?

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

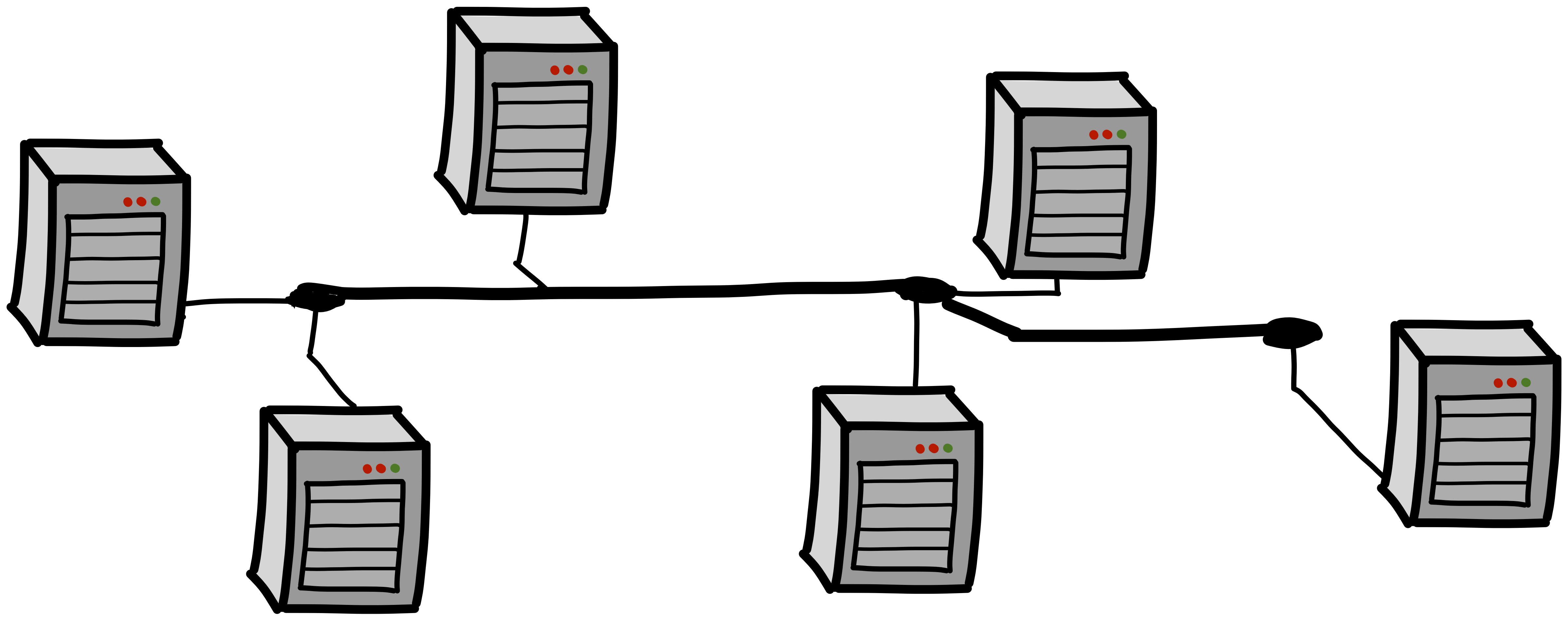
Is communication the bottleneck?

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

Is communication the bottleneck?

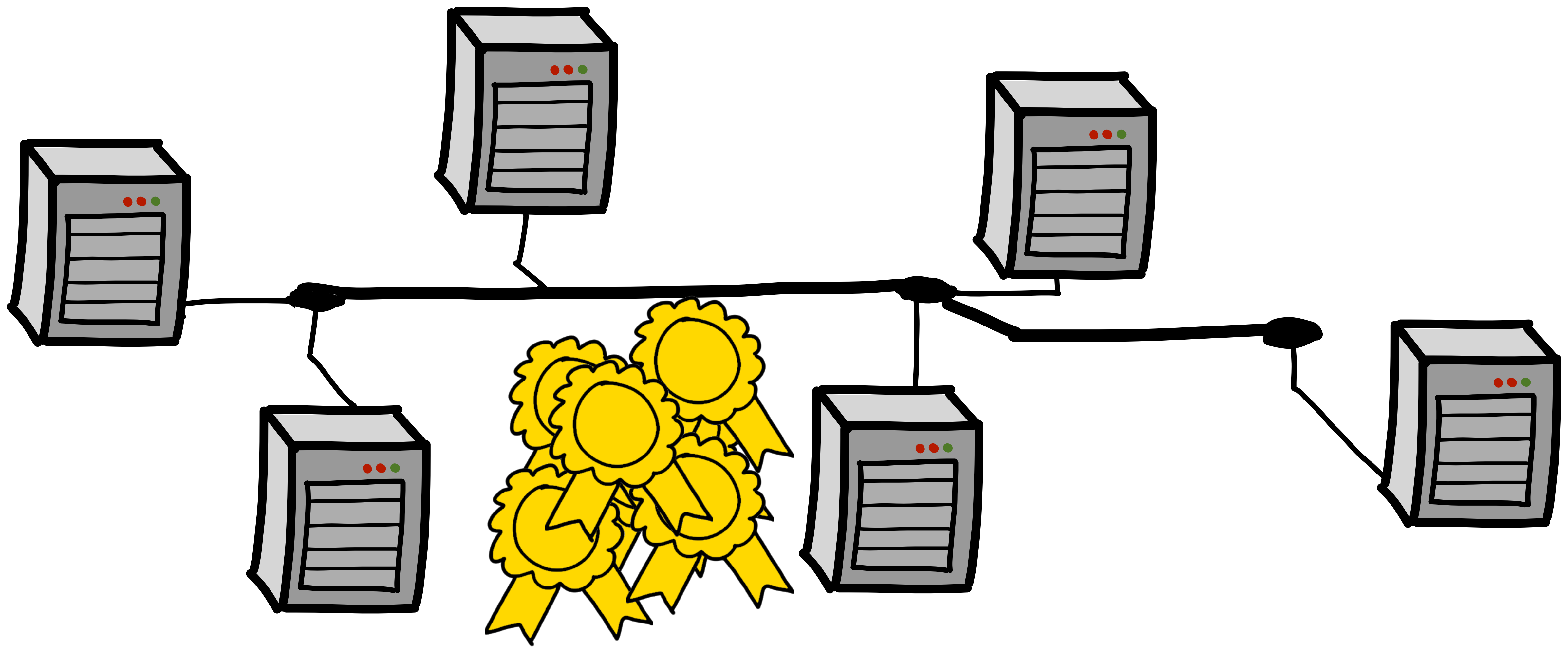


Is communication the bottleneck?



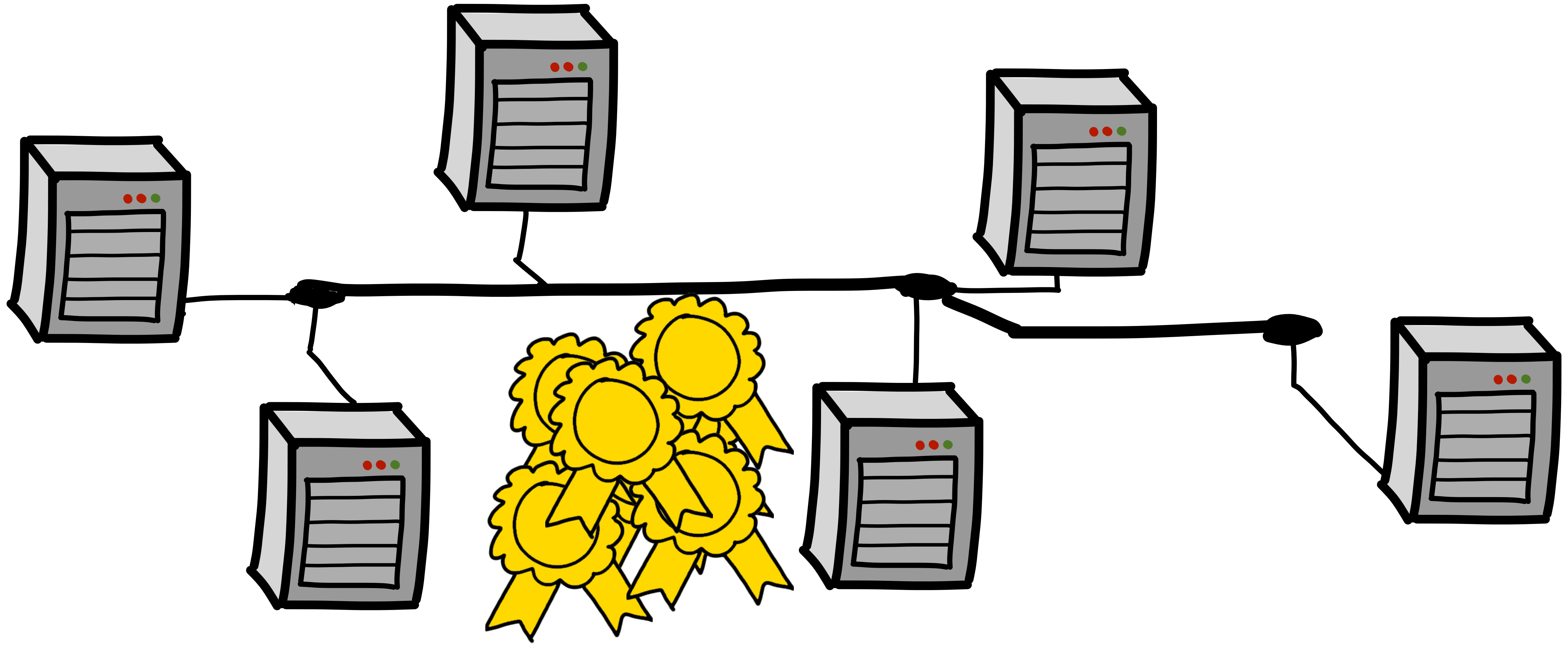
- **Large-scale automated distributed signing:**

Is communication the bottleneck?



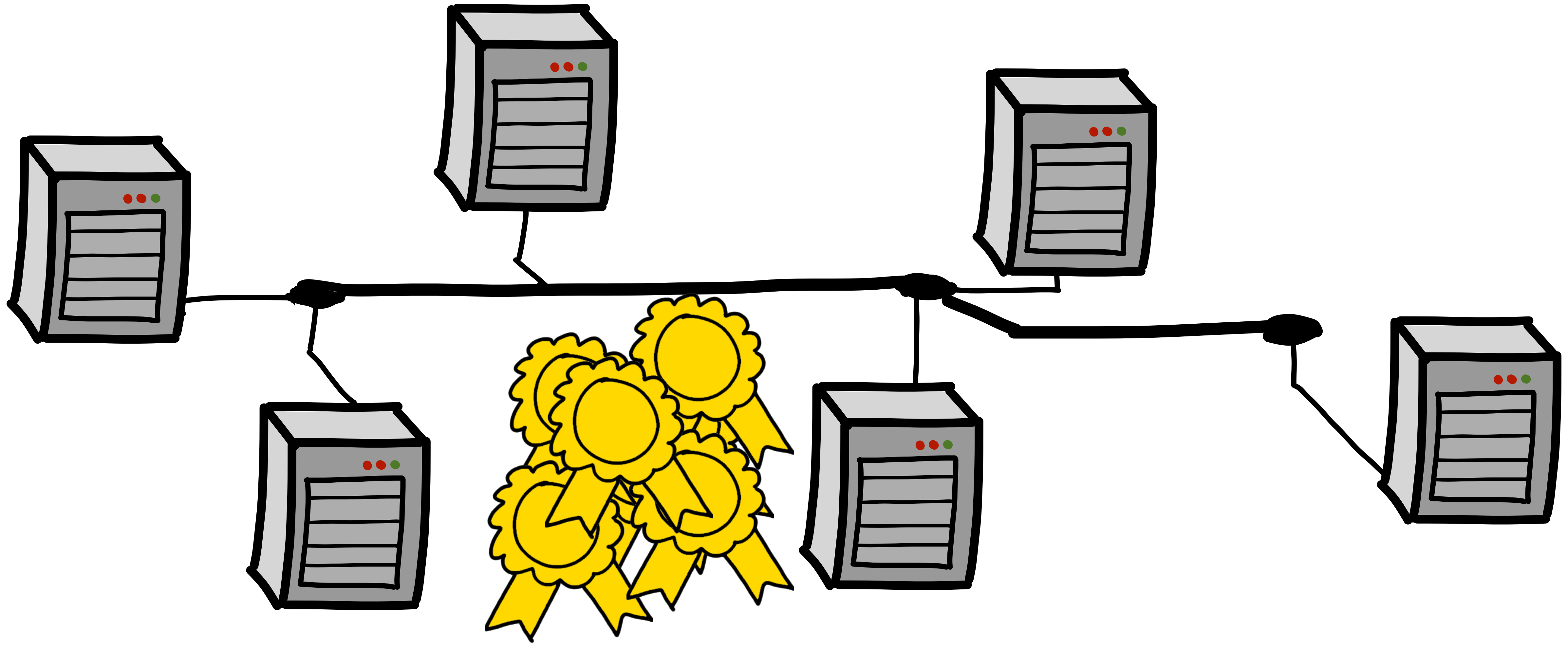
- **Large-scale automated distributed signing:**

Is communication the bottleneck?



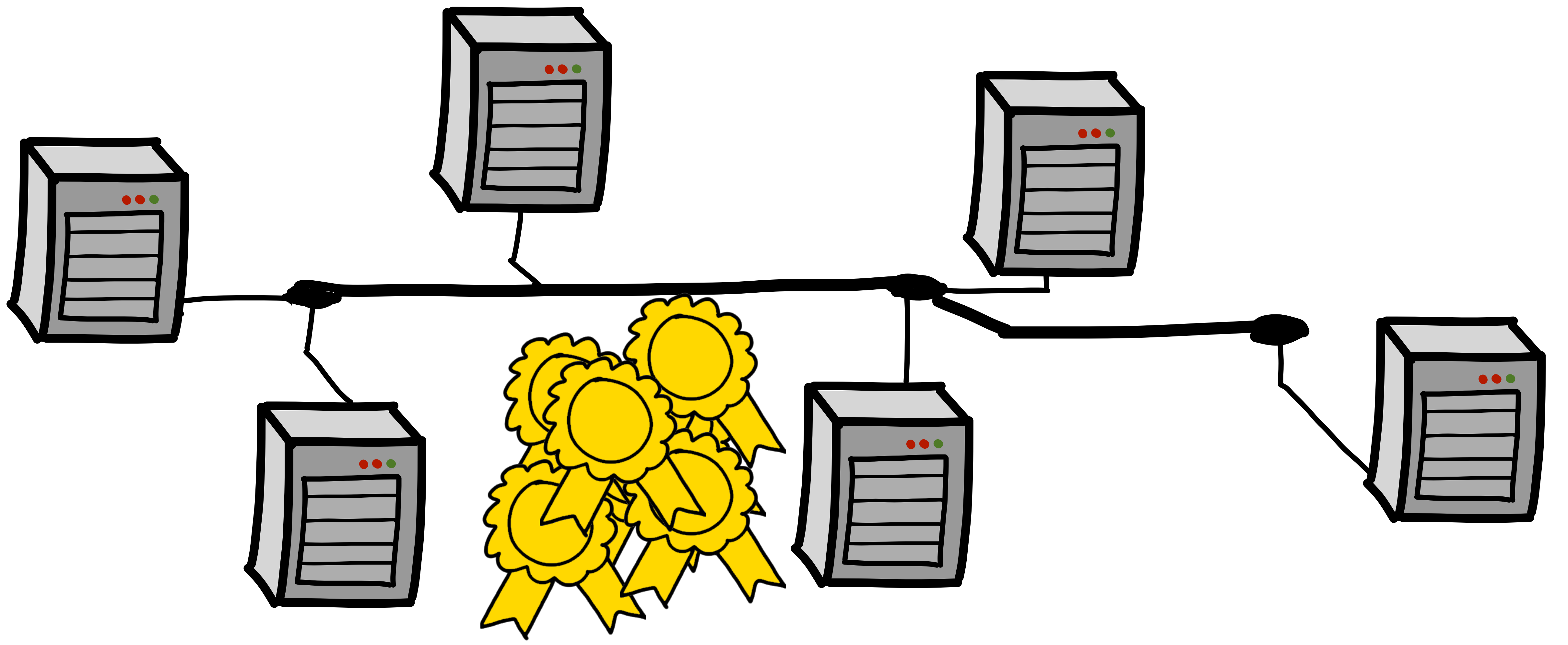
- **Large-scale automated distributed signing:**
  - Threshold 2:   3.8ms/sig   <= ~263 sig/second

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 <500Mb bandwidth

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