threshold ecdsa learnings

Jack Doerner Yash Kondi Eysa Lee abhi shelat

Northeastern University

I’ll save you frustration by skipping the part where I explain what threshold signing is because Elizabeth + team have covered it well.

But did they explain security notions?

N-1 security

Security-with-abort assuming at least 1 honest party.

Identifiable abort is also possible.

This talk

But did they explain security notions?

N-1 security

Security-with-abort assuming at least 1 honest party.

Identifiable abort is also possible.

This talk

N/2 security

Assuming honest majority makes some issues easier.

Threshold ECDSA Challenges

- (Distributed) KeyGeneration of ECDSA and EdDSA is identical to Schnorr
- Signing is where we encounter troublesome non-linearity

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

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

$$R = k \cdot G$$

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

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

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

output σ

Threshold ECDSA Challenges

- (Distributed) KeyGeneration of ECDSA and EdDSA is identical to Schnorr
- Signing is where we encounter troublesome non-linearity

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

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

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

$$k \leftarrow \mathbb{Z}_q$$
$$R = k \cdot G$$
$$e = H(m)$$

Threshold ECDSA Challenges

- (Distributed) KeyGeneration of ECDSA and EdDSA is identical to Schnorr
- Signing is where we encounter troublesome non-linearity

SchnorrSign(sk, m) :

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

output σ

ECDSASign(sk, m) :

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

output $\sigma = (s, R)$

Threshold ECDSA Challenges

- (Distributed) KeyGeneration of ECDSA and EdDSA is identical to Schnorr
- Signing is where we encounter troublesome non-linearity

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

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

output σ

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

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

output $\sigma = (s, R)$

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

$$e = H(R \| m)$$
$$s = k - \mathsf{sk} \cdot e$$
$$\sigma = (s, R)$$

output σ

Threshold ECDSA Challenges

- (Distributed) KeyGeneration of ECDSA and EdDSA is identical to Schnorr
- Signing is where we encounter troublesome non-linearity

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

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

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

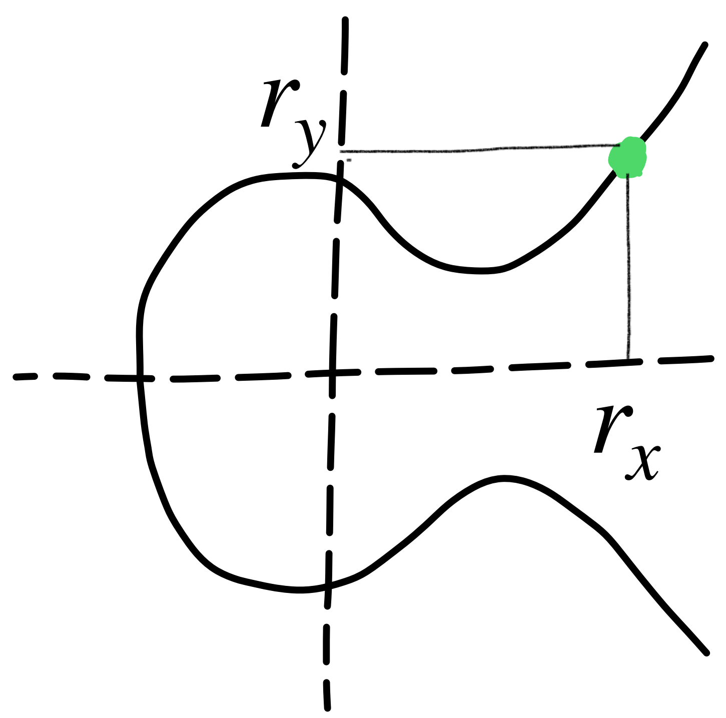
$$k \leftarrow \mathbb{Z}_q$$
$$R = k \cdot G$$
$$e = H(m)$$
$$s = \frac{e + \mathsf{sk} \cdot r_x}{k}$$
output $\sigma = (s, R)$

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

$$k = F(\mathsf{sk}, m)$$
$$R = k \cdot G$$
$$e = H(R \| m)$$
$$s = k - \mathsf{sk} \cdot e$$
$$\sigma = (s, R)$$
output σ

Threshold ECDSA Challenges

$R = (r_x, r_y)$



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

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

$$R = k \cdot G$$

$$e = H(m)$$

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

output $\sigma = (s, R)$

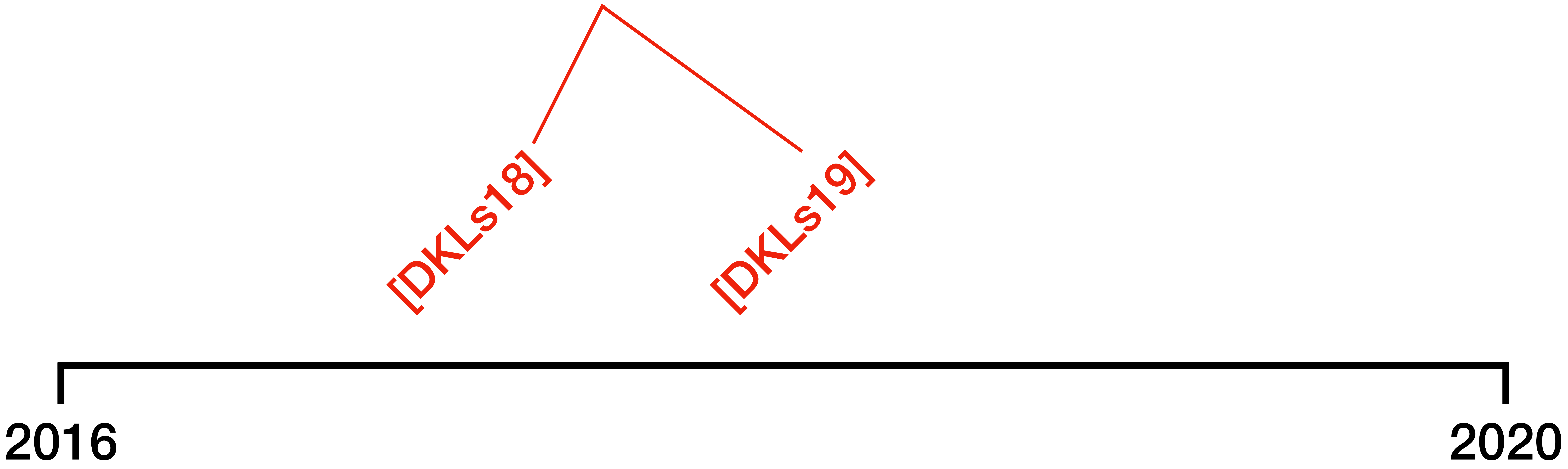
Multiplication of secret values

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

Division (Modular inverse)

N-1 security

Uses only ECDSA assumption,
Employs an efficient check against malicious adversary.



N-1 security

Relies on Paillier

Relies on Class Groups

Relies on more generic MPC.

Our Key advantage

Additive Homorphic Encryption (e.g. Paillier) implement the mult + inv.

Adds extra assumption, heavy computation, seems to require tricky ZK proofs.

Our Key advantage

Additive Homorphic Encryption (e.g. Paillier) implement the mult + inv.

Adds extra assumption, heavy computation, seems to require tricky ZK proofs.

ST19 implements any MPC that computes

$$g^{f(x,x^{-1})}$$

Idea: SPDZ Mac in the exponent.

Adds 2x & >13 rounds due to extra statistical MACs

Our Key advantage

Additive Homomorphic Encryption (e.g. Paillier) implement the mult + inv.

Adds extra assumption, heavy computation, seems to require tricky ZK proofs.

ST19 implements any MPC that computes

$$g^{f(x,x^{-1})}$$

Idea: SPDZ Mac in the exponent.

Adds 2x & >13 rounds due to extra statistical MACs

Our family of protocols exploit a computational self-MAC created by a non-linear operation in the exponent.

Faster, fewer rounds.

DOCSS19 eval

		LAN		Continental WAN		Worldwide WAN	
	n	Sig(ms)	KGen(ms)	Sig(ms)	KGen(ms)	Sig(ms)	KGen (ms)
Rep3	3	2.78	1.45	27.22	29.44	367.87	291.32
Shamir	3	3.02	1.39	78.75	35.52	1140.09	486.82
Mal. Rep3	3	3.45	1.57	82.14	39.97	1128.01	429.47
Mal. Shamir	3	4.43	1.89	174.95	37.35	2340.53	485.11
MASCOT	2	6.56	4.32	196.19	185.71	2688.92	2632.07
MASCOT–	2	3.61	4.41	54.38	181.12	729.08	2654.59
DKLS [20]	2	3.58	43.73	15.33	109.80	234.37	1002.97
Unbound [43]	2	11.33	315.96	31.08	424.02	490.73	1010.98
Kzen [36]	2	310.71	153.87	1282.81	577.67	14441.83	7237.93

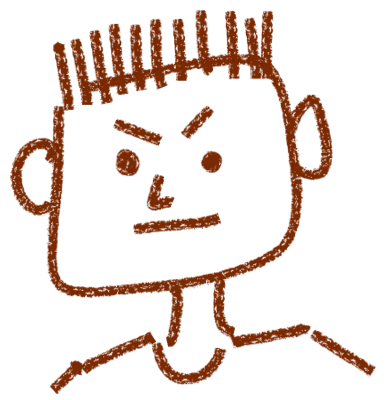
Table 1: Comparison with prior work. Numbers for our protocols are obtained by taking the mean over the maximum execution time over many runs.

2-out-of-2

Xue et al 2021

Signing Protocols	Computation		Communication		Passes
	offline	online	offline	online	
LNR18 [26]	28E + 157M (461ms)	14E + 121M (302ms)	$32\ell_N + 67\kappa$ (12KB)	$16\ell_N + 51\kappa$ (6.6KB)	8
GG18 [19]	42E + 40M (1237ms)	17M (3ms)	$40\ell_N + 18\kappa$ (15.5KB)	9κ (288B)	9
CGGMP20 [6]	208E + 44M (2037ms)	2M (0.2ms)	$118\ell_N + 20\kappa$ (44KB)	κ (32B)	4
2ECDSA (Paillier)	14E + 11M (226ms)	2M (0.2ms)	$16\ell_N + 11\kappa$ (6.3KB)	κ (32B)	3
Lin17 [25] (Paillier-EC)	2E + 8M (34ms)	1E + 2M (8ms)	12κ (192B)	$2\ell_N$ (768B)	3
GG18 [19] (Paillier-EC)	18E + 40M (360ms)	17M (3ms)	$16\ell_N + 18\kappa$ (6.6KB)	9κ (288B)	9
2ECDSA (Paillier-EC)	8E + 14M (141ms)	2M (0.2ms)	$10\ell_N + 12\kappa$ (4.1KB)	κ (32B)	3
CCLST19 [7]	4E + 8M (475ms)	1E + 2M (190ms)	6κ (208B)	14κ (505B)	3
CCLST20 [8]	28E + 8M (3316ms)	17M (3ms)	140κ (4.5KB)	9κ (288B)	8
YCX21 [33]	28E + 8M (4550ms)	17M (3ms)	140κ (4.5KB)	9κ (288B)	8
2ECDSA (CL)	11E + 11M (1386ms)	2M (0.2ms)	53κ (1.7KB)	κ (32B)	3
DKLS18 [15]	13M (2.9ms)	2M (0.2ms)	$16\kappa^2$ (169.8KB)	κ (32B)	2
DKLS19 [16]	13M (3.7ms)	2M (0.2ms)	$20\kappa^2$ (180KB)	κ (32B)	7
2ECDSA (OT)	11M (2.6ms)	2M (0.2ms)	$8\kappa^2$ (90.9KB)	κ (32B)	3

Important cases

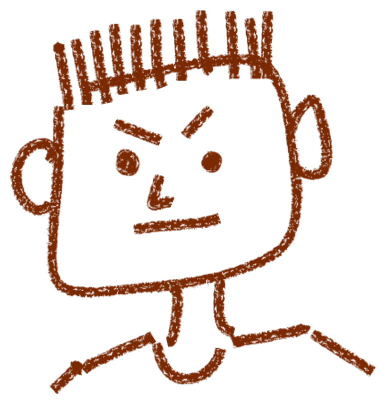

2-out-of-2




k-out-of-k

Credit: Eysa for drawings

Important cases

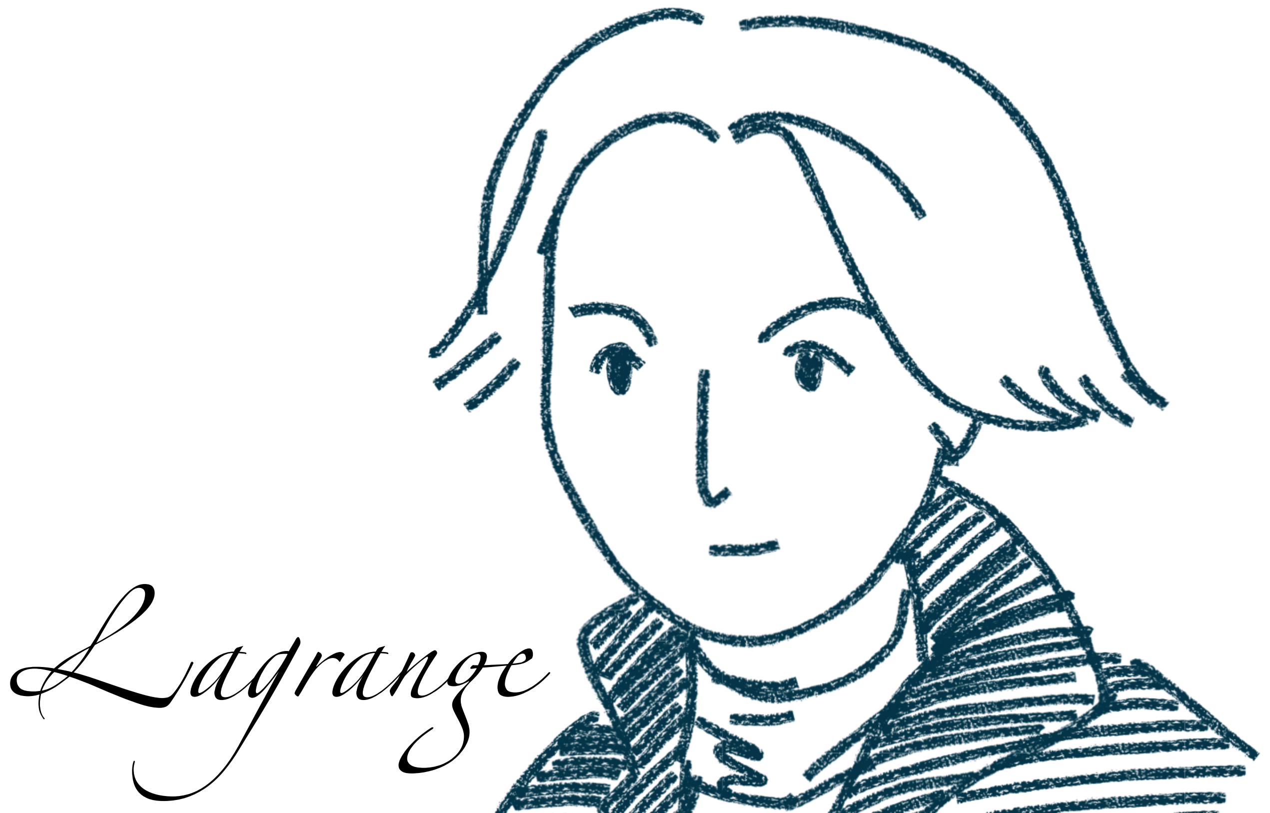


2-out-of-2

k-out-of-k

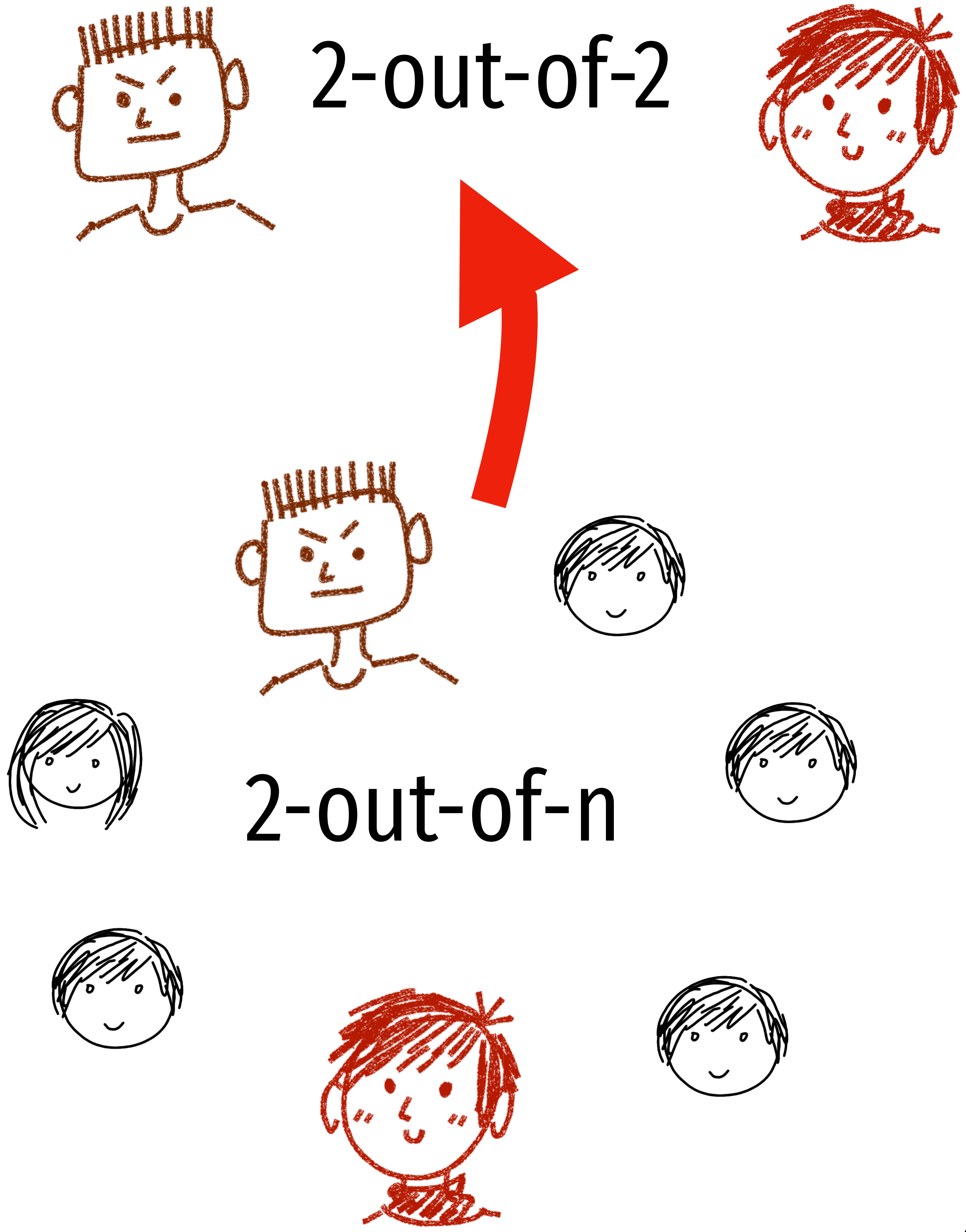
One Idea

$$sk = \sum_{j=0}^{k} sk_j \ell_j(0)$$



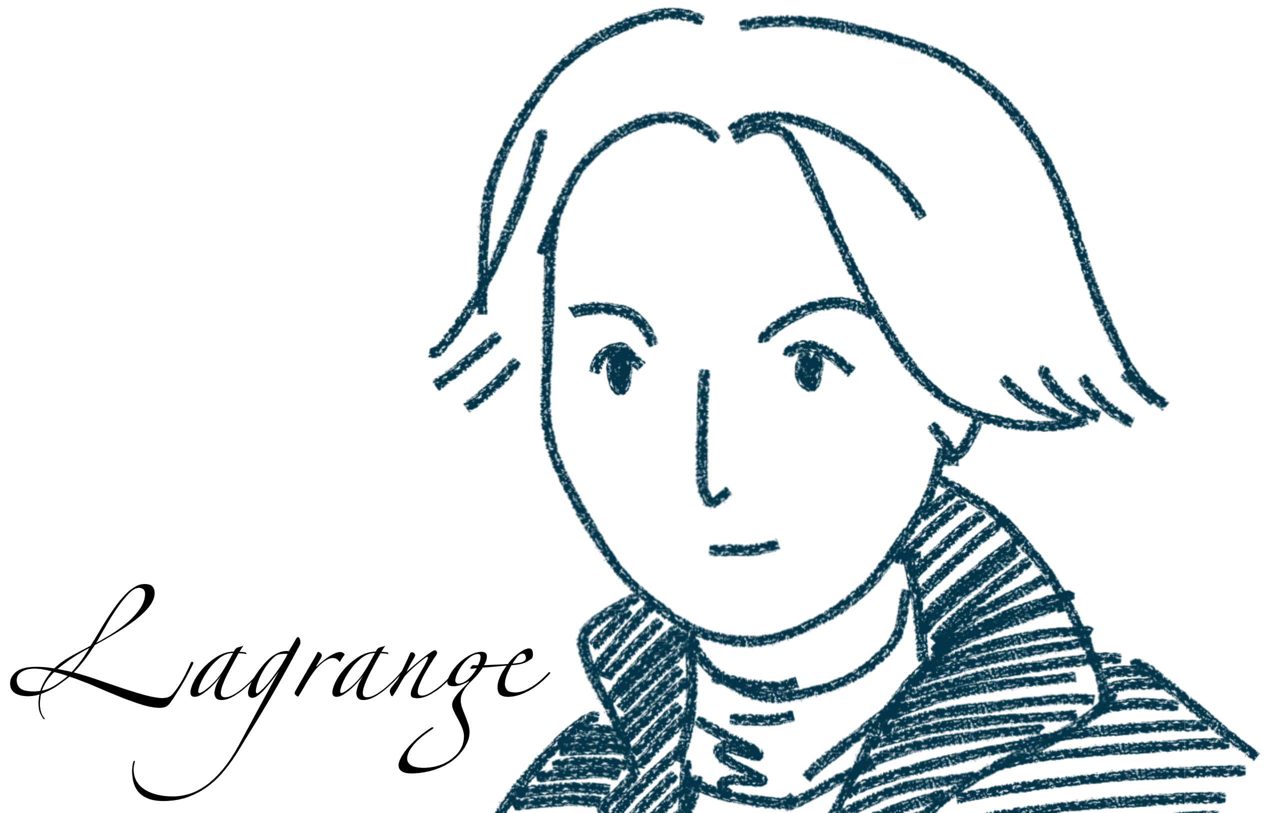
Credit: Eysa for drawings

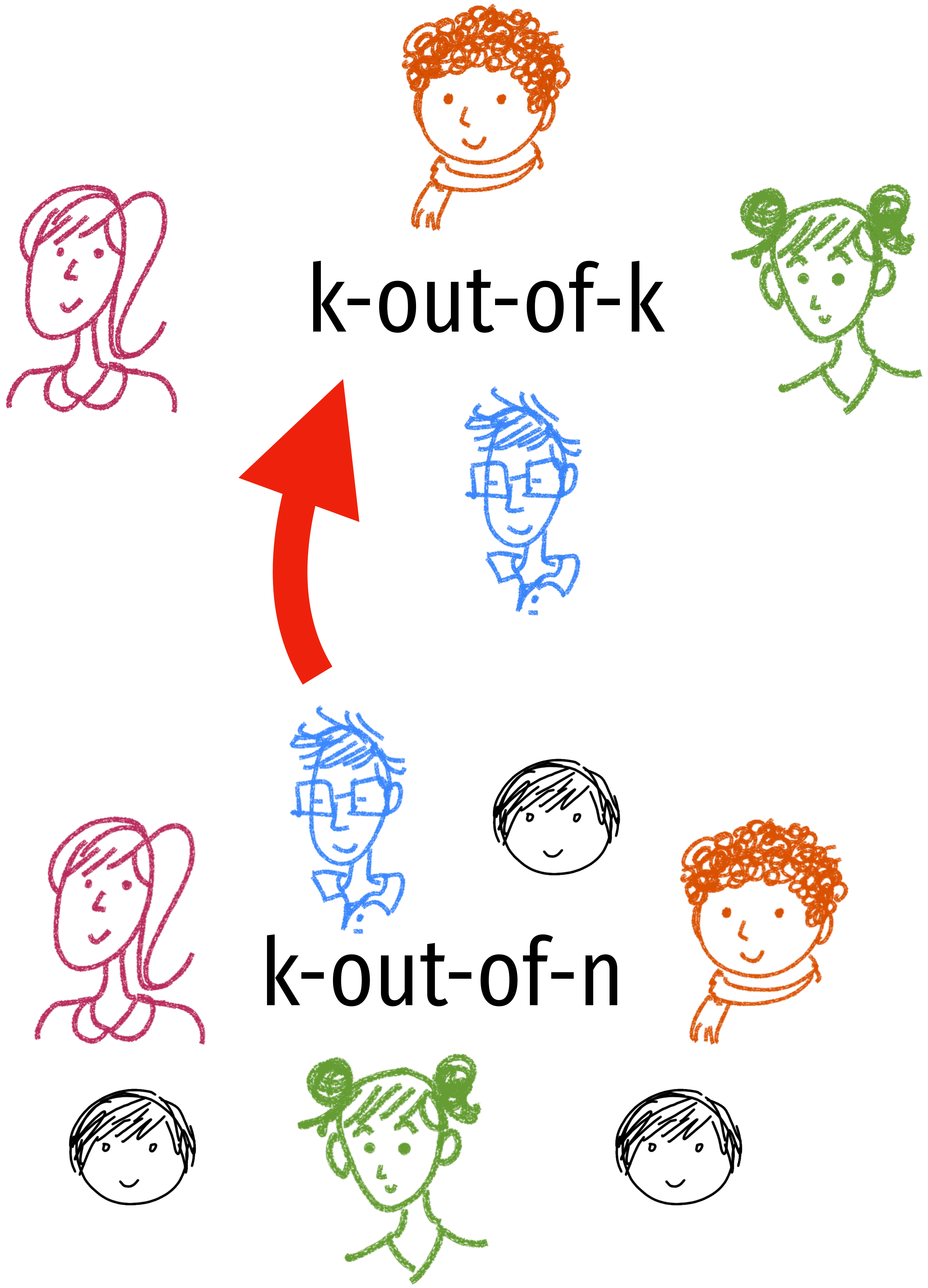
Important cases



One Idea

$$sk = \sum_{j=0}^{k} sk_j \ell_j(0)$$



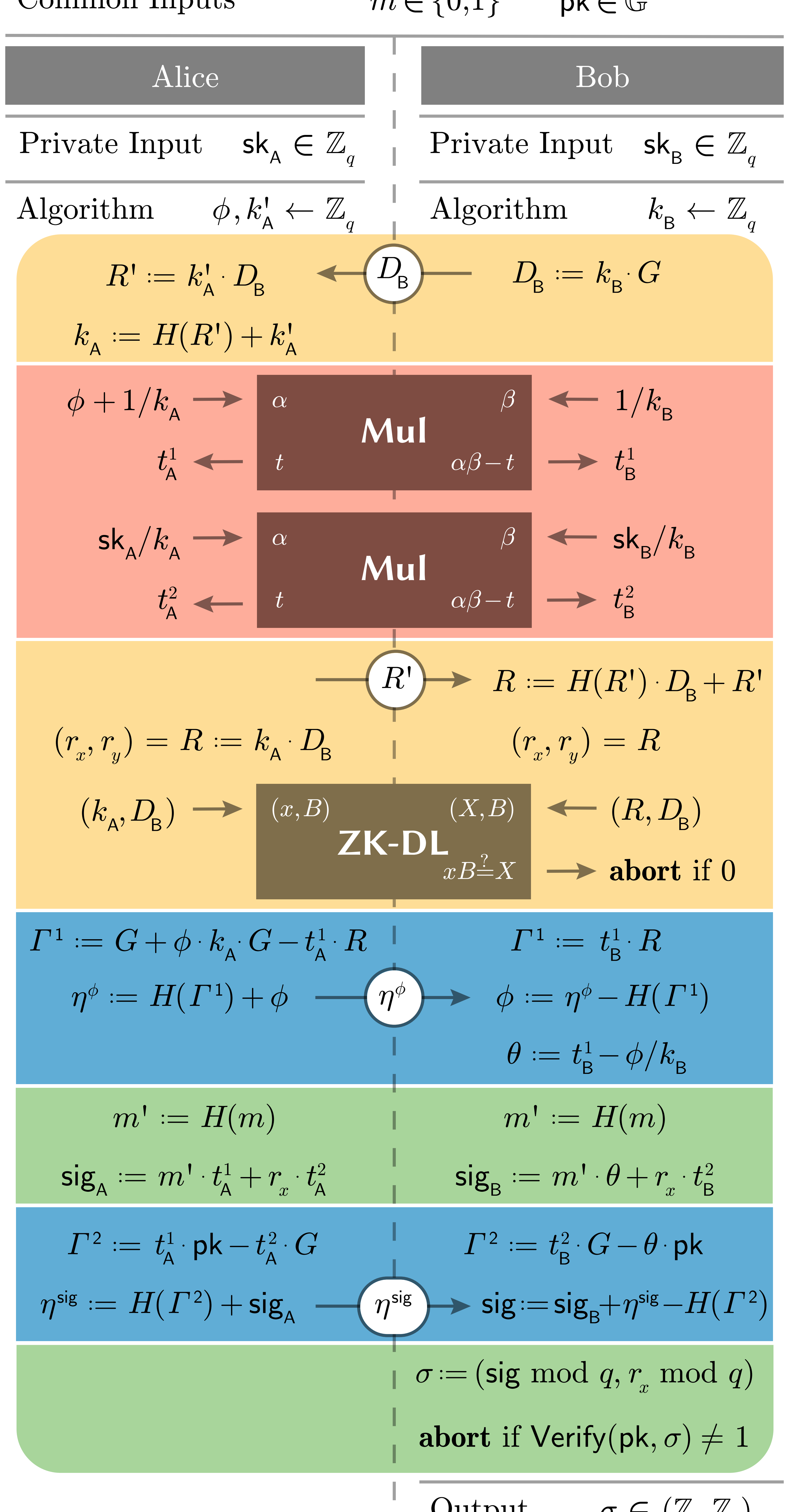


Credit: Eysa for drawings

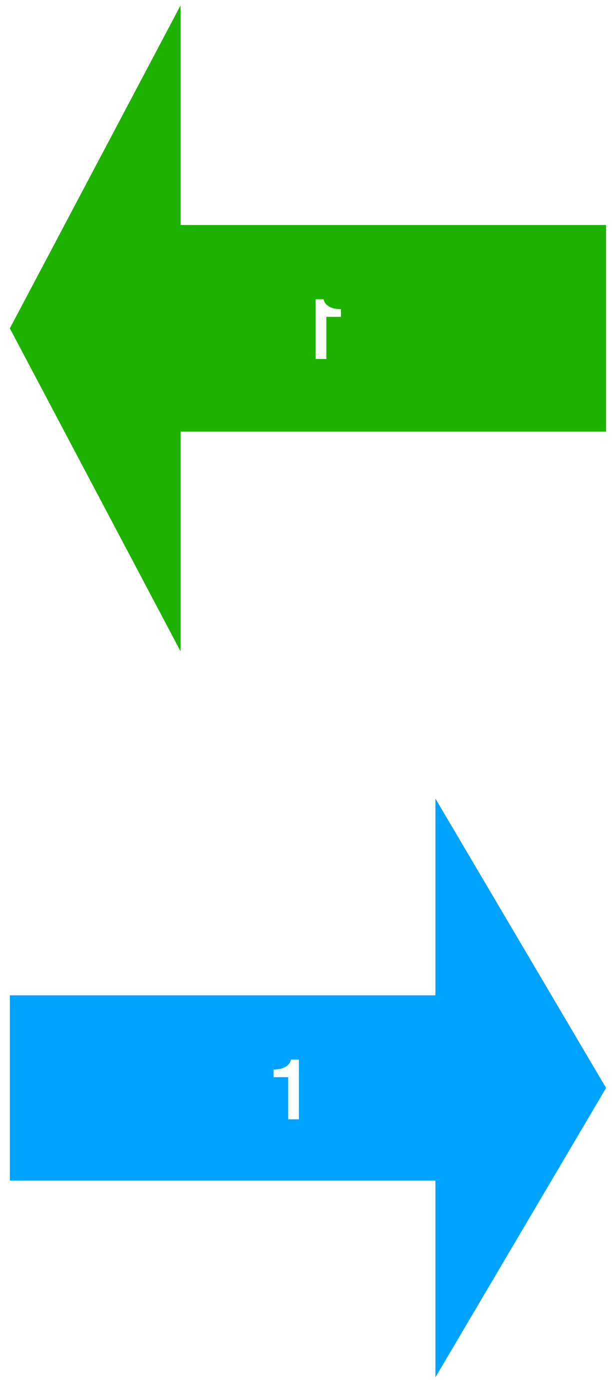
1

Improvements we've discovered while implementing and helping other teams implement.

\\\\\\\\\\\\\\\\ Thanks to Ben Diamond, Arash Afshar, Matthias Geihs, Ben Riva, Lance Roy, Samuel Ranellucci, Yehuda Lindell, Lucas Meier, Web3Auth, Sepior \\\\\\\\\\\\\\\\ in random order



2-out-of-2 from 2018



1 message from Bob to Alice
1 response from Alice to Bob

Functionality 2. $\mathcal{F}_{\mathsf{SampledECDSA}}$:

DKLS18

Ideal functionality

This functionality is parametrized in a manner identical to $\mathcal{F}_{\mathsf{ECDSA}}$. Note that Alice may engage in the Offset Determination phase as many times as she wishes.

Setup (2-of-n): On receiving $(\texttt{init})$ from all parties:

1) Sample and store the joint secret key $\mathsf{sk} \leftarrow \mathbb{Z}_q$.
2) Compute and store the joint public key $\mathsf{pk} := \mathsf{sk} \cdot G$.
3) Send $(\texttt{public-key}, \mathsf{pk})$ to all parties.
4) Store $(\texttt{ready})$ in memory.

Instance Key Agreement: On receiving $(\texttt{new}, \mathsf{id}^{\mathsf{sig}}, m, \mathsf{B})$ from Alice and $(\texttt{new}, \mathsf{id}^{\mathsf{sig}}, m, \mathsf{A})$ from Bob, if $(\texttt{ready})$ exists in memory, and if $(\texttt{message}, \mathsf{id}^{\mathsf{sig}}, \cdot, \cdot)$ does not exist in memory, and if Alice and Bob both supply the same message m and each indicate the other as their counterparty, then:

1) Sample $k_{\mathsf{B}} \leftarrow \mathbb{Z}_q$.
2) Store $(\texttt{message}, \mathsf{id}^{\mathsf{sig}}, m, k_{\mathsf{B}})$ in memory.
3) Send $(\texttt{nonce-shard}, \mathsf{id}^{\mathsf{sig}}, D_{\mathsf{B}} := k_{\mathsf{B}} \cdot G)$ to Alice.

Offset Determination: On receiving $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, i, R_i)$ from Alice, if $(\texttt{message}, \mathsf{id}^{\mathsf{sig}}, m, k_{\mathsf{B}})$ exists in memory, but $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, j, \cdot)$ for $j = i$ does not exist in memory:

4) Sample $k_i^{\Delta} \leftarrow \mathbb{Z}_q$.
5) Store $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, i, R_i, k_i^{\Delta})$ in memory.
6) Compute $k_{i,\mathsf{A}}^{\Delta} = k_i^{\Delta} / k_{\mathsf{B}}$ and send $(\texttt{offset}, \mathsf{id}^{\mathsf{sig}}, i, k_{i,\mathsf{A}}^{\Delta})$ to Alice.

Functionality 2. $\mathcal{F}_{\mathsf{SampledECDSA}}$:

DKLS18

This functionality is parametrized in a manner identical to $\mathcal{F}_{\mathsf{ECDSA}}$. Note that Alice may engage in the Offset Determination phase as many times as she wishes.

Setup (2-of-n): On receiving $(\texttt{init})$ from all parties:

1) Sample and store the joint secret key $\mathsf{sk} \leftarrow \mathbb{Z}_q$.
2) Compute and store the joint public key $\mathsf{pk} := \mathsf{sk} \cdot G$.
3) Send $(\texttt{public-key}, \mathsf{pk})$ to all parties.
4) Store $(\texttt{ready})$ in memory.

Instance Key Agreement: On receiving $(\texttt{new}, \mathsf{id}^{\mathsf{sig}}, m, \mathsf{B})$ from Alice and $(\texttt{new}, \mathsf{id}^{\mathsf{sig}}, m, \mathsf{A})$ from Bob, if $(\texttt{ready})$ exists in memory, and if $(\texttt{message}, \mathsf{id}^{\mathsf{sig}}, \cdot, \cdot)$ does not exist in memory, and if Alice and Bob both supply the same message m and each indicate the other as their counterparty, then:

1) Sample $k_{\mathsf{B}} \leftarrow \mathbb{Z}_q$.
2) Store $(\texttt{message}, \mathsf{id}^{\mathsf{sig}}, m, k_{\mathsf{B}})$ in memory.
3) Send $(\texttt{nonce-shard}, \mathsf{id}^{\mathsf{sig}}, D_{\mathsf{B}} := k_{\mathsf{B}} \cdot G)$ to Alice.

Offset Determination: On receiving $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, i, R_i)$ from Alice, if $(\texttt{message}, \mathsf{id}^{\mathsf{sig}}, m, k_{\mathsf{B}})$ exists in memory, but $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, j, \cdot)$ for $j = i$ does not exist in memory:

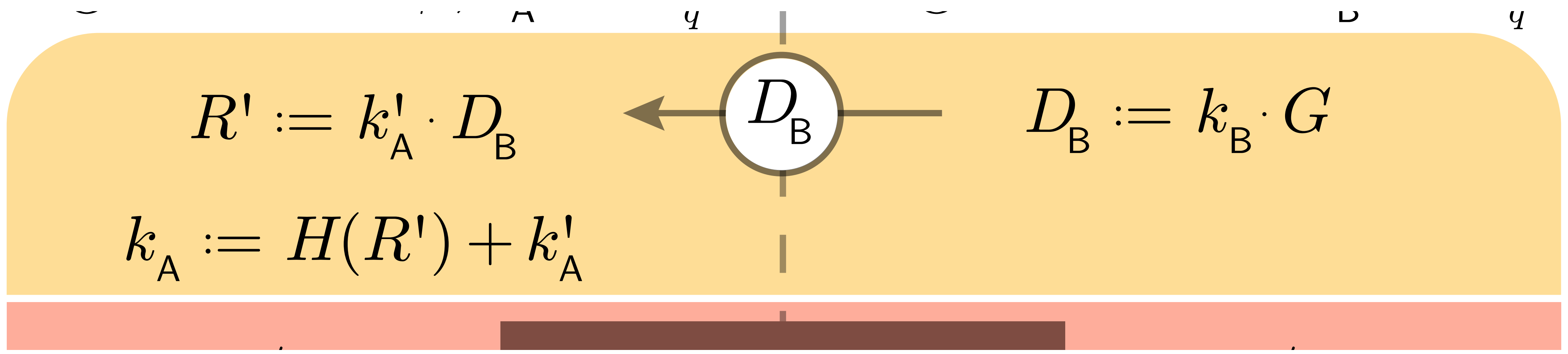
4) Sample $k_i^{\Delta} \leftarrow \mathbb{Z}_q$.
5) Store $(\texttt{nonce}, \mathsf{id}^{\mathsf{sig}}, i, R_i, k_i^{\Delta})$ in memory.
6) Compute $k_{i,\mathsf{A}}^{\Delta} = k_i^{\Delta} / k_{\mathsf{B}}$ and send $(\texttt{offset}, \mathsf{id}^{\mathsf{sig}}, i, k_{i,\mathsf{A}}^{\Delta})$ to Alice.

Ideal functionality

Our old ideal model allowed a benign form of bias in nonce selection.

Secure in the Generic Group Model.

Alice can “grind” alternative R vals.



Functionality 4.1. $\mathcal{F}_{\text{ECDSA-2P}}(\mathcal{G}, n)$: **Two-party ECDSA**

Setup: On receiving $(\texttt{init}, \mathsf{sid})$ from some party $\mathcal{P}_i$ such that $\mathsf{sid} =: \mathcal{P}_1 \| \ldots \| \mathcal{P}_n \| \mathsf{sid}'$ and $i \in [n]$ and sid is fresh, send $(\texttt{init-req}, \mathsf{sid}, i)$ to $\mathcal{S}$. On receiving $(\texttt{init}, \mathsf{sid})$ from all parties,

...skipped...

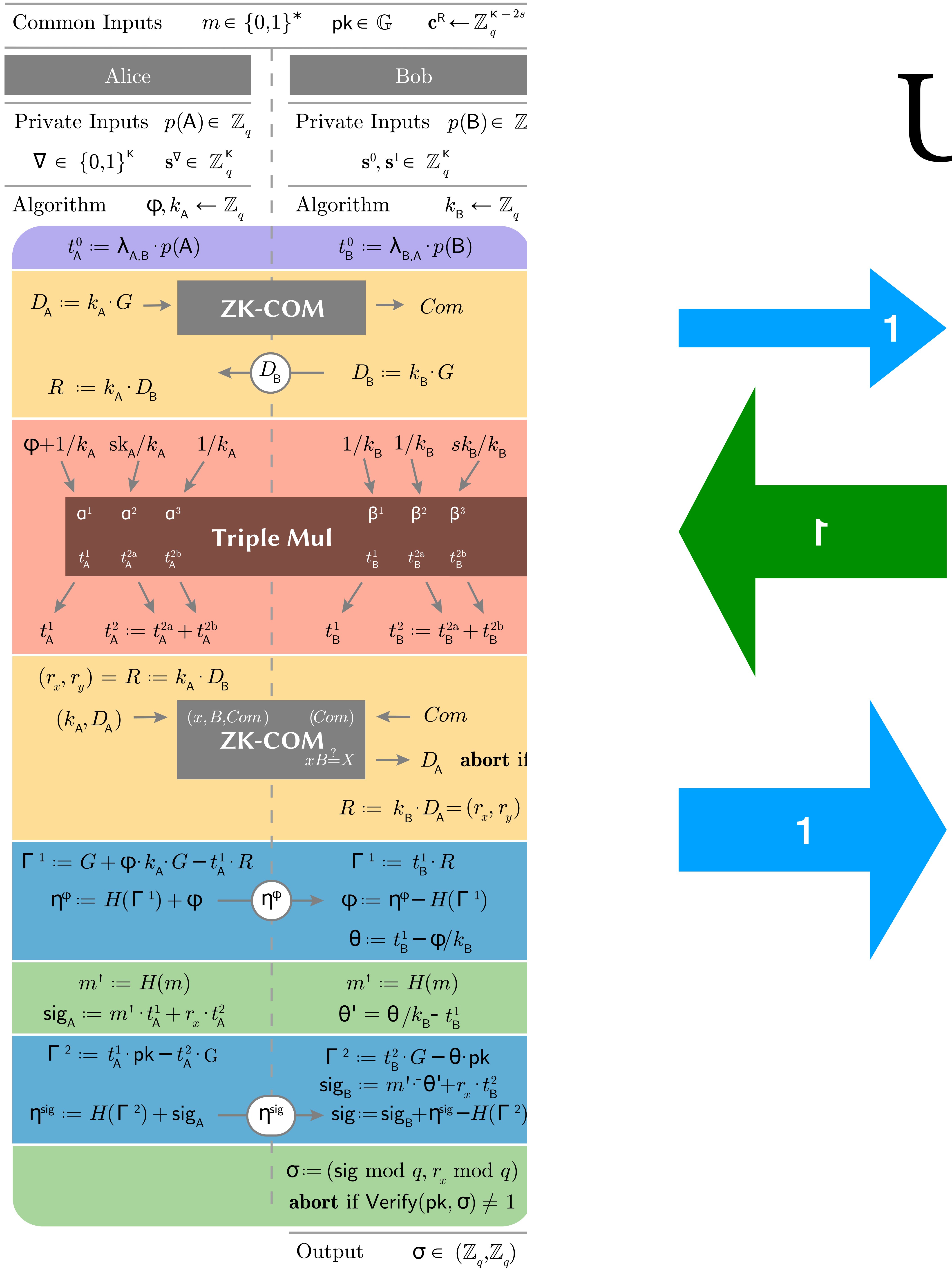
Signing: On receiving $(\texttt{pre-sign}, \mathsf{sid}, \mathsf{sigid})$ from $\mathcal{P}_\mathsf{A}$, parse $\mathsf{sigid} =: \mathsf{A}' \| \mathsf{B} \| \mathsf{sigid}'$, and ignore $\mathcal{P}_\mathsf{A}$'s message if $\mathsf{A}' \neq \mathsf{A}$ or $\mathsf{B} \notin [n]$ or sigid is not fresh or $(\texttt{pk-delievered}, \mathsf{sid}, \mathsf{A})$ does not exist in memory; otherwise, send $(\texttt{ready}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{P}_\mathsf{B}$. When $\mathcal{P}_\mathsf{B}$ subsequently sends $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m)$, if $(\texttt{pk-delievered}, \mathsf{sid}, \mathsf{B})$ exists in memory, then

12. Sample $\sigma \leftarrow \mathsf{ECDSASign}(\mathcal{G}, \mathsf{sk}, m)$ and parse $(s, r^{\mathsf{x}}) := \sigma$.
13. If $\mathcal{P}_\mathsf{A}$ is corrupt, then send $(\texttt{leakage}, \mathsf{sid}, \mathsf{sigid}, r^{\mathsf{x}})$ directly to $\mathcal{S}$.
14. Send $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, m)$ to $\mathcal{P}_\mathsf{A}$.
15. If $\mathcal{P}_\mathsf{A}$ responds to the signature request with $(\texttt{proceed}, \mathsf{sid}, \mathsf{sigid}, m)$ such that the value of m is the same as the one previously supplied by $\mathcal{P}_\mathsf{B}$, then send $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ to $\mathcal{P}_\mathsf{B}$ and ignore all future messages with the signature ID sigid.
16. If $\mathcal{P}_\mathsf{A}$ responds to the signature request with $(\texttt{fail}, \mathsf{sid}, \mathsf{sigid})$, then send $(\texttt{failure}, \mathsf{sid}, \mathsf{sigid})$ to $\mathcal{P}_\mathsf{B}$ and ignore all future messages with the signature ID sigid.

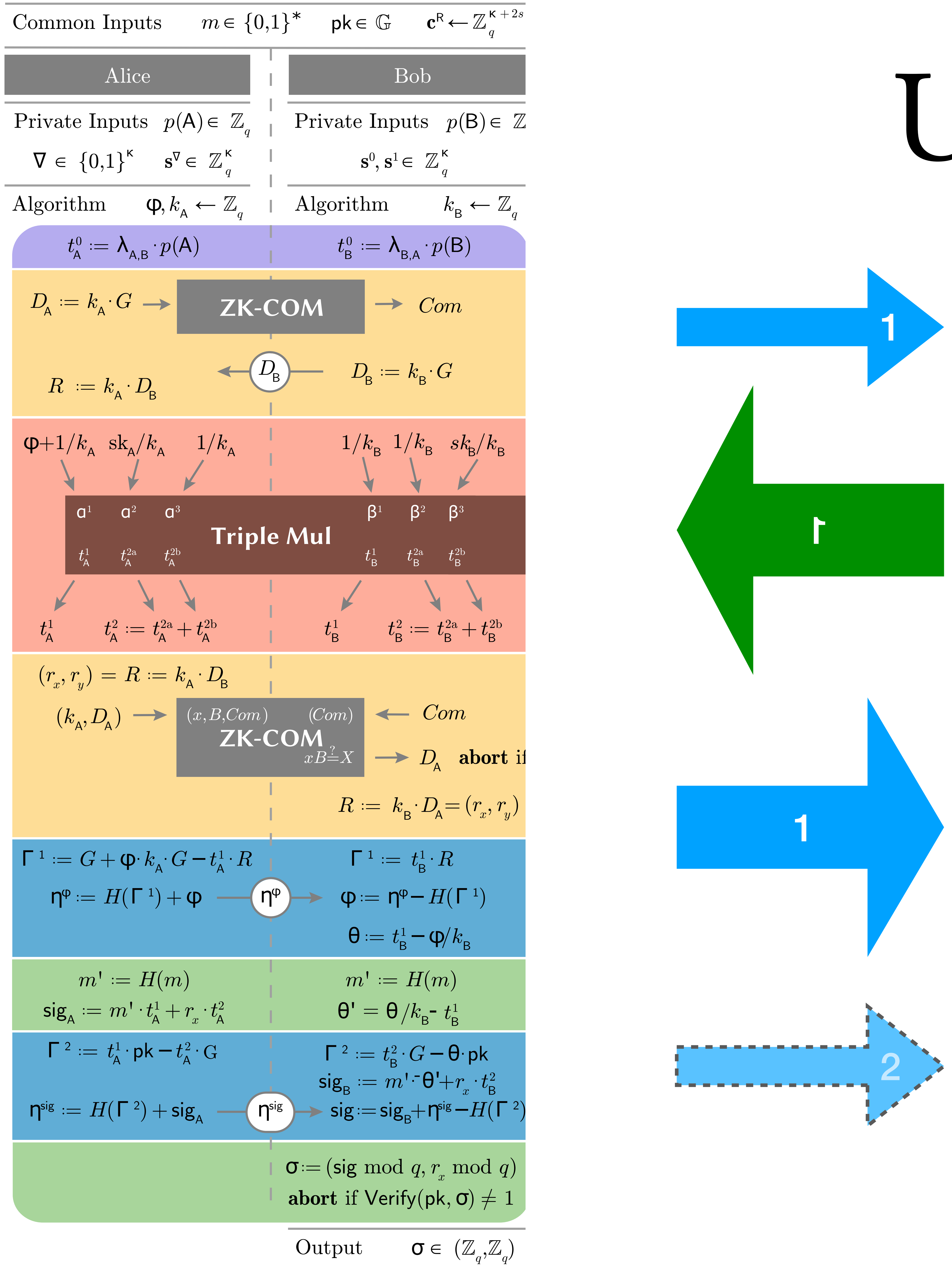
Update: new 2-out-of-n protocol removes bias, but requires 1 more message.

This message can be pipelined (2 messages total).

Updated protocol



Updated protocol

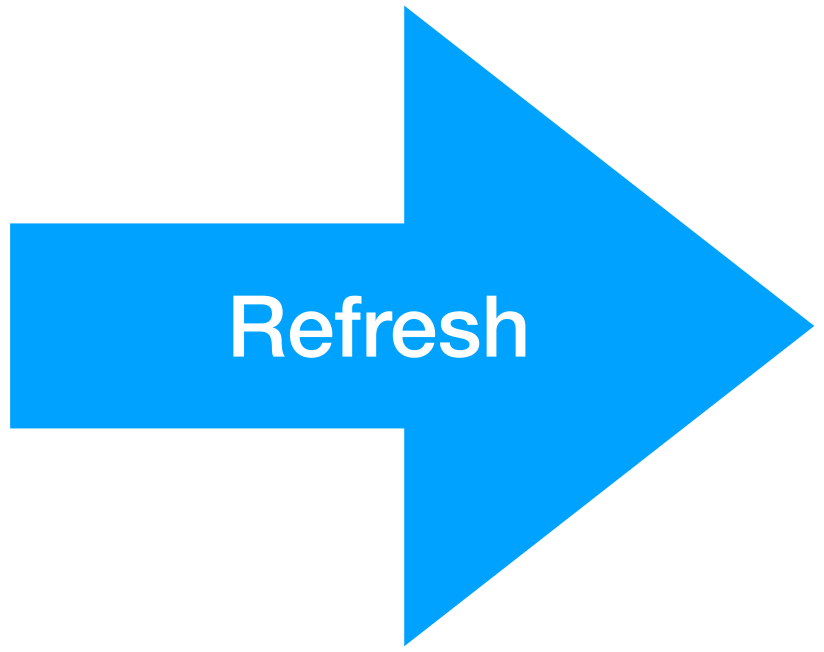


This round can be pipelined with the next instance.

Protocol maintains OT state, so this change is no additional burden.

2 Key refresh (proactive security)

Everyone has a key for pk.

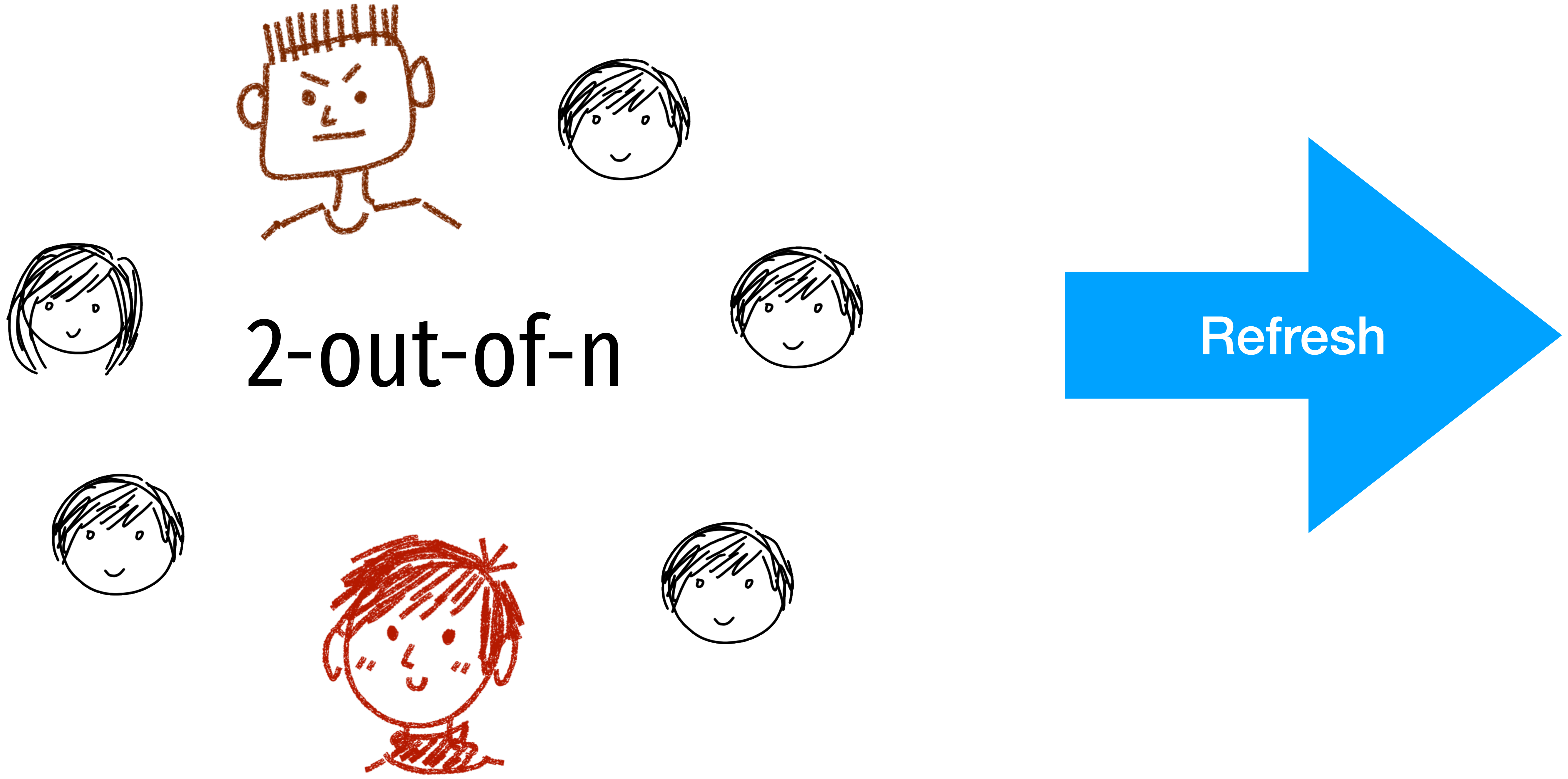


Everyone has a new key for pk.

2 Key refresh (proactive security)

Everyone has a key for pk.

Everyone has a new key for pk.

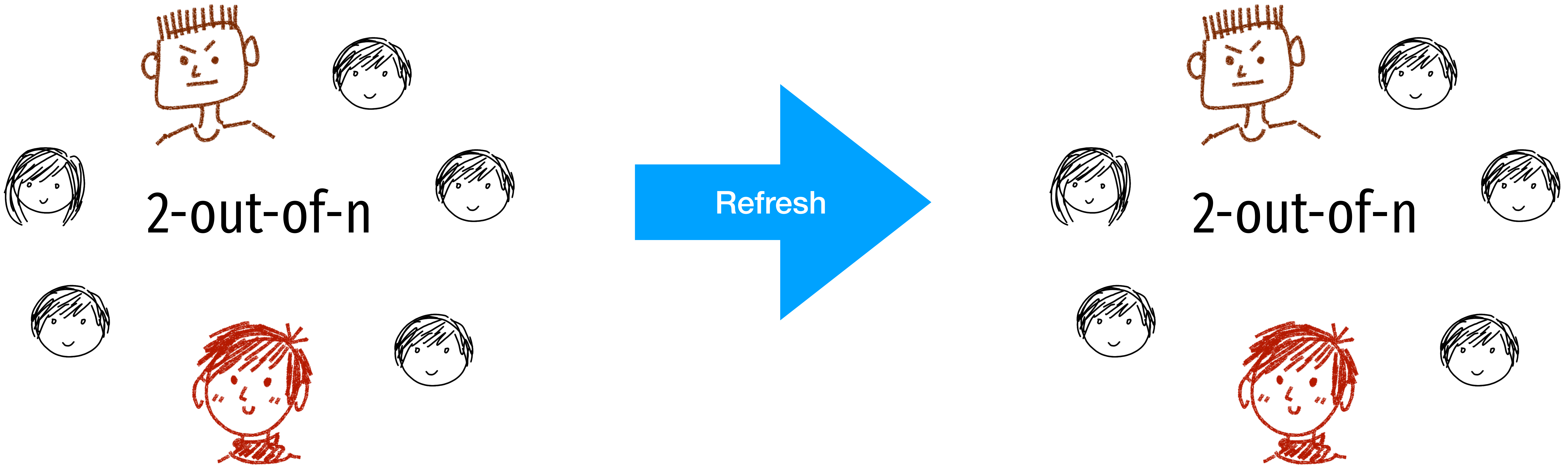


2

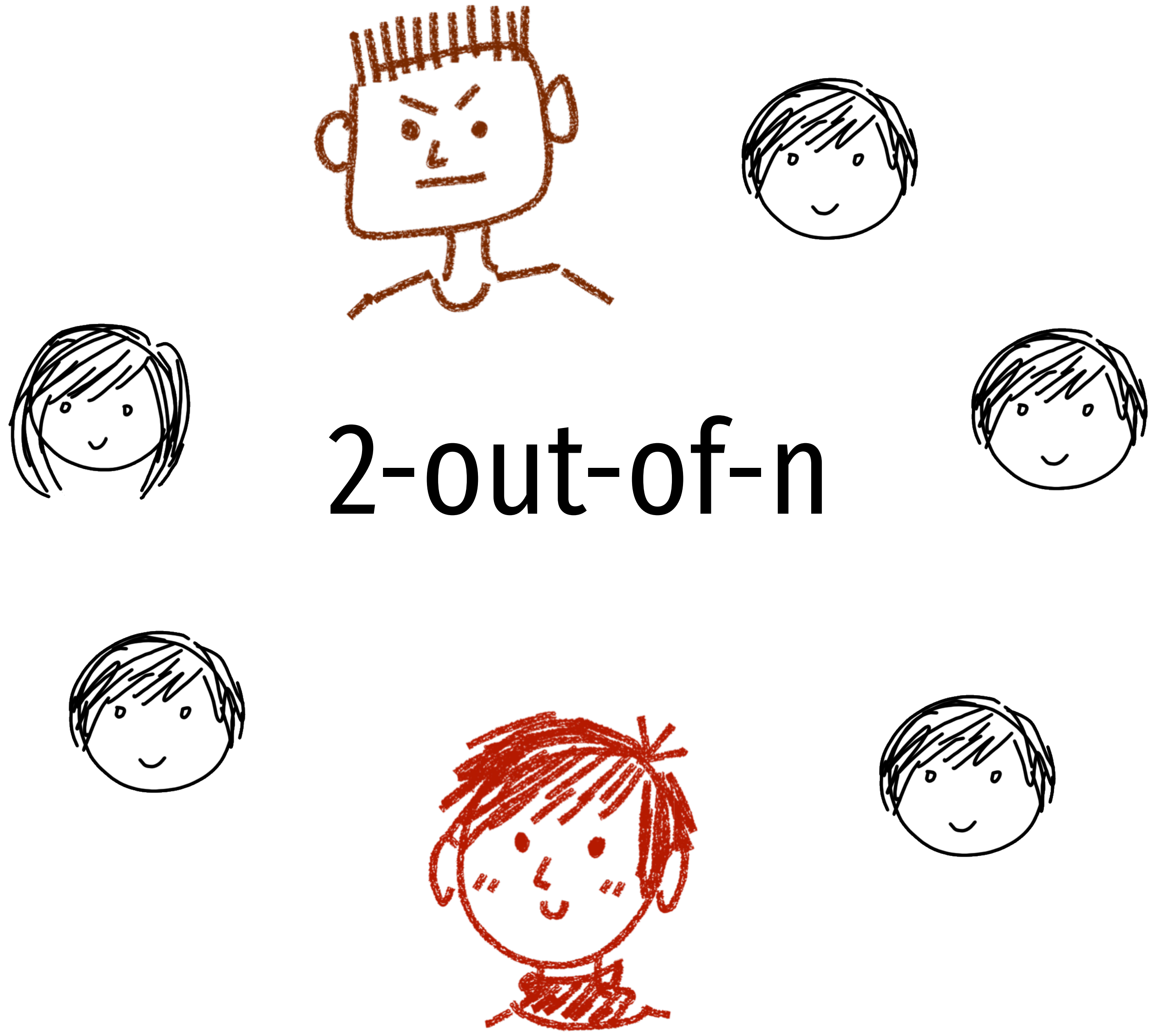
Key refresh (proactive security)

Everyone has a key for pk.

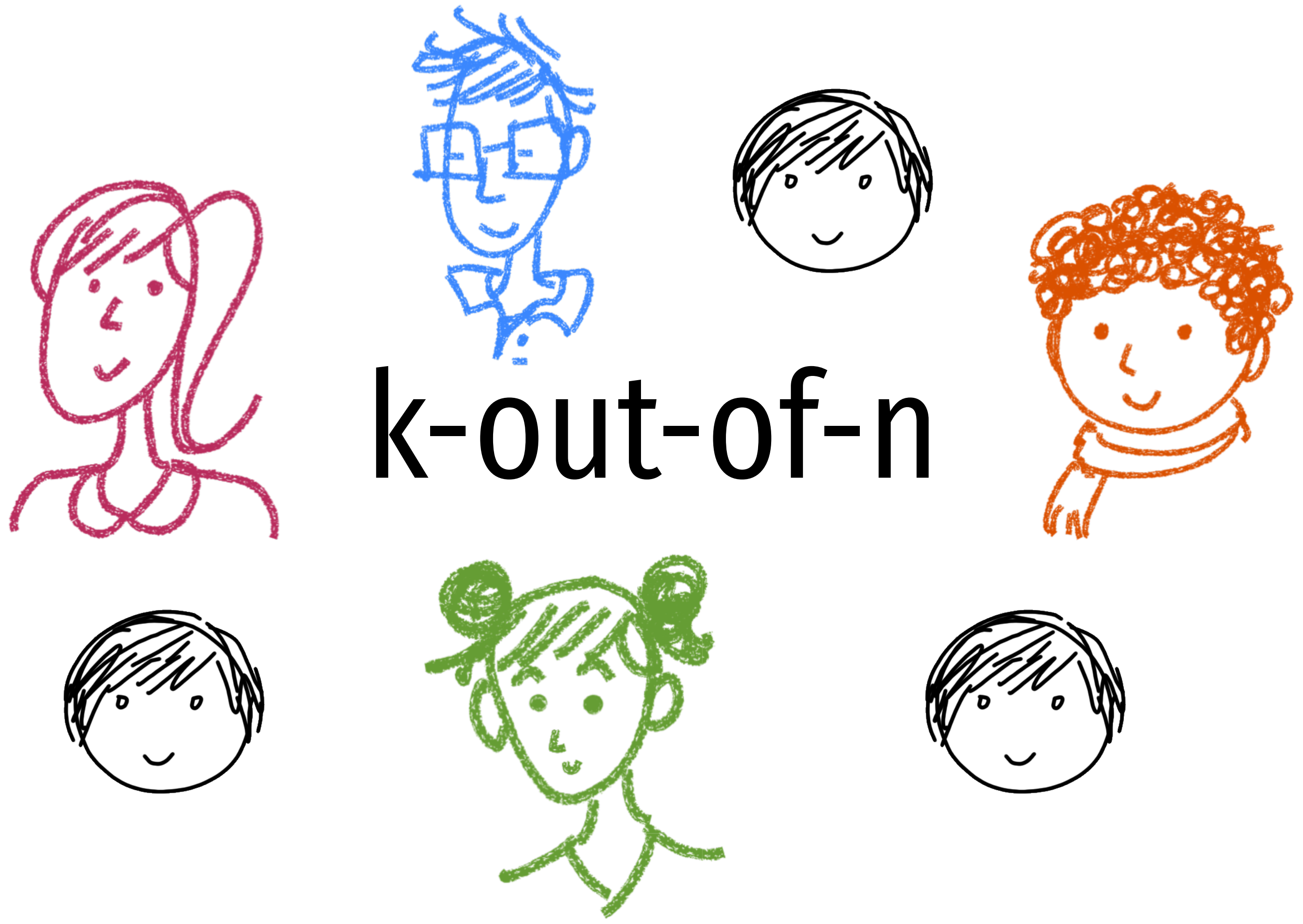
Everyone has a new key for pk.



Key refresh is easy [KMOS21]



Beaver trick to refresh
pairwise OTE with XOR.
PRG Seed.



Re-run DKG.
Re-run OTE.
Works well because our
key setup is fast.

TLDR: Key setup times are CRITICAL

Paillier and other schemes require a heavy key setup which makes refresh heavy.

Reason: (CAN'T Rerandomize Paillier N)

LAN/WAN k-out-of-k 2019

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 slowdown due to round complexity.

3

LAN/WAN k-out-of-k 2019

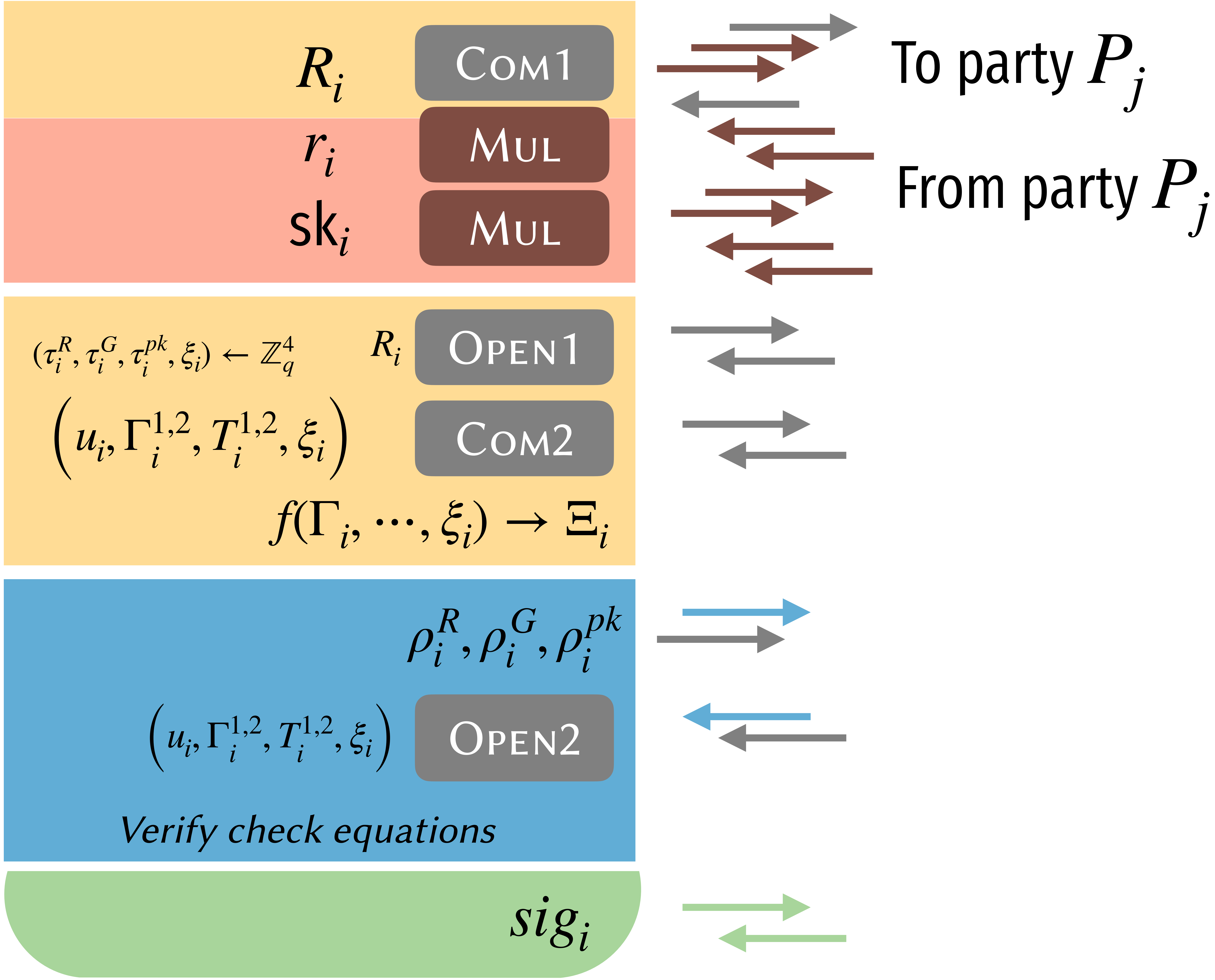
Milliseconds

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

$\log(t)+6$

WAN slowdown due to round complexity.

Updated protocol



5 Rounds
No ZK proofs
No hidden fees

Mostly Symmetric operations for *signing*. (Check eqns use 13 ec ops)

OT Extensions

Roy shows a break in KOS for special cases of $\mathbb{F}_{2^k}$

The break does not apply to $k = 128$, but it identifies a gap in the proof.

Our implementation is moving to SoftSpoken OT.

Concurrency issue in implementation.
If one instance aborts, all should abort. Fixed. [Riva]

Gaps between Theory and Practice

Random Oracle Model

Interparty Communication

[UX] Initializing the session, argument checking

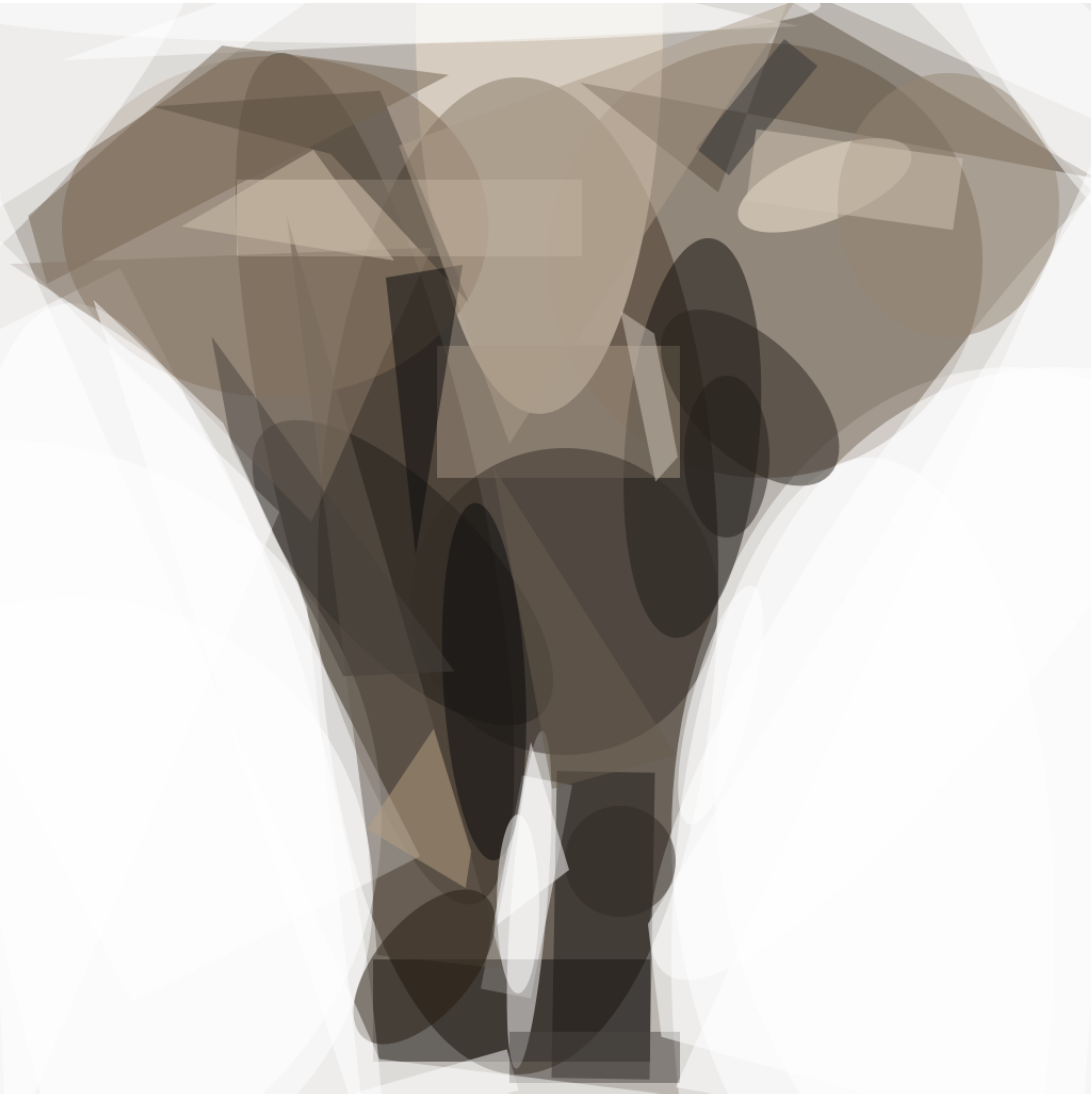
Use of Fiat-Shamir

If the protocol needs a programmable Random Oracle, every (sub)protocol instance needs a different RO.

One way is to hash a unique prefix.
Several recent bug bounties on this issue.

Our ’17 academic implementation spent 574 lines synchronizing fresh RO tags.

Encouraged me to learn TLA+ spec.
Found a simpler way <100l.



The elephant in the room is straight-line extractability.

A protocol that uses ZK proofs in a concurrent setting needs to extract witnesses without rewinding.

For standard security notions.

A protocol that uses ZK proofs in a concurrent setting needs to extract witnesses without rewinding.

For standard security notions.

Fiat-Shamir requires rewinding to extract a witness.

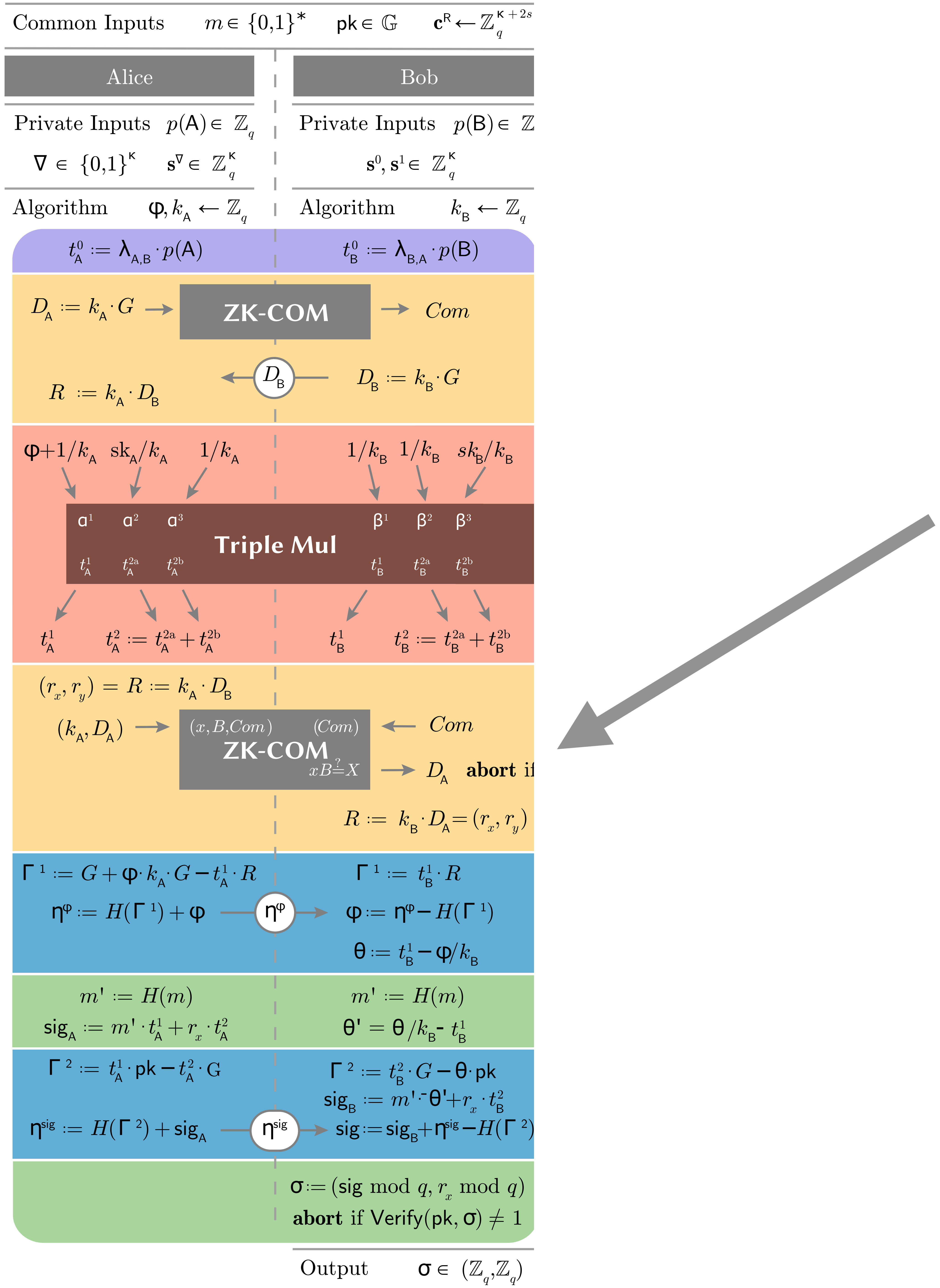
The best approach is straight-line extractability.
Pass03, Fischlin05, Kondi-shelat 21

Requires 10 copies of proof, extra prover time, verifier time.

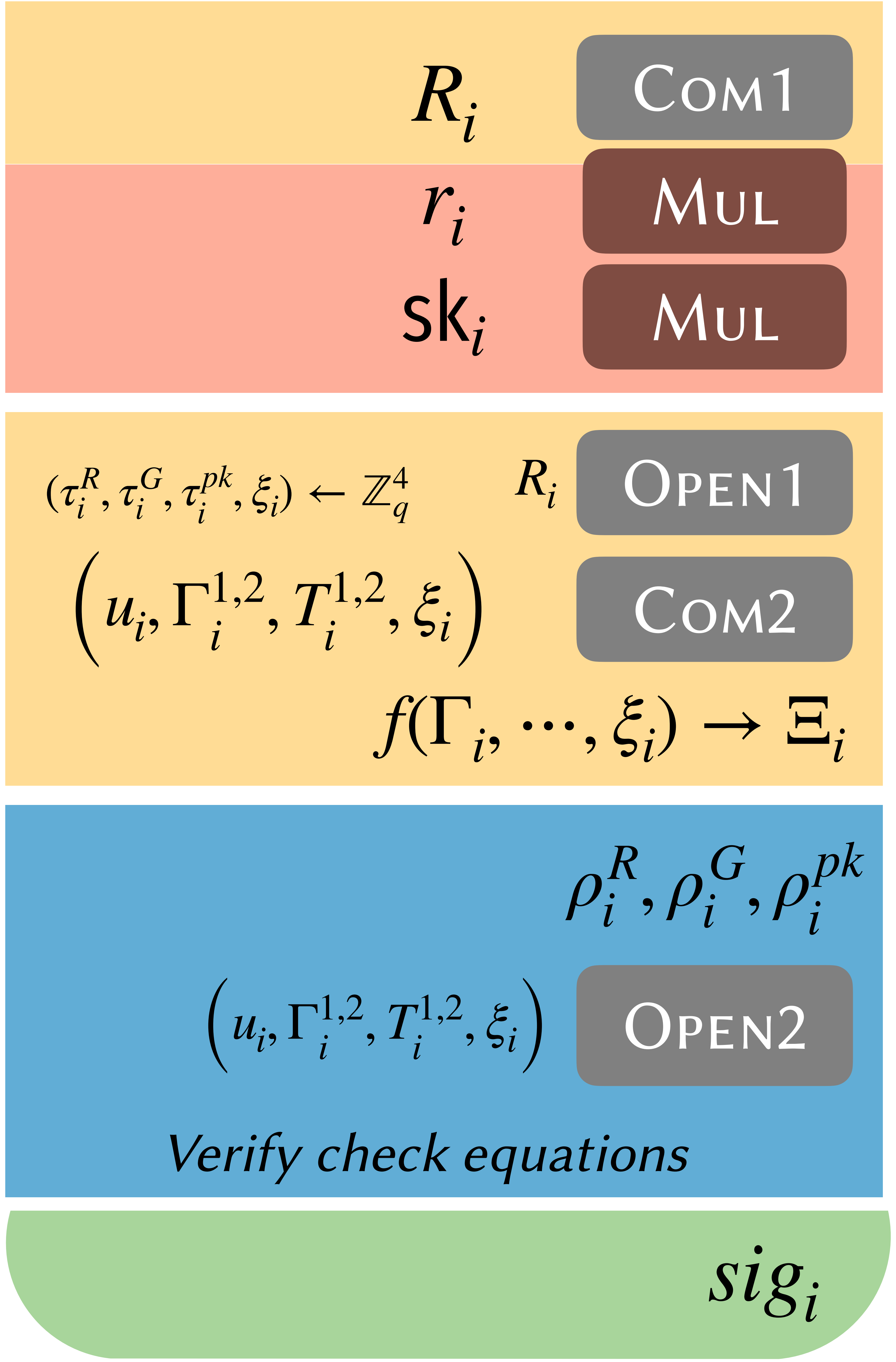
Concurrent setting means web3.

Or web2.
Or Internet.

But not at home...



2-out-of-n protocol uses 1 Schnorr proof.



k-out-of-n does not use ZK proofs. Avoids this overhead.

Paillier needs proofs to sign

are detected, $\mathcal{P}_i$ sets $R = \Gamma^{\delta^{-1}}$ and stores (R, k_i, χ_i). For malicious security, the aforementioned process is augmented with the following ZK-proofs:

(a) The plaintext of K_i lies in range $\mathcal{I}_\varepsilon$.

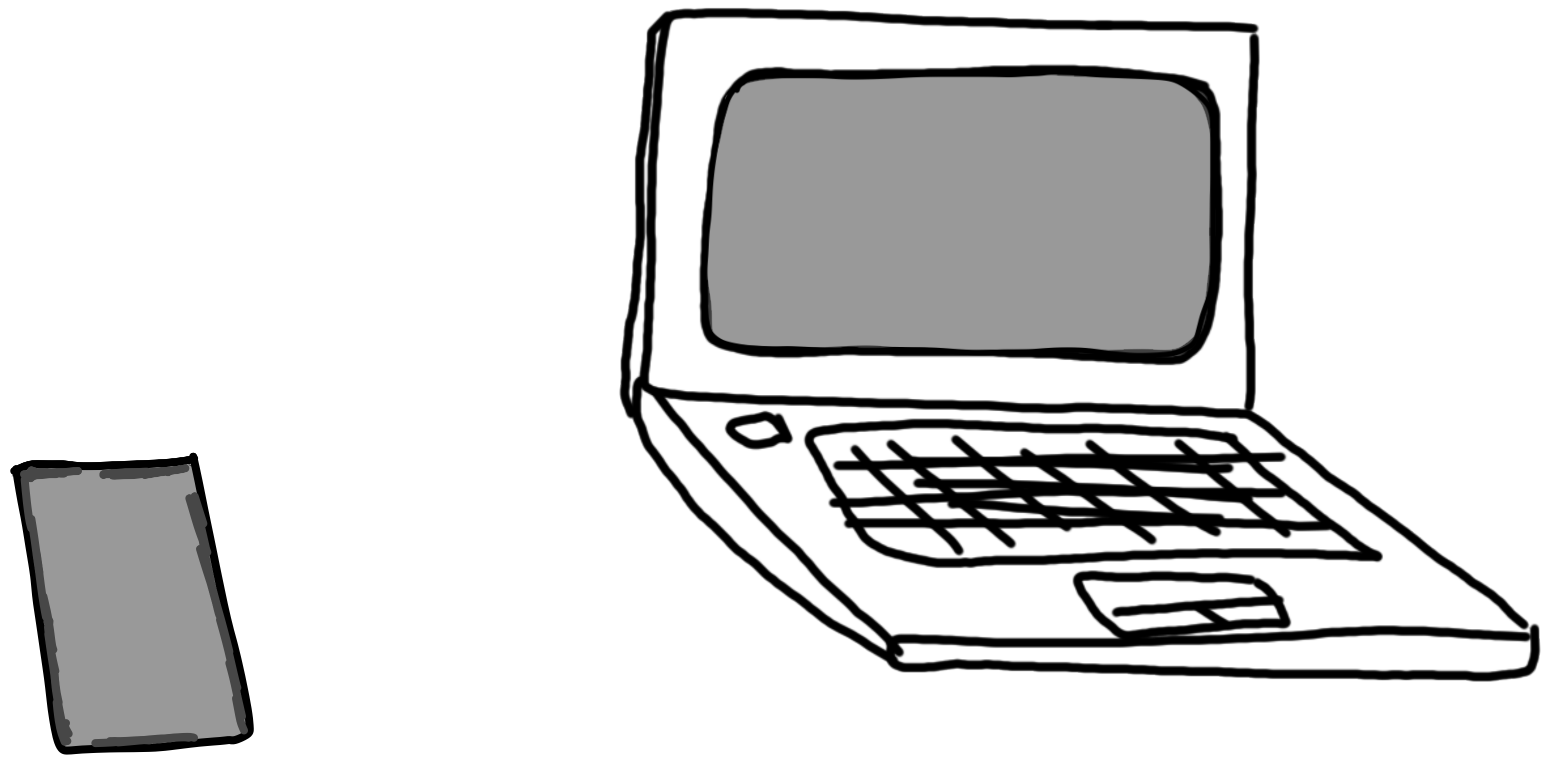
(b) The ciphertext $D_{j,i}$ was obtained as an affine-like opperation on K_j where the multiplicative coefficient is equal to the exponent of Γ_i, and it lies in range $\mathcal{I}_\varepsilon$, and the additive coefficient is equal to hidden value of $F_{j,i}$, and lies in range $\mathcal{J}_\varepsilon$.

(c) The ciphertext $\hat{D}_{j,i}$ was obtained as an affine operation on K_j where the multiplicative coefficient is equal to the exponent of X_i, and it lies in range $\mathcal{I}_\varepsilon$, and the additive coefficient is equal to hidden value of $\hat{F}_{j,i}$, and it lies in range $\mathcal{J}_\varepsilon$.

(d) The exponent of Γ_i is equal to the plaintext-value of G_i.

CGGMP

How to avoid straight-line extraction penalty?



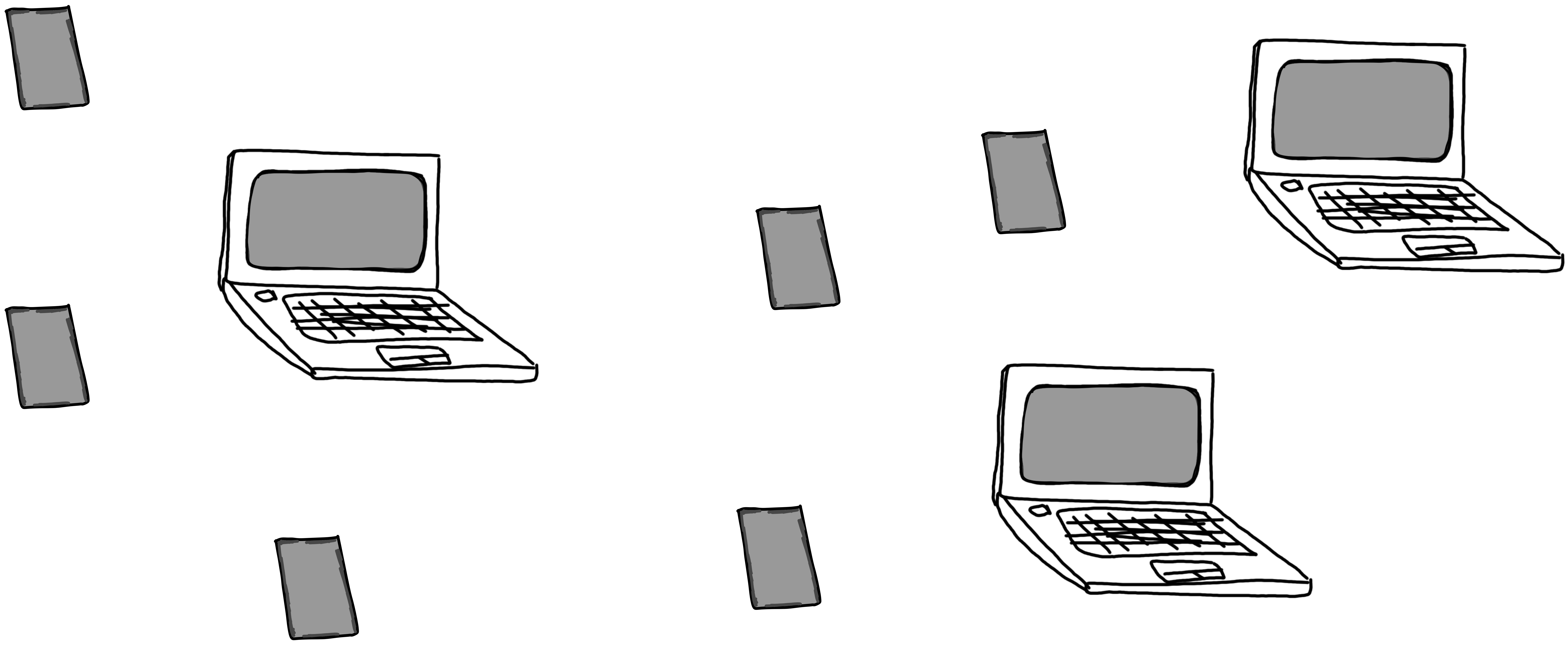
Protocol is run on devices owned by the same entity.

Enforces each device **serializes** its executions.

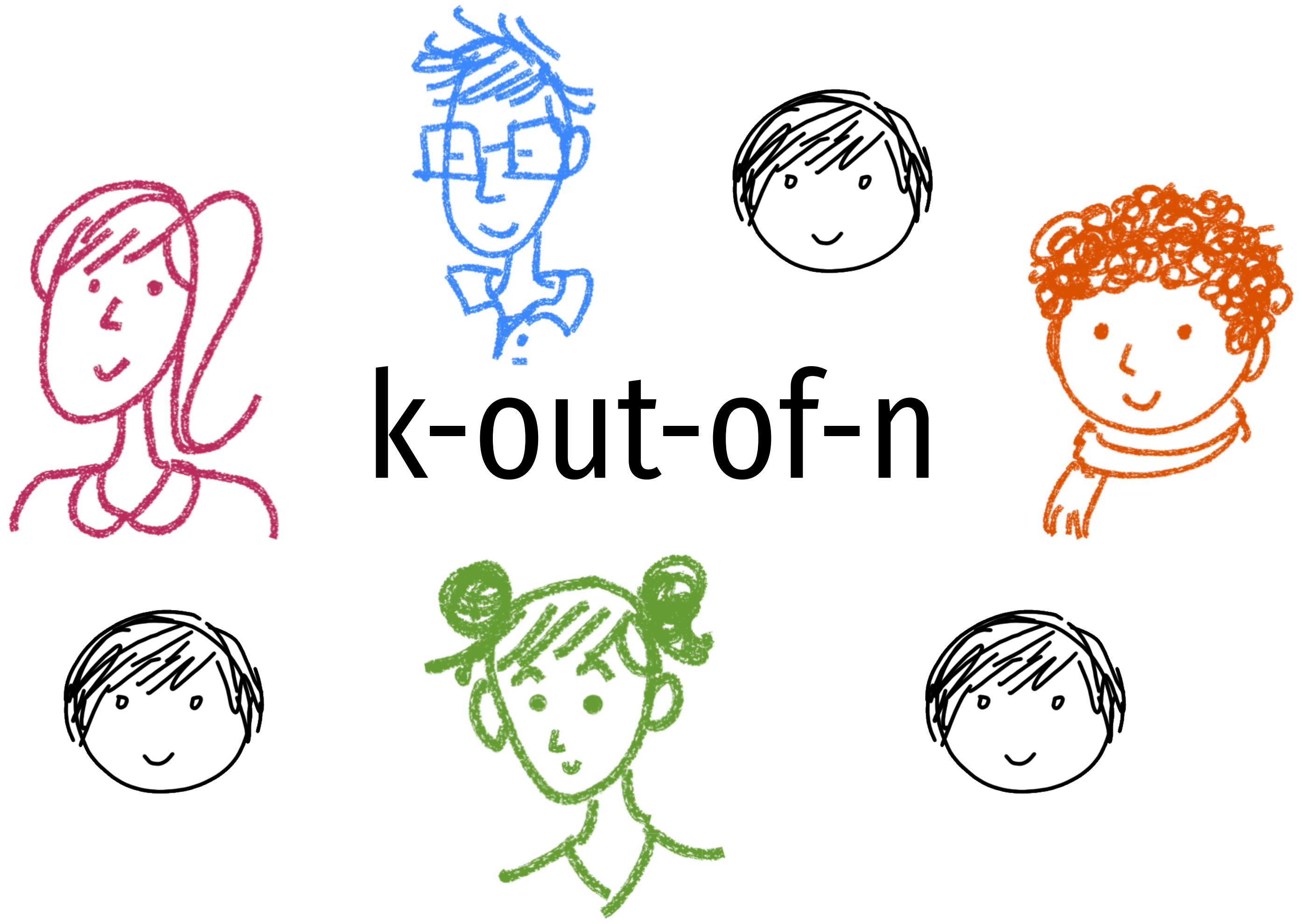
Does not work if one entity is a server (common Bob for many clients).

6

The really really difficult issues



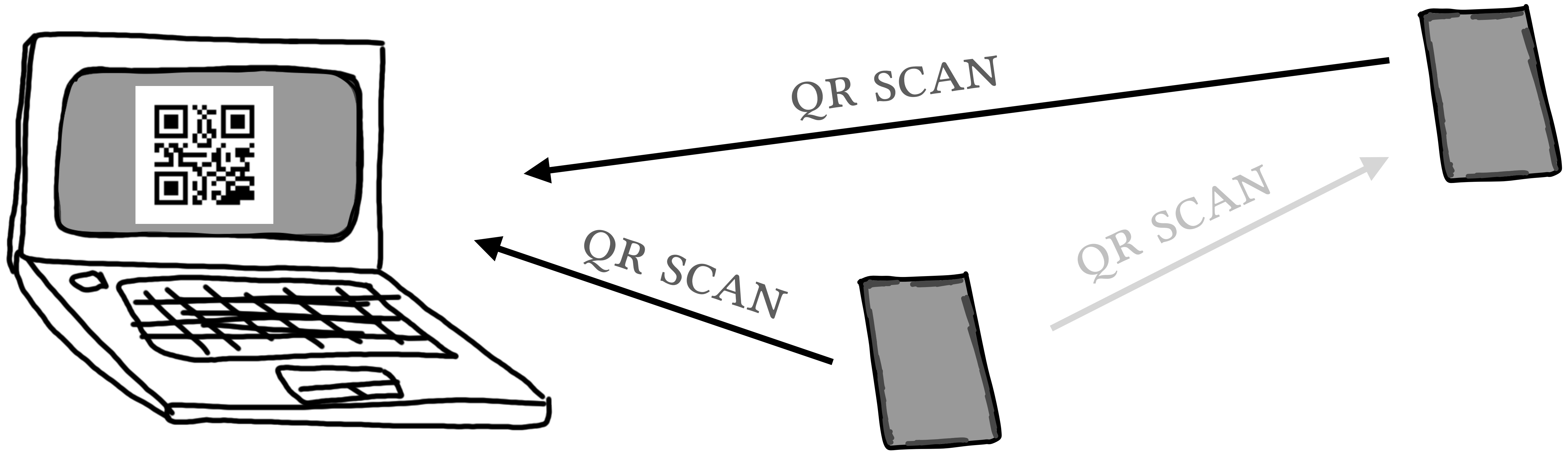
The environment Z



Starting assumptions are hard:
common knowledge of participants, msg, session id, authenticated channels.

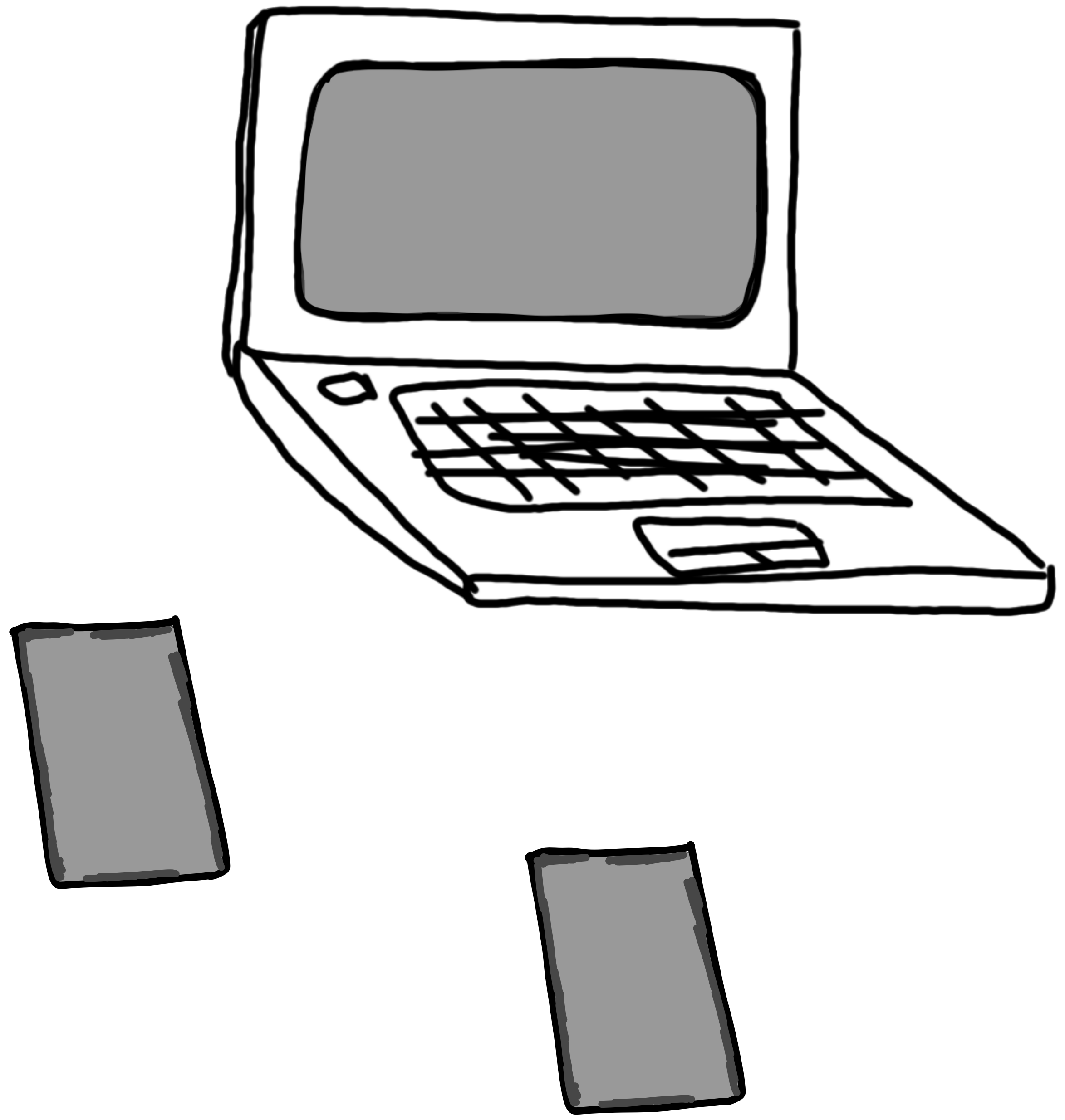
```
./target/release/main -m 13 -i 500 --parties nodekey:d9e492c62214380c7206f15f8c2efd55c9c606c44d30b6b444fd5f6926b7477e@w0.u
s-central1-a.c.neuwork.internal:6000,nodekey:d16878d51772537c363b38d4870a8e964561593ae8d6a32d94949e41e2d2c45a@w1.us-east4-
a.c.neuwork.internal:6001,nodekey:7c70d391bf3ed1c655f4f4b910a191e8445211c95e1a0385dfac1052755bd324@w2.us-west1-a.c.neuwork
.internal:6002,nodekey:744dcf572fb5589829a1e73e882888264e722a1fdb5d4e17c5e29ccc8f9a6d79@w3.europe-west2-c.c.neuwork.intern
al:6003,nodekey:c41976d17843eb582ed9bec167c1262eabffec66de2ea9e82807ff91bd30c166@w4.us-east1-b.c.neuwork.internal:6004,nod
ekey:7a5abd6d4bbcde5158618f63d697e5cb0d78ce3be51afbf111362a150c101e47@w5.us-west3-a.c.neuwork.internal:6005,nodekey:7c0740
65f485b44926cbaf2c48a237174906cce31f7b1cd3169d0a39fd44564f@w6.us-central1-b.c.neuwork.internal:6006,nodekey:00127a6f430934
2b564b05342e225777760477dc6c30cadb60780e1b789f0542@w7.us-east4-b.c.neuwork.internal:6007,nodekey:b8e1c4a5b8daa83c1ce6d1571
5510e4598f6547f75c136b9016b292e4967f731@w8.europe-west2-b.c.neuwork.internal:6008,nodekey:e590ed197aa4032d0fea554c7c71c95d
bc4adf2dd3c2d23d8fa8cf44ac66d357@w9.us-west1-b.c.neuwork.internal:6009,nodekey:0474a1f2f55f426d3e1af7f62633617afbfe84ed784
a6add5d205b1940ec3315@w10.europe-west1-b.c.neuwork.internal:6010,nodekey:a05a402e2beaddd8762eeb4241c57e585c4e390d73c5828d2
aaee9009ab8c12a@w11.us-central1-c.c.neuwork.internal:6011,nodekey:f64d499638737b2e8529377cd18927cd2f7bd1ac271315782742b73e
748e4f07@w12.us-east4-c.c.neuwork.internal:6012,nodekey:20d6f90cbd462e700f93ecaa58cb6d552571b0a79aeb4bd4c5de10c22fd6ee47@w
13.europe-west1-c.c.neuwork.internal:6013,2023/03/29 04:09:40 pk: nodekey:a8229b136331cdc27675213ac003569496527381016590cfa
```

Setup needs many scans

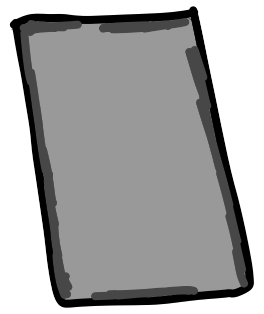


All devices are local, same owner, k-out-of-k at setup.

Growing participant set



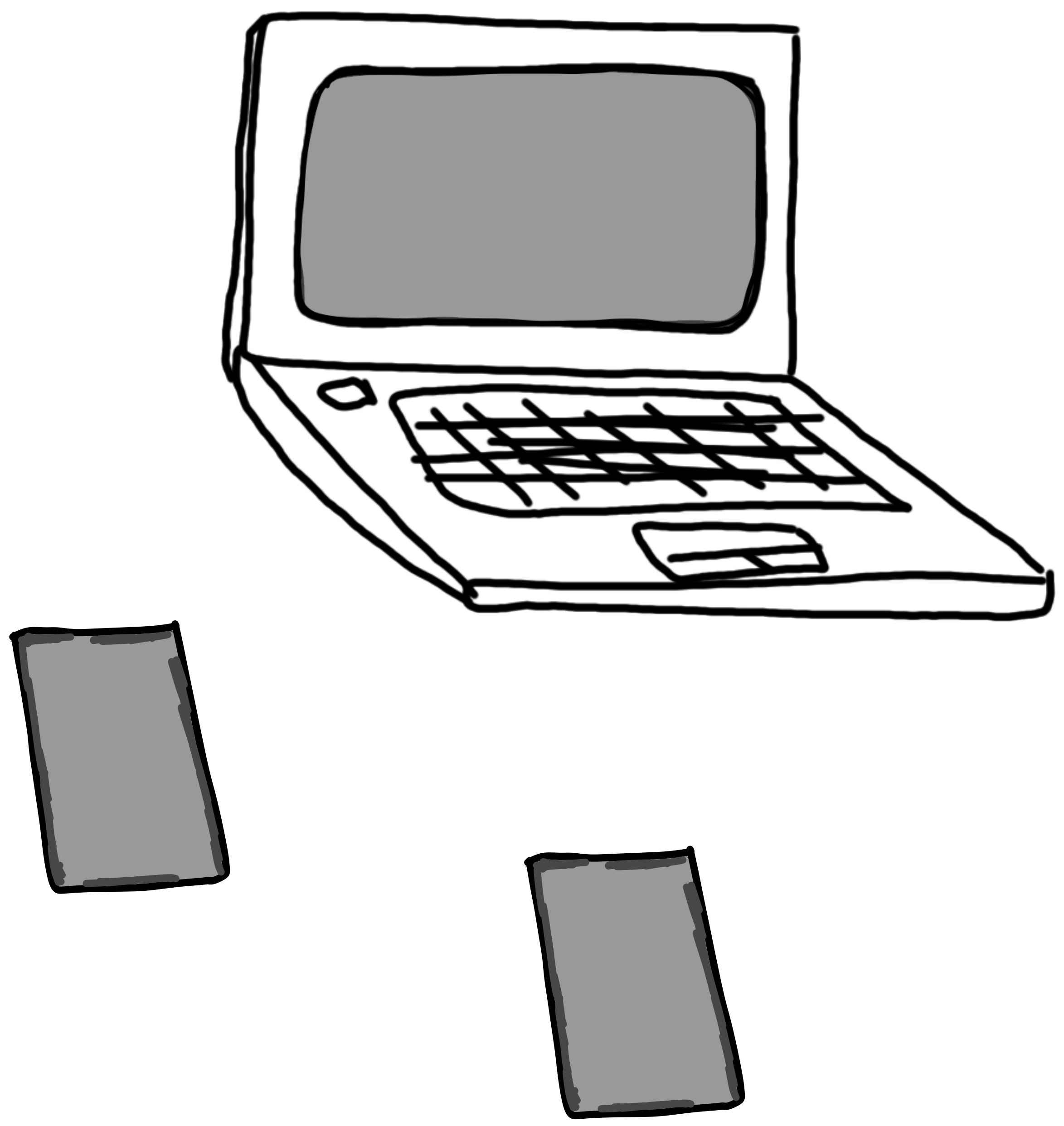
K-out-of-k already setup.

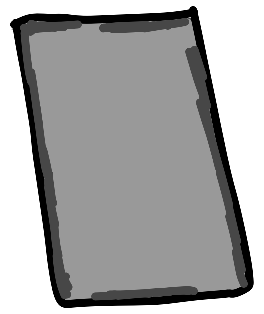


Want to setup k-out-of-(k+1).

Changing threshold

Handled using
key refresh
methods.

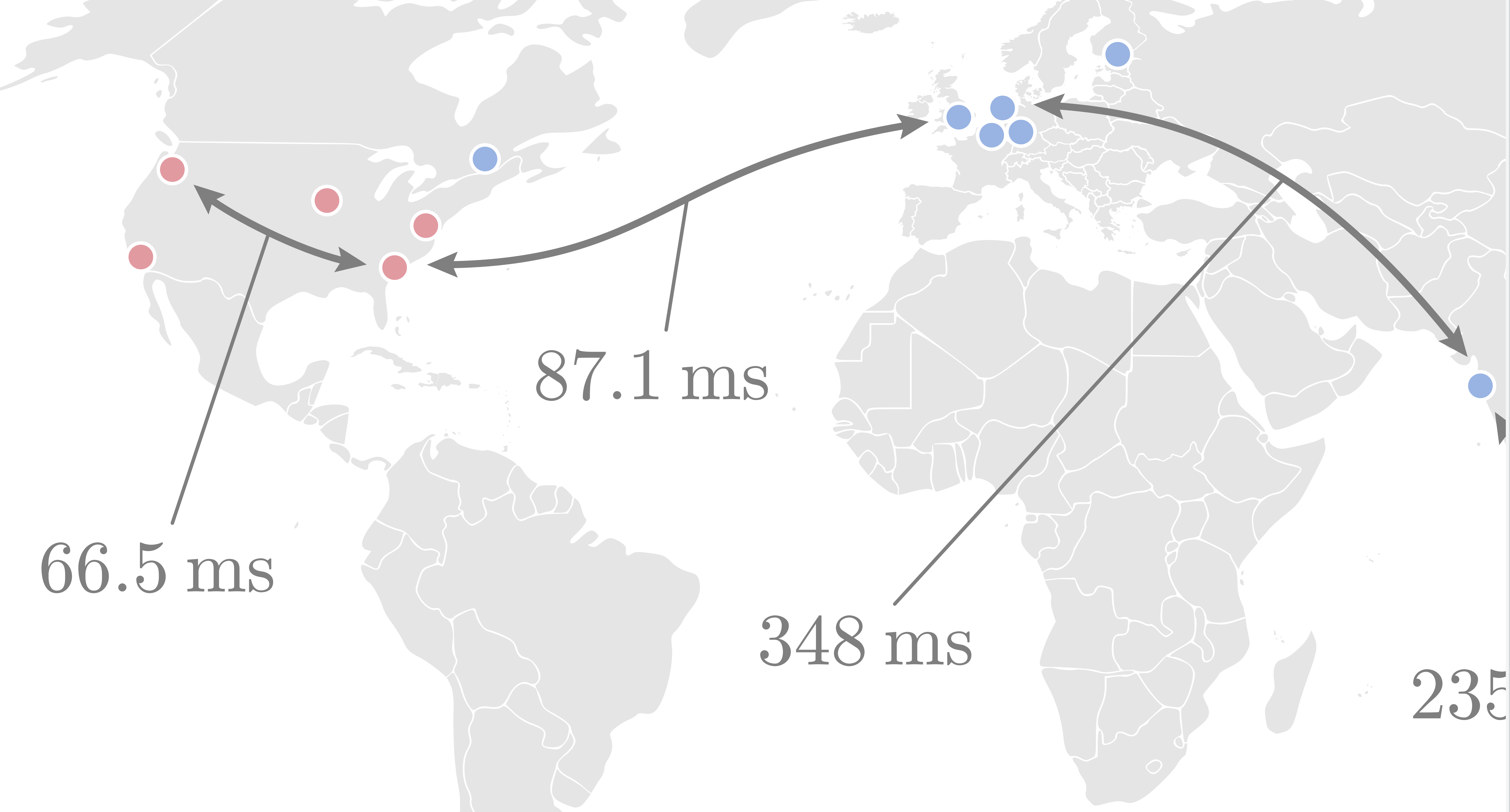


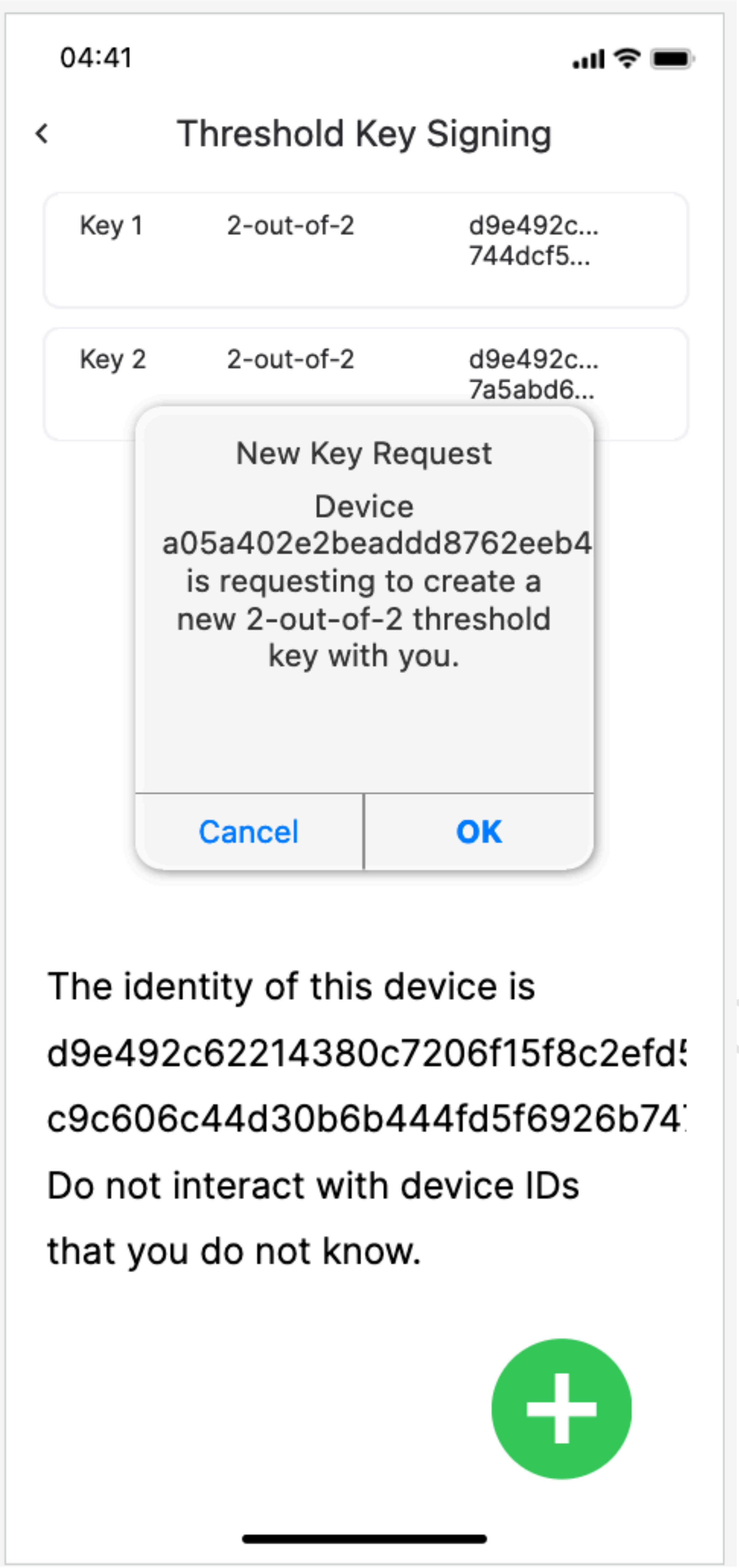


K-out-of-k already
setup.

Want to setup
(k+1)-out-of-(k+1).

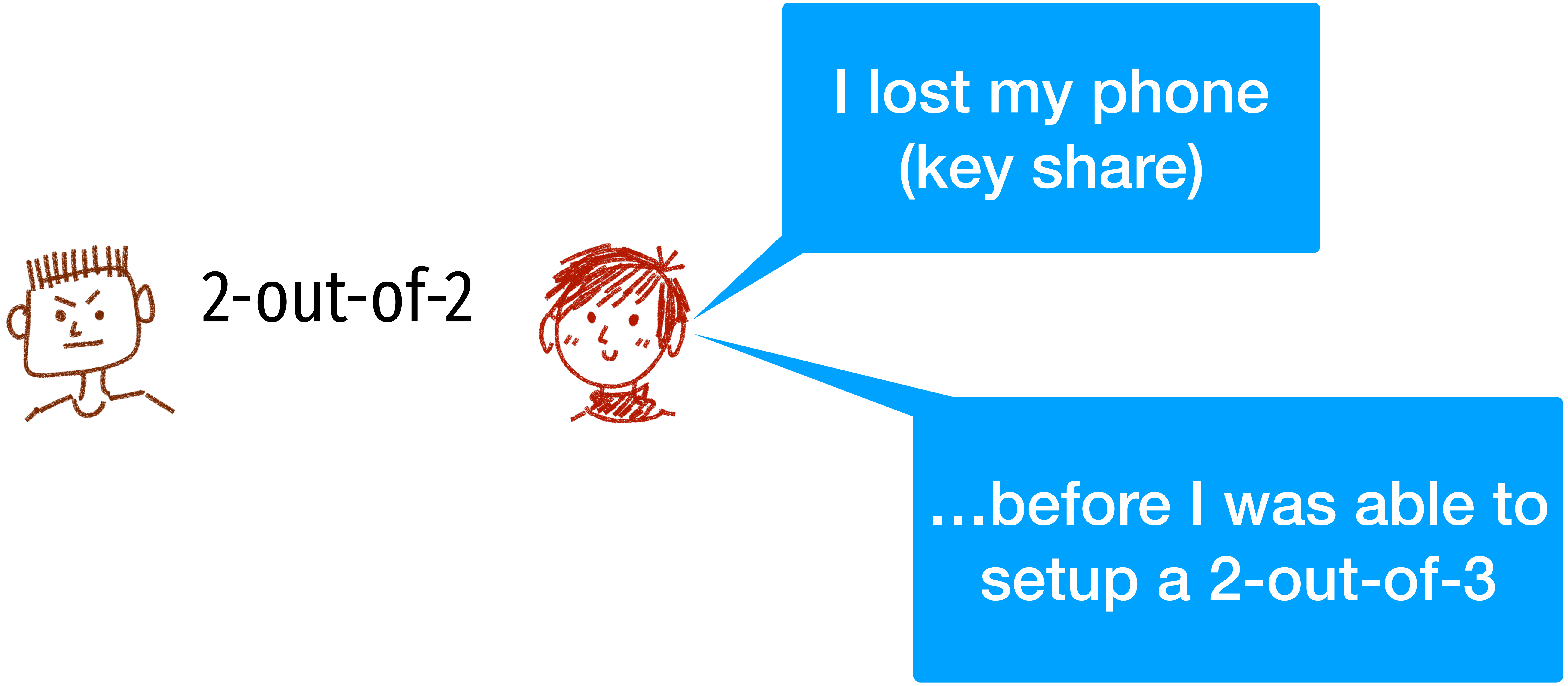
Parties are not local





common knowledge of participants, and session id,
authenticated channels.

Recovery


2-out-of-2
I lost my phone (key share)
...before I was able to setup a 2-out-of-3

Is threshold a 10x better experience for {user, organization}?

Appendix