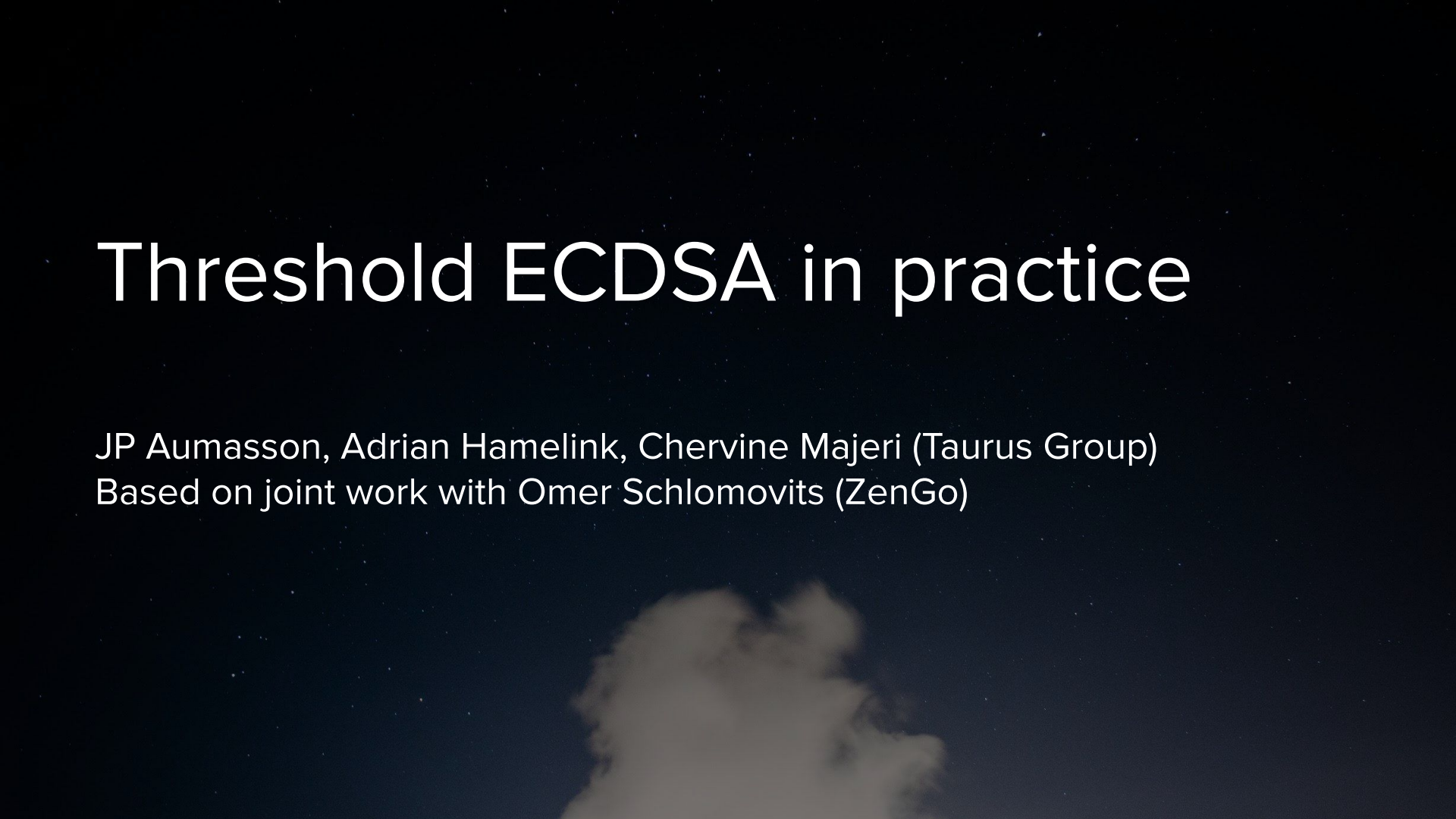
Threshold ECDSA in practice

JP Aumasson, Adrian Hamelink, Chervine Majeri (Taurus Group)
Based on joint work with Omer Schlomovits (ZenGo)



Agenda

Introduction: motivations and basic notions and formalism (JP)

ECDSA & MPC: viewing threshold ECDSA as an MPC functionality (Adrian)

Protocols: overview of the main protocols and their properties (Adrian)

Real-world: review of needs, security models, assumptions (JP/Chervine)

Conclusion: thoughts and open problems (JP)

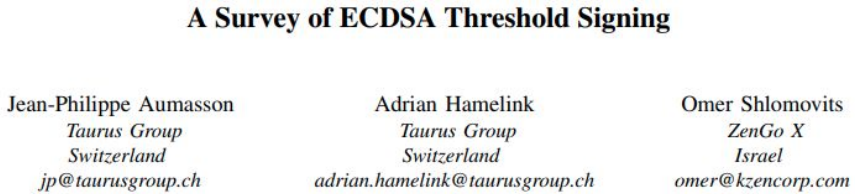
Introduction

Background & motivations

Research unit of **Taurus Group** - taurusgroup.ch

- Digital asset infrastructure for financial organizations
- Need security, reliability, maintainability, and usability
- Protocol using an **HSM**'s TEE for signing, quorum validation, security controls

How to leverage threshold crypto to offer an HSM-free alternative?
What are "real-world" security and performance needs?
What protocol and libraries are acceptable?



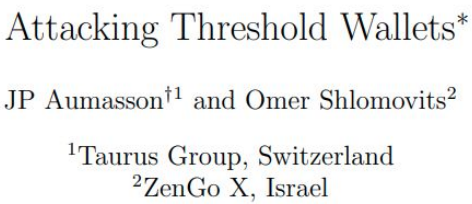
A Survey of ECDSA Threshold Signing

Jean-Philippe Aumasson
Taurus Group
Switzerland
jp@taurusgroup.ch

Adrian Hamelink
Taurus Group
Switzerland
adrian.hamelink@taurusgroup.ch

Omer Shlomovits
ZenGo X
Israel
omer@kzencorp.com

eprint.iacr.org/2020/1390



Attacking Threshold Wallets*

JP Aumasson[†1] and Omer Shlomovits[2]

[1]Taurus Group, Switzerland
[2]ZenGo X, Israel

eprint.iacr.org/2020/1052

Threshold signing motivations

Distributing trust, avoid a SPoF, enable “4-eye control”

Off-chain quorum mechanism, coin-agnostic, **hiding governance patterns**

Multi-signatures require multiple keys, work differently for different platforms

Alternative / complement to HSMs/TEEs, but address different needs

Efficient schemes for RSA, Schnorr sigs (including Ed25519), **ECDSA is harder**

Threshold signing?

2 protocols, although keygen can be centralized when it makes sense:

(**t**, **n**) parameters where **t**+1 shares are necessary and sufficient to sign

Distributed key generation (DKG): **n** parties obtain shares of a private key

Signing: **t+1** parties use their share to collectively sign a given message

ECDSA & MPC

Classic ECDSA

$sk \in Z_q$, $pk = sk \bullet G$, message m, $H(m) \in Z_q$

$Sign_{sk}(m)$	**$Verify_{pk}((r,s), m)$**
• Sample random k in Z^*_q • $R = k \bullet G = (r_x, r_y)$, $r = r_x \pmod q$ • $s = k^{-1} (H(m) + r \bullet sk) \pmod q$ • Output (r, s)	• $R' = [H(m)s^{-1}] \bullet G + [rs^{-1}] \bullet pk = (r'_x, r'_y)$ • Check $r'_x \pmod q = r$

Classic ECDSA to threshold ECDSA ?

$sk \in Z_q$, $pk = sk \cdot G$, message m, $H(m) \in Z_q$

Threshold ⇒ use secret sharing for sk and k

$Sign_{sk}(m)$	**$Verify_{pk}((r,s), m)$**
• Sample random k in Z^*_q • $R = k \cdot G = (r_x, r_y)$, $r = r_x \pmod q$ • $s = k^{-1} (H(m) + r \cdot sk) \pmod q$ • Output (r, s)	• $R' = [H(m)s^{-1}] \cdot G + [rs^{-1}] \cdot pk = (r'_x, r'_y)$ • Check $r'_x \pmod q = r$

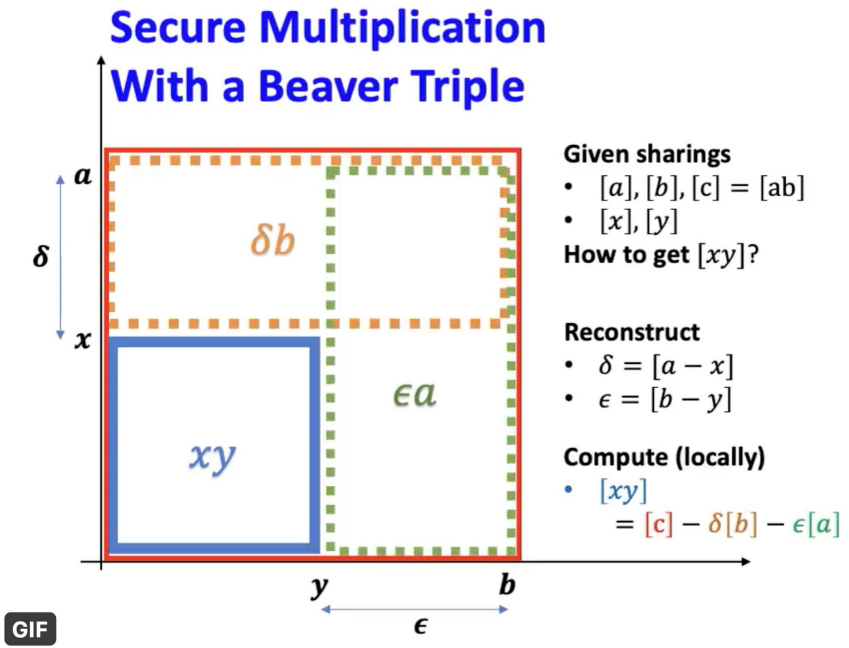
Threshold ECDSA as an MPC functionality

Secrets are now *shared*, i.e. **sk** → [**sk**] with **sk** = $sk_1 + \cdots + sk_n$, [**pk**] = [**sk**] • G

Formulate $Sign_{[sk]}$ using Arithmetic Black Box (ABB) functionality:

[r] ← Rand()	$r \in Z_q$
[c], [a], [b] ← RandMul()	$c = ab$
x ← Open([x])	$x \in Z_q$
[c] ← Mul([a], [b])	$c = ab \in Z_q$
[z] ← a • [x] + b • [y]	$z = ax + by \in Z_q$
[X] ← [x] • G	$x \in Z_q, X = x \bullet G \in \langle G \rangle$

ABB Example: Multiplication



[z] ← Mult([x], [y]):

- [c], [a], [b] ← RandMul()
- δ ← Open([a] - [x])
- ε ← Open([b] - [y])
- [z] ← [c] - δ[b] - ε[a]

Credit: A. Yanai https://twitter.com/i/status/1324424768649601024

ABB Example: Inversion

[y] ← Invert([x]):

- [b] ← Rand()
- [z] ← Mul([x], [b])
- z ← Open([z]) $z = x\ b$
- [y] ← z^{-1} [b] $y = (x\ b)^{-1}\ b = x^{-1}$

Threshold ECDSA as an MPC functionality

PreSign	Sign
- $[k] \leftarrow \text{Rand}()$ - $[b] \leftarrow \text{Rand}()$ - $[e] \leftarrow \text{Mul}([b],[k])$ $e = b\,k$ - $[f] \leftarrow \text{Mul}([b],[sk])$ $f = b\,sk$ - $e \leftarrow \text{Open}([e])$ $e = b\,k$ - $[t] \leftarrow e^{-1} \cdot [f]$ $t = (bk)^{-1} f = k^{-1} sk$ - $[k^{-1}] \leftarrow e^{-1} \cdot [b]$ k^{-1} - $R \leftarrow \text{Open}([k] \cdot G)$ $R = k \cdot G = (r_x, r_y)$ Store $(r_x, [k^{-1}], [t])$	- Retrieve $(r_x, [k^{-1}], [t])$ - $[s] \leftarrow H(M) \cdot [k^{-1}] + r_x \cdot [t]$ - $s \leftarrow \text{Open}([s])$ $s = k^{-1} \cdot H(M) + k^{-1} \cdot r_x \cdot sk$ $k^{-1}(H(M) + r_x \cdot sk)$ Output (r_x, s)

ECDSA without generic MPC?

Without MPC, we must be careful with **privacy** and **correctness**

Blinding factors helps hide secrets when revealed

Prove correctness by

- Proving computations in **zero-knowledge**
- **Commit** to values before publishing
- **Verify** algebraic relations

Shared secret multiplication

How to compute $[z] = [x] \cdot [y]$?

$$z = x \cdot y = (x_1 + \cdots + x_n) \cdot (y_1 + \cdots + y_n) = \sum_{i,j} x_i \cdot y_j \overset{?}{=} \sum_i z_i$$

Multiplicative-to-Additive conversion (MtA) protocol:

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

$$z_i = x_i \cdot y_i + \sum_{i \neq j} a_{i,j} + b_{i,j}$$

Multiplicative-to-Additive share conversion

Using Homomorphic encryption

Alice: x		**Bob**: y
$c \leftarrow Enc_A(x)$	$c \rightarrow$	$b \leftarrow\$ Z_q$
	$\leftarrow c'$	$c' \leftarrow (c \odot Enc_A(y)) \oplus Enc_A(-b)$
$a \leftarrow Dec_A(c')$		$c' = Enc_A(x \cdot y - b)$

$$a + b = (x \cdot y - b) + b = x \cdot y$$

Security concerns

Same **unforgeability** property wanted

Threshold optimal ⇒ t = n - 1 ⇒ dishonest majority ⇒ no **robustness**

Parties must **abort** when checks go wrong

Identification helps eject misbehaving parties

Type of adversary (active vs. passive)

Universal Composability (UC) proof

Protocols

2018, first *efficient* constructions

Fast Multiparty Threshold ECDSA with Fast Trustless Setup

Rosario Gennaro[1] and Steven Goldfeder[2]

[1] City University of New York
rosario@cs.ccny.cuny.edu
[2] Princeton University[§]
goldfeder@cornell.edu

Paillier as homomorphic scheme

Explicitly verify all computations

Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody*

Yehuda Lindell[†] Ariel Nof[†] Samuel Ranellucci[‡]

October 14, 2018

UC security

Paillier or OT based MtA

Verify result “in-the-exponent” with ElGamal commitments

Gennaro and Goldfeder ‘20

- $[k], [b] \leftarrow \text{Rand}()$, $[B] \leftarrow [b] \cdot G$, commit to B_i
- $[v] \leftarrow \text{Mult}([k], [b])$, $[t] \leftarrow \text{Mult}([k], [sk])$, check with pk_i
- $v \leftarrow \text{Open}([v])$, $v = k\,b$
- $B \leftarrow \text{Open}([B])$, verify decommit
- $R \leftarrow v^{-1} \cdot B = (k^{-1}\, b^{-1})\,(b \cdot G) = k^{-1} \cdot G$, $[V] \leftarrow [k] \cdot B$, prove using MtA msgs
- $V \leftarrow \text{Open}([V])$, verify $V = v \cdot G$
- $[s] \leftarrow m \cdot [k] + r \cdot [t]$,
- $s \leftarrow \text{Open}([s])$

2019-2020,design *trade-offs*

Threshold ECDSA from ECDSA Assumptions:
The Multiparty Case

Jack Doerner
j@ckdoerner.net
Northeastern University

Yashvanth Kondi
ykondi@ccs.neu.edu
Northeastern University

Eysa Lee
eysa@ccs.neu.edu
Northeastern University

abhi shelat
abhi@neu.edu
Northeastern University

May 22, 2020

OT based Multiplication

Logarithmic rounds *but*
constant number of operations

Bandwidth-efficient threshold EC-DSA

Guilhem Castagnos[1], Dario Catalano[2], Fabien Laguillaumie[3],
Federico Savasta[2,4], and Ida Tucker[3]

[1] Université de Bordeaux, INRIA, CNRS, IMB UMR 5251, F-33405 Talence, France.
[2] Università di Catania, Italy.
[3] Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France.
[4] Scuola Superiore di Catania, Italy

GG18

~~Paillier~~ → Class group MtA

2020, more security *features*

One Round Threshold ECDSA with Identifiable Abort

Rosario Gennaro
The City University of New York
rosario@ccny.cuny.edu

Steven Goldfeder
Cornell Tech/Offchain Labs
goldfeder@cornell.edu

Detection and attribution of misbehavior via secure “aborts”

UC Non-Interactive, Proactive, Threshold ECDSA

Ran Canetti* Nikolaos Makriyannis† Udi Peled†

May 8, 2020

Use Paillier as commitments

Proactive key refresh

Only 4 rounds (3 offline)

Abort with identification

Detect misbehaving users:

- Failure to **verify ZK proof**
- Consistency of **R** using Enc($\mathbf{k}_i$) from MtA in Mul([k],[sk])
- Valid **decommitment**
- **Algebraically** [k]•R = G, and pk = [t]•R *where* $R = k^{-1} \bullet G$ *and* $t = k\ sk$
- Verify **signature**

Identification protocol needed for last two checks.

Proactive key refresh

If [0] is a secret sharing of 0 with shares 0_i for party i,
then [z] = [x] + [0] is a new secret sharing of [x] but with different shares.

In **Refresh+AuxInfo** protocol, each party distributes shares of $[0^{(i)}]$

$$[sk'] = [sk] + \sum_i [0^{(i)}]$$

Auxiliary parameters such as Paillier keys, and Pedersen are also regenerated.

Old shares are no longer valid.

GG ‘20 + CMP ‘20 = CGGMP ‘20

UC Security

Proactive key refresh in 3 rounds

Non interactive signing

Two protocols for presigning & identification

- 3 offline rounds + $O(n^2)$ cost for identification
- 6 offline rounds + $O(n)$ cost for identification

Protocols comparison (from CGGMP ‘20)

Signing Protocol	*Rounds*	*Group Ops*	*Ring Ops*	*Communication*	*Proactive*	*ID Abort*	*UC*
Gennaro and Goldfeder [26]	9	$10n$	$50n$	$10\kappa + 20N$ (7 KiB)	✗	✗	✗
Lindell et al. [37] (Paillier)†‡	8	$80n$	$50n$	$50\kappa + 20N$ (7.5 KiB)	✗	✗	✓
Lindell et al. [37] (OT)†	8	$80n$	0	50κ (190 KiB)	✗	✗	✓
Doerner et al. [23]	$\log(n) + 6$	5	0	$10 \cdot \kappa^2$ (90 KiB)	✗	✗	✓
Castagnos et al. [16]*	8	$15n$	0	$100 \cdot \kappa$ (4.5 KiB)	✗	✗	✗
This Work: *Interactive*§	4 *or* 7	$10n$	$90n$	$10\kappa + 50N$ (15 KiB)	✓	✓	✓
This Work: *Non-Int. Presign*§	3 *or* 6	$10n$	$90n$	$10\kappa + 50N$ (15 KiB)	✓	✓	✓
This Work: *Non-Int. Sign*	1	0	0	κ (256 bits)	✓	✓	✓

Real-world crypto

Use cases

Shares should be controlled by different entities

“Simple” cases:

- B2C company with 2-party wallet shared control
- Multiple organizations sharing control of a wallet

Less simple cases (operation segregation needs):

- Cold wallet of a single organization (few addresses, high latency ok)
- Hot wallet a single organization (many addresses, need low latency)
    - How to do BIP32/44 efficiently?
    - Depends on the pooling model (relation between addresses and owners)

Security models vs. reality

Corruption:

- **Static** more realistic most of the time, so adaptive security safer
- Rarely the ability to “corrupt” one part, “uncorrupt” it to “corrupt” another
- Either a system is to be trusted (for some time), or it’s not and then it’s forever

Protocol obedience:

- **Malicious** models a compromised system
- Honest-but-curious makes little sense

Majority:

- It depends: with 2-2 you need **dishonest majority**
- Might be cases with a large **n** and small **t** where honest majority makes sense

Software implementing TSS

2 main open-source libraries used in production:

Binance's https://github.com/binance-chain/tss-lib (Go)

Zengo's https://github.com/ZenGo-X/multi-party-ecdsa (Rust)

Reviewed in detail in our survey, we did paid security audits of both

Recent lib by **ING bank** https://github.com/ing-bank/threshold-signatures/
(With Omer we found and reported some bugs)

Some organizations have non-OSS code, partially relying on OSS (arithmetic, etc.)

Deployment constraints

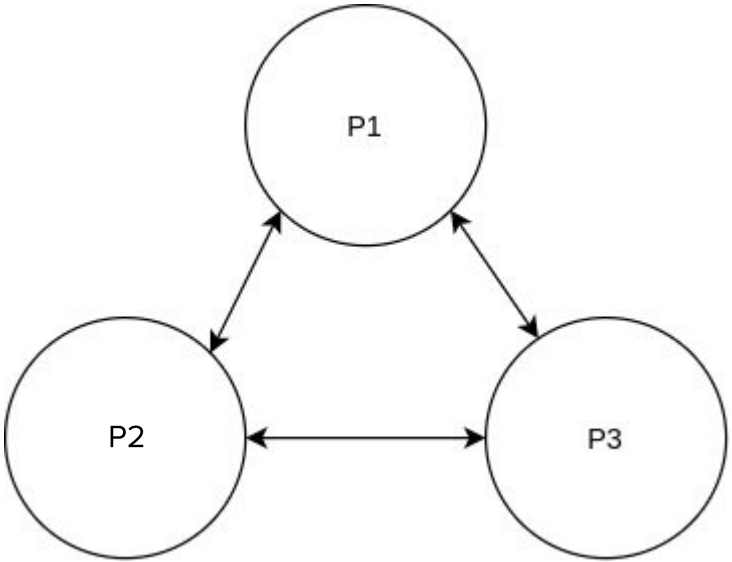
Most papers describing protocols assume that

- **Key distribution** is in place and safe (enabling secure channels)
  - Usually fine
- Communication is **lossless and in-order** (thus reliable) and low-latency
  - Not always the case

- **Practical** aspects of deployment often missing
  - Protocol limitations (GG18/20) may impact deployment potential
  - Slower signatures means scalability at infrastructure-level is important

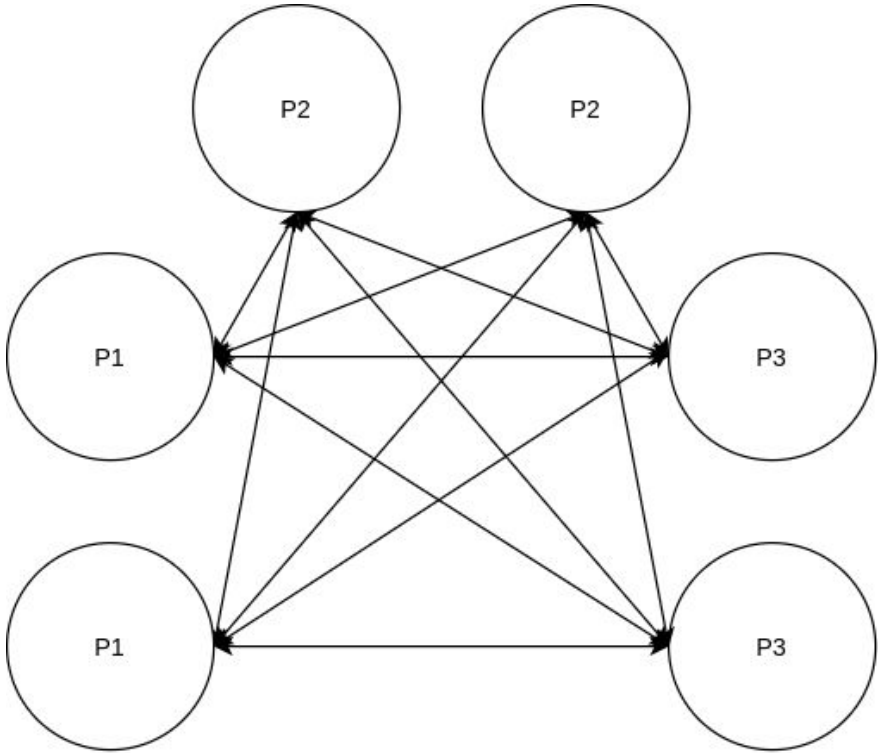
How would “less simple” deployment work in practice?

Naive deployment



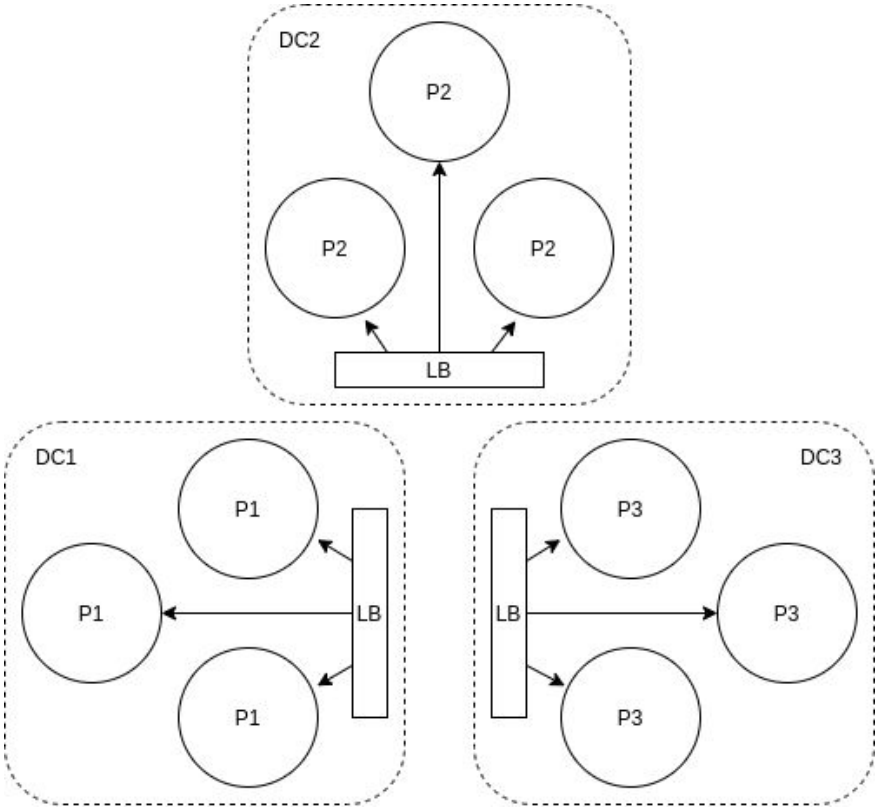
- Exactly mirrors crypto diagrams
- Doesn’t hold up in practice
  - No scalability
  - No failover
- Improvement: Support rolling out multiple instance of the same party.

Impractical deployment



- Multiple instances of each party with a mesh network
- Addresses scalability issues
- Mesh networks across different clouds is impractical/hard.

Robust deployment



- One party in each DC means segregation of duty is possible
- Access via load-balancer is standard
- DCs can scale their instances horizontally w/o any change to configuration
- Some engineering tricks to perform multiple rounds with the same participant for each party

Performance

Benchmark notes from a **robust deployment**:

- Keygen: 7-8s
  - 5s to generate safe primes
  - 1s for the other operations
  - WAN overhead 700ms

- Signature: 2-3s
  - 400ms for each of the two (optional*) MtAwc proof
  - 300ms for each of the two (optional*) MtAwc verifications
  - WAN overhead 900ms

Conclusions

Theory + practice = 🙌

Non-trivial protocols driven by practical needs

- Lindell and GG18 were game-changers
- CMP with aborts checking most of the boxes

Some protocols in production and battle-tested

Crypto addresses part of the problem, main real-world risks related to

- Implementation matching of described provensecure protocol
- Platform trust, software assurance, access control, back-ups, etc.

Open problems and challenges

NIST standardization ongoing https://csrc.nist.gov/Projects/threshold-cryptography

Formal verification of TSS-based protocols and software?

BTC supports Schnorr, BLS getting adoption; TSS ECDSA soon useless? :)

BIP32/HKD is not easy to threshold