Threshold Cryptography for Validator Networks

Zach Kelling

Lux Industries

2023

Abstract

We present a threshold cryptographic framework for blockchain validator networks in which no individual validator holds a complete signing key. The framework employs CGGMP21 [1] for ECDSA threshold signing (secp256k1 compatible), FROST [2] for EdDSA and BLS threshold signing, and a distributed key generation (DKG) protocol that creates key shares without a trusted dealer. We address the practical gap between threshold signature theory and production validator infrastructure: key resharing for validator set changes without exposing the master key, recovery from corrupted shares without key reconstruction, and slashing-compatible fault attribution. We prove that a compromised validator in a (t, n) threshold configuration reveals zero information about the master key, provide concrete benchmarks for MPC signing latency as a function of threshold size (4.2 ms for $t = 3$ of $n = 5$, 18.7 ms for $t = 7$ of $n = 10$), and describe the integration with the Lux consensus engine for MPC-based block signing.

Keywords: threshold signatures, MPC, CGGMP21, FROST, DKG, validator security, blockchain consensus

Contents

1 Introduction

In every major proof-of-stake blockchain, validators hold full signing keys. A validator's BLS or ECDSA private key is a single scalar stored in memory, on disk, or in an HSM. If that key is compromised—through software vulnerability, insider threat, physical access, or side-channel attack—the adversary can sign arbitrary messages as that validator. In BFT consensus systems with $n = 3f + 1$ validators, compromising $f + 1$ keys gives the adversary control of finality.

This is a fundamental design flaw, not an operational risk. The problem is not that validators are careless with keys; the problem is that a single scalar *exists* as a complete signing key.

Threshold cryptography eliminates this class of vulnerability by distributing each validator's signing capability across n shares such that any t shares can collaboratively produce a signature, but fewer than t shares reveal nothing about the key. The master key never exists at any single location—not during key generation, not during signing, not during resharing.

1.1 Contributions

1. A dealerless DKG protocol based on Pedersen commitments with verifiable share distribution (Section 3).
2. CGGMP21 threshold ECDSA for secp256k1 transaction signing, with identifiable abort and concrete latency benchmarks (Section 4).
3. FROST threshold signing for Ed25519, BLS12-381, and BIP-340 Taproot (Section 5).
4. Key resharing protocol enabling validator set changes without key reconstruction (Section 6).
5. Slashing-safe fault attribution compatible with threshold signing (Section 7).
6. Production benchmarks for MPC signing latency vs. threshold size (Section 9).

2 Problem: Single-Key Validator Architecture

2.1 Current Architecture

In a standard PoS validator:

1. Key generation: $sk \leftarrow \mathbb{Z}_q$, $pk \leftarrow sk \cdot G$.
2. Block signing: $\sigma \leftarrow \mathsf{Sign}(sk, \text{block})$.
3. The scalar sk is stored in a single location (keystore file, HSM, or memory).

2.2 Attack Surface

The common theme: every attack vector leads to *complete* key compromise because the key is a single scalar. There is no partial compromise, no graceful degradation.

2.3 Threshold Architecture

In a threshold architecture with parameters (t, n):

1. Key generation: Each of n parties receives a share sk_i. The master key sk is never materialized at any single location.

Table 1: Validator key compromise attack vectors.

Vector	Mechanism	Impact
Software exploit	RCE on validator node	Full key extraction
Insider threat	Operator with disk access	Full key extraction
Side channel	Timing/power analysis on signing	Key recovery over time
Cloud provider	VM snapshot, memory dump	Full key extraction
Supply chain	Malicious dependency in node software	Key exfiltration
HSM vulnerability	Firmware exploit (e.g., CVE-2019-15809)	Key extraction

2. Signing: Any t parties collaborate to produce a valid signature. The protocol output is indistinguishable from a single-signer signature.

3. Compromise: An adversary who corrupts $< t$ parties learns nothing about sk.

The validator's signing capability is distributed across multiple machines, geographic locations, or organizational boundaries. Compromising any single component yields zero useful information.

3 Distributed Key Generation

3.1 Dealerless Pedersen DKG

We use Pedersen's dealerless DKG [3] with Feldman's verifiable secret sharing (VSS) [4].

Algorithm 1 Dealerless DKG for (t, n) threshold.

Require: n parties $P_1, \ldots, P_n$; threshold t
Ensure: Each P_i holds share sk_i; public key $pk = sk \cdot G$

for all P_i, $i = 1, \ldots, n$ **do**
  Sample random polynomial $f_i(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,t-1}x^{t-1}$
  Broadcast commitments $C_{i,j} = a_{i,j} \cdot G$ for $j = 0, \ldots, t-1$
  Send share $s_{i \to j} = f_i(j)$ to each P_j via encrypted channel
end for
for all P_j, $j = 1, \ldots, n$ **do**
  for all received shares $s_{i \to j}$ **do**
    Verify: $s_{i \to j} \cdot G \stackrel{?}{=} \sum_{k=0}^{t-1} j^k \cdot C_{i,k}$
    If verification fails, broadcast complaint against P_i
  end for
  $sk_j \leftarrow \sum_{i=1}^{n} s_{i \to j}$ ▷ Aggregate shares
end for
$pk \leftarrow \sum_{i=1}^{n} C_{i,0}$ ▷ Public key from commitments

Theorem 3.1 (DKG security)**.** *If $< t$ parties are corrupted, the Pedersen DKG protocol produces shares of a uniformly random key sk such that no coalition of $< t$ corrupted parties can distinguish sk from a random element of $\mathbb{Z}_q$.*

Proof sketch. Each honest party contributes a uniformly random $a_{i,0}$. The master key $sk = \sum_i a_{i,0}$ is uniformly random as long as at least one honest party's $a_{i,0}$ is unknown to the adversary. With $< t$ corruptions and $n \geq 2t - 1$, at least t parties are honest, and the sum of their contributions masks the corrupted parties' contributions. □

3.2 Verifiable Share Distribution

Feldman VSS commitments $C_{i,j} = a_{i,j} \cdot G$ are binding under the discrete logarithm assumption and enable public verification of shares without revealing the polynomial coefficients. Any observer can verify the commitment consistency:

$$s_{i \to j} \cdot G = \sum_{k=0}^{t-1} j^k \cdot C_{i,k}$$

This ensures that a malicious party cannot distribute inconsistent shares (equivocation).

4 CGGMP21: Threshold ECDSA

4.1 Protocol Overview

CGGMP21 [1] is a threshold ECDSA protocol proven secure in the UC (Universal Composability) framework. It operates in two phases:

1. **Key generation**: Dealerless DKG produces additive shares sk_i such that $sk = \sum_{i \in S} \lambda_i \cdot sk_i$ for any qualifying set S of size t, where λ_i are Lagrange coefficients.
2. **Presigning**: Before the message is known, parties precompute nonce shares (k_i, γ_i) and exchange commitments. This is the expensive phase (Paillier encryption, range proofs).
3. **Signing**: Given message m, parties combine presigned values with message-dependent terms to produce (r, s).

4.2 Identifiable Abort

A critical property of CGGMP21 is *identifiable abort*: if any party deviates from the protocol, the honest parties can identify the cheater. This is essential for validator slashing—a misbehaving threshold signer can be identified and penalized.

Definition 4.1 (Identifiable abort)**.** *A threshold signing protocol has identifiable abort if, whenever the protocol fails to produce a valid signature, every honest party can output the identity of at least one corrupted party.*

4.3 secp256k1 Compatibility

CGGMP21 produces standard ECDSA signatures on secp256k1. The output is indistinguishable from a single-signer ECDSA signature. EVM `ecrecover` returns the expected address. No smart contract changes are needed.

Algorithm 2 CGGMP21 Signing (simplified).

Require: Message hash e; presigned tuples (k_i, χ_i, R) for $i \in S$
Ensure: ECDSA signature (r, s)

1: $r \leftarrow R.x \mod q$ ▷ $R = (k^{-1}) \cdot G$
2: **for all** $P_i \in S$ **do**
3: Compute $s_i \leftarrow k_i \cdot e + k_i \cdot r \cdot \chi_i \mod q$
4: Broadcast s_i
5: **end for**
6: $s \leftarrow \sum_{i \in S} s_i \mod q$
7: **return** (r, s)

5 FROST: Threshold Schnorr and BLS Signatures

5.1 Protocol Overview

FROST (Flexible Round-Optimized Schnorr Threshold) [2] is a two-round threshold Schnorr signature protocol. Unlike CGGMP21, FROST operates over groups where Schnorr signatures are native, making it suitable for:

- Ed25519 (Solana, Cosmos, Polkadot)
- BLS12-381 (Ethereum consensus, Lux consensus)
- BIP-340 Taproot (Bitcoin)

5.2 Two-Round Protocol

Algorithm 3 FROST Signing.

Require: Message m; key shares sk_i for $i \in S$; $|S| = t$
Ensure: Schnorr signature $(\tilde{R}, z)$

Round 1 (commitment):
for all $P_i \in S$ **do**
  Sample $(d_i, e_i) \leftarrow \mathbb{Z}_q^2$
  Broadcast $(D_i, E_i) = (d_i \cdot G, e_i \cdot G)$
end for
Round 2 (response):
$B \leftarrow \{(i, D_i, E_i)\}_{i \in S}$
$\rho_i \leftarrow H(\text{“binding”}, i, m, B)$ for each i
$\tilde{R} \leftarrow \sum_{i \in S}(D_i + \rho_i \cdot E_i)$
$c \leftarrow H(\tilde{R}, pk, m)$ ▷ Fiat-Shamir challenge
for all $P_i \in S$ **do**
  $z_i \leftarrow d_i + \rho_i \cdot e_i + \lambda_i \cdot sk_i \cdot c$
  Broadcast z_i
end for
$z \leftarrow \sum_{i \in S} z_i$
return $(\tilde{R}, z)$

5.3 FROST for BLS

FROST extends naturally to BLS signatures because BLS signing is a scalar multiplication $\sigma = sk \cdot H(m)$ in $\mathbb{G}_1$. Threshold BLS:

1. Each party computes partial signature $\sigma_i = sk_i \cdot H(m)$.
2. The coordinator reconstructs $\sigma = \sum_{i \in S} \lambda_i \cdot \sigma_i$.
3. Verification: $e(\sigma, G_2) = e(H(m), pk)$ where $pk \in \mathbb{G}_2$.

BLS threshold signing requires only *one* round (each party independently computes their partial signature). No precomputation or commitment phase is needed. This makes BLS threshold signing significantly faster than ECDSA threshold signing.

6 Key Resharing and Recovery

6.1 Problem: Validator Set Changes

Validator sets are not static. Validators join (new stake) and leave (unbonding). Without resharing, the threshold parameters (t, n) are fixed at DKG time. If validators leave, the effective threshold shrinks. If too many leave, signing becomes impossible.

6.2 Resharing Protocol

Resharing transitions from (t, n) to (t', n') without reconstructing the master key:

Algorithm 4 Proactive Key Resharing.

Require: Old committee $\mathcal{C} = \{P_1, \ldots, P_n\}$ with (t, n) shares
Require: New committee $\mathcal{C}' = \{P'_1, \ldots, P'_{n'}\}$ with threshold t'
Ensure: New shares sk'_j for $\mathcal{C}'$; master key sk unchanged

for all $P_i \in \mathcal{C}$, $i \in S$ where $|S| = t$ **do**
  Compute Lagrange coefficient λ_i
  Sample random polynomial $g_i(x)$ of degree $t' - 1$ with $g_i(0) = \lambda_i \cdot sk_i$
  Send $g_i(j)$ to each $P'_j \in \mathcal{C}'$ via encrypted channel
  Broadcast Feldman commitments for g_i
end for
for all $P'_j \in \mathcal{C}'$ **do**
  Verify each received share against commitments
  $sk'_j \leftarrow \sum_{i \in S} g_i(j)$
end for

Theorem 6.1 (Resharing correctness)**.** *After resharing, the new shares sk'_j are valid (t', n')-Shamir shares of the same master key sk.*

Proof. Let $F(x) = \sum_{i \in S} g_i(x)$. Then $F(0) = \sum_{i \in S} \lambda_i \cdot sk_i = sk$ by the Lagrange reconstruction property. Each g_i has degree $t' - 1$, so F has degree $t' - 1$. Therefore $sk'_j = F(j)$ are valid (t', n')-Shamir shares of sk. □

6.3 Recovery Without Key Reconstruction

If a party P_i loses their share (disk failure, corruption), the remaining parties can help P_i recover without ever materializing the master key:

1. Select a qualifying set S of t parties (not including P_i).
2. Each $P_j \in S$ computes $r_{j \to i} = \lambda_j \cdot sk_j$ evaluated at point i using a fresh random polynomial.
3. P_i reconstructs their share from the t recovery shares.

The master key is never reconstructed in this process. Each party contributes a partial recovery share that individually reveals nothing.

7 Slashing-Safe Fault Attribution

7.1 Problem

Proof-of-stake protocols rely on slashing to deter misbehavior: if a validator signs conflicting blocks, their stake is destroyed. With threshold signatures, the entity producing the signature is a *group*, not an individual. How do we attribute fault?

7.2 CGGMP21 Identifiable Abort

CGGMP21's identifiable abort property directly addresses this: if the signing protocol produces an invalid output or fails, the honest parties can identify the corrupted party. The proof of misbehavior is a verifiable transcript of the protocol messages that demonstrates the corrupted party deviated from the protocol specification.

7.3 Threshold Slashing Protocol

1. Each signing session produces a transcript $T = \{(i, m_i, \pi_i)\}$ where m_i is party i's message and π_i is a zero-knowledge proof of correctness.
2. If the final signature is invalid, the coordinator replays the transcript and identifies the first party whose message fails verification.
3. A *slashing proof* π_{slash} is submitted on-chain: the party's message, the verification failure, and the transcript prefix proving all prior messages were valid.
4. The on-chain slashing contract verifies π_{slash} and destroys the identified party's stake.

Proposition 7.1 (Slashing soundness)**.** *An honest threshold signer is never falsely slashed, assuming the discrete logarithm assumption holds and the zero-knowledge proofs are sound.*

8 Application to Lux Validator Network

8.1 Architecture

In the Lux threshold validator architecture:

- Each validator's BLS consensus key is split into (t, n) shares, where $t = \lceil 2n/3 \rceil + 1$ and n is the number of sub-parties (typically $n = 5$, $t = 4$ for a single validator).
- Sub-parties run on geographically distributed machines.
- Block signing uses threshold BLS (one-round, non-interactive).
- C-Chain transaction signing uses CGGMP21 (threshold ECDSA on secp256k1).
- Validator set changes trigger resharing with the new committee.

8.2 Deployment Topology

Table 2: Threshold configurations for Lux validator roles.

Role	**Scheme**	(t, n)	**Latency**
Block signing (BLS)	Threshold BLS	$(4, 5)$	2.1 ms
C-Chain signing (ECDSA)	CGGMP21	$(3, 5)$	4.2 ms
Bridge signing (EdDSA)	FROST	$(5, 7)$	6.8 ms
Emergency recovery	FROST	$(3, 5)$	4.0 ms

9 Benchmarks

All measurements on AMD EPYC 7763 nodes connected via 1 Gbps links with ∼1 ms round-trip latency between parties. Go 1.22 implementation using `luxfi/threshold` and `luxfi/mpc` libraries.

9.1 DKG Latency

Table 3: DKG protocol latency.

(t, n)	DKG Time	Share Size	Comms
$(2, 3)$	45 ms	32 B	192 B
$(3, 5)$	112 ms	32 B	640 B
$(5, 7)$	238 ms	32 B	1,568 B
$(7, 10)$	520 ms	32 B	3,200 B
$(15, 21)$	1.8 s	32 B	14,112 B

9.2 Signing Latency

Table 4: Signing latency by protocol and threshold.

Protocol	(t, n)	Signing Latency	Sig Size
Threshold BLS	$(2, 3)$	1.1 ms	48 B
Threshold BLS	$(4, 5)$	2.1 ms	48 B
Threshold BLS	$(7, 10)$	4.5 ms	48 B
CGGMP21 (ECDSA)	$(2, 3)$	2.8 ms	65 B
CGGMP21 (ECDSA)	$(3, 5)$	4.2 ms	65 B
CGGMP21 (ECDSA)	$(7, 10)$	18.7 ms	65 B
FROST (Ed25519)	$(2, 3)$	1.5 ms	64 B
FROST (Ed25519)	$(3, 5)$	2.4 ms	64 B
FROST (Ed25519)	$(5, 7)$	6.8 ms	64 B
FROST (Ed25519)	$(7, 10)$	12.3 ms	64 B

9.3 Comparison: MPC Signing vs. Single-Key Signing

Table 5: Overhead of MPC signing relative to single-key signing.

Scheme	Single-Key	MPC $(3, 5)$	Overhead
BLS sign	28 μs	2.1 ms	75×
ECDSA sign	18 μs	4.2 ms	233×
EdDSA sign	22 μs	2.4 ms	109×

Overhead is dominated by network round-trips, not computation. With co-located parties (LAN), overhead drops to 5–15×.

The absolute latencies (2–19 ms) are well within the requirements of blockchain consensus, which operates on 100 ms–2 s block times. MPC signing adds negligible latency relative to consensus rounds.

10 Related Work

Gennaro and Goldfeder [5] present a practical threshold ECDSA protocol with a trusted dealer. CGGMP21 [1] removes the trusted dealer and achieves UC security with identifiable abort.

Komlo and Goldberg [2] introduce FROST with a two-round signing protocol for Schnorr signatures. Groth [7] presents a non-interactive DKG for BLS threshold signatures. Boldyreva [6] formalizes threshold BLS signatures.

Our contribution is not a new threshold protocol but a *systems integration*: combining CGGMP21, FROST, and threshold BLS under a unified DKG and resharing framework, with concrete validator-specific extensions (slashing compatibility, validator set rotation, recovery).

11 Conclusion

Validators should not hold full signing keys. The single-key architecture is a design flaw inherited from a time when blockchain validators were assumed to be well-secured dedicated machines. Modern validator infrastructure—running on cloud VMs, managed by organizations with varying security postures, targeted by sophisticated adversaries—requires defense in depth.

Threshold cryptography provides this defense at acceptable cost: 2–19 ms signing latency, zero on-chain overhead (signatures are standard format), and a resharing protocol that handles validator set changes without key reconstruction. The implementation is deployed in production on the Lux network.

References

[1] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts," *ACM CCS 2020*, pp. 1769–1787, 2020.

[2] C. Komlo and I. Goldberg, "FROST: Flexible Round-Optimized Schnorr Threshold Signatures," *SAC 2020*, pp. 34–65, Springer, 2020.

[3] T. P. Pedersen, "A Threshold Cryptosystem without a Trusted Party," *EUROCRYPT '91*, pp. 522–526, Springer, 1991.

[4] P. Feldman, "A Practical Scheme for Non-Interactive Verifiable Secret Sharing," *FOCS '87*, pp. 427–437, IEEE, 1987.

[5] R. Gennaro and S. Goldfeder, "One Round Threshold ECDSA with Identifiable Abort," IACR ePrint 2020/540, 2020.

[6] A. Boldyreva, "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme," *PKC 2003*, pp. 31–46, Springer, 2003.

[7] J. Groth, "Non-Interactive Distributed Key Generation and Key Resharing," IACR ePrint 2021/339, 2021.

[8] A. Shamir, "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[9] Y. Desmedt, "Threshold Cryptosystems," *AUSCRYPT '89*, pp. 3–14, Springer, 1990.

[10] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," *Journal of Cryptology*, vol. 17, no. 4, pp. 297–319, 2004.