Non-Custodial Securities Custody via Threshold Signing

Zach Kelling

Lux Industries

2023

Abstract

We present a non-custodial custody architecture for digital securities in which no single party—not the exchange operator, not the user, not any custodian—can unilaterally move assets. The construction uses threshold signing with two complementary protocols: CGGMP21 (threshold ECDSA over secp256k1) for EVM-compatible chains and FROST (threshold Schnorr over Ed25519) for EdDSA chains. Each user's wallet is a 2-of-3 threshold address with shares distributed across the user's device, the exchange operator (auto-co-signs after compliance checks), and a cold backup. We prove three security properties: (1) *non-custodial guarantee*—no single party can forge a signature; (2) *liveness*—any two of three parties can sign, so asset recovery is possible even if one party is lost; (3) *key rotation safety*—shares can be refreshed (reshared) without changing the on-chain address. We describe the HSM attestation layer in which every signing share is co-attested by a hardware security module, and prove that the HSM attestation composes with the threshold protocol without weakening the unforgeability guarantee. The construction is deployed on the Liquid EVM (chain IDs 8675309–8675311) using the `luxfi/mpc` and `luxfi/threshold` libraries.

Contents

1 Introduction

Traditional securities custody relies on a custodian bank that holds assets on behalf of clients. The custodian is a single point of failure: if it is compromised, all client assets are at risk. If it becomes insolvent, clients are general creditors. Blockchain-based securities remove the custodian by making the private key the proof of ownership, but this shifts the risk to key management: a lost key means lost assets, and a stolen key means stolen assets.

Threshold signing eliminates the single-key problem. Instead of a single private key, the signing capability is distributed across multiple parties via secret sharing. A t-of-n threshold scheme requires t parties to cooperate to produce a valid signature, while any $t-1$ parties learn nothing about the key. The on-chain address is derived from the *aggregate* public key and is indistinguishable from a standard single-key address.

This paper describes the custody architecture deployed on the Liquid EVM for digital securities. The architecture uses two threshold signing protocols:

- **CGGMP21** [1]: Threshold ECDSA for secp256k1. Produces standard (r, s, v) Ethereum signatures. Used on EVM chains (Liquid EVM, Ethereum, Polygon, BSC).
- **FROST** [2]: Threshold Schnorr for Ed25519 and BIP-340 Taproot. Produces standard 64-byte Schnorr signatures. Used on Solana, TON, Cosmos, and Bitcoin Taproot.

Both protocols operate over the same Shamir secret sharing infrastructure (the `luxfi/threshold` LSS library) and share a common key generation ceremony.

2 Threshold Signing Protocols

2.1 CGGMP21: Threshold ECDSA

Definition 2.1 (CGGMP21 Protocol [1])**.** *A t-of-n threshold ECDSA protocol with three phases:*

1. ***Key Generation (DKG).*** *Parties jointly generate a Shamir sharing of the ECDSA private key x over $\mathbb{Z}_q$ (where q is the secp256k1 group order). Each party i receives share $x_i = f(i)$ for a random degree-$(t-1)$ polynomial f with $f(0) = x$. Each party also generates a Paillier key pair for zero-knowledge proofs.*
2. ***Presigning (offline).*** *Any t parties generate presignatures (k_i, χ_i) where k_i are shares of a random nonce k and $\chi_i = k_i \cdot x_i$. This uses Paillier-encrypted multiplication with ZK range proofs. Presignatures are message-independent and can be precomputed.*
3. ***Signing (online).*** *Given message m and a presignature, each party computes a partial signature. The combiner assembles these into a standard ECDSA signature (r, s) where $r = (k^{-1} \cdot G)_x$ and $s = k^{-1}(H(m) + r \cdot x)$. This is a single round.*

Theorem 2.1 (CGGMP21 Unforgeability)**.** *Under the ECDSA one-more discrete log assumption and the Paillier N-th residuosity assumption, any PPT adversary corrupting at most $t-1$ parties cannot produce a valid ECDSA signature on a new message except with negligible probability.*

Proof sketch. The proof proceeds by reduction to the ECDSA unforgeability game. A simulator $\mathcal{S}$ interacts with $t-1$ corrupted parties, providing simulated views of the DKG and presigning phases using equivocal commitments. The key insight is that the Paillier-based ZK proofs in the presigning phase are UC-secure: the simulator can extract the adversary's inputs from the ZK proofs and simulate the honest parties' messages. If the adversary produces a valid signature without the cooperation of at least one honest party, $\mathcal{S}$ uses this to break the ECDSA unforgeability assumption. □

2.2 FROST: Threshold Schnorr

Definition 2.2 (FROST Protocol [2])**.** *A t-of-n threshold Schnorr signing protocol:*

1. ***Key Generation.*** *Identical Shamir DKG as CGGMP21, but over the Ed25519 scalar field. The aggregate public key is $X = x \cdot G$ where G is the Ed25519 base point.*
2. ***Round 1 (commitment).*** *Each signer i samples nonce pair (d_i, e_i) and publishes commitments $(D_i, E_i) = (d_i G, e_i G)$.*
3. ***Round 2 (response).*** *Given the message m and all commitments, each signer computes a binding factor $\rho_i = H(i, m, D_1, E_1, \ldots, D_t, E_t)$, the group nonce commitment $R = \sum_i D_i + \rho_i E_i$, and the challenge $c = H(R, X, m)$. Each signer produces a partial signature $z_i = d_i + \rho_i e_i + c\lambda_i x_i$ where λ_i are Lagrange coefficients.*
4. ***Aggregation.*** *The combiner computes $z = \sum_i z_i$ and outputs signature (R, z).*

Theorem 2.2 (FROST Unforgeability)**.** *Under the discrete log assumption in the Ed25519 group and the random oracle model, any PPT adversary corrupting at most $t - 1$ parties cannot produce a valid Schnorr signature on a new message except with negligible probability.*

Proof sketch. The proof uses a reduction to the OMDL (one-more discrete log) problem. The simulator embeds an OMDL challenge into the DKG, programs the random oracle to produce consistent challenges, and uses the adversary's forgery to solve one more discrete log instance than the number of signing queries. The FROST commitment scheme (using binding factors ρ_i) prevents the rogue-key attack that affects naive Schnorr multisignature schemes. □

3 2-of-3 Wallet Architecture

Each user's wallet uses a 2-of-3 threshold with the following share distribution:

Share	Holder	Role
$\mathbf{s}_1$	User device	Stored in device secure enclave (iOS Keychain / Android Keystore)
$\mathbf{s}_2$	ATS operator	Auto-co-signs after pre-trade compliance (KYC, AML, sanctions, offering checks)
$\mathbf{s}_3$	Cold backup	Encrypted to user passphrase, stored in Shamir backup (S3 + user's custody)

Table 1: 2-of-3 share distribution for user wallets.

Theorem 3.1 (Non-Custodial Guarantee)**.** *No single party can produce a valid signature. Specifically:*

1. *The ATS operator alone cannot move user assets (holds only $\mathbf{s}_2$; needs $\mathbf{s}_1$ or $\mathbf{s}_3$).*
2. *The user alone cannot bypass compliance (holds only $\mathbf{s}_1$; needs $\mathbf{s}_2$ or $\mathbf{s}_3$).*
3. *A compromised backup alone is useless (holds only $\mathbf{s}_3$; needs $\mathbf{s}_1$ or $\mathbf{s}_2$).*

Proof. By the unforgeability of the threshold signing protocol (Theorem 2.1 for CGGMP21, Theorem 2.2 for FROST), producing a valid signature requires at least $t = 2$ shares. Each party holds exactly one share. Therefore no single party can sign. This holds information-theoretically for the secret sharing layer (any single share is uniformly random, independent of the key) and computationally for the signing protocol (under the OMDL/ECDSA assumptions). □

3.1 Signing Flows

Normal transaction (user + operator):

1. User initiates transaction on their device.
2. Device generates partial signature using $\mathbf{s}_1$.
3. ATS operator receives request, runs pre-trade compliance checks.

4. If compliant, operator generates partial signature using $\mathbf{s}_2$.
5. Signatures are combined into a standard on-chain signature.

Device lost (operator + backup):

1. User proves identity through KYC re-verification.
2. Backup share $\mathbf{s}_3$ is recovered using the user's passphrase.
3. Operator co-signs with $\mathbf{s}_2$ to authorize a reshare that generates a new $\mathbf{s}_1$ for the replacement device.
4. The on-chain address does not change (reshare preserves the aggregate public key).

Operator compromise (user + backup):

1. User activates emergency recovery using device share $\mathbf{s}_1$ and backup share $\mathbf{s}_3$.
2. Funds are moved to a new threshold address with a fresh operator share.

4 HSM Attestation

Every signing operation is co-attested by a hardware security module (HSM). The HSM does not hold a threshold share; instead, it provides an attestation that the signing environment is uncompromised.

Definition 4.1 (HSM Co-Signing)**.** *The HSM maintains a signing key k_{HSM} inside tamper-resistant hardware. For each threshold signing request:*

1. *The MPC node presents the partial signature σ_i and the signing context (message hash, signer set, policy).*
2. *The HSM verifies that the policy permits the operation (e.g., withdrawal amount within daily limit, destination on allowlist).*
3. *If the policy check passes, the HSM produces an attestation $\alpha_i = \mathsf{Sign}_{k_{HSM}}(\sigma_i \| context)$.*
4. *The partial signature is accepted only if α_i verifies.*

Theorem 4.1 (HSM Composition)**.** *Adding HSM attestation does not weaken the threshold unforgeability guarantee: if the threshold protocol is unforgeable under $t-1$ corruption, then the protocol with HSM attestation is also unforgeable under $t-1$ corruption, even if the HSM key is compromised.*

Proof sketch. The HSM attestation is an additional verification layer applied after partial signature generation. Removing the HSM check (as would happen if the HSM key is compromised and the adversary can forge attestations) reduces to the base threshold protocol, which is unforgeable by Theorem 2.1/2.2. The HSM can only *add* security (by rejecting unauthorized operations) but its compromise cannot *remove* the threshold unforgeability property. Formally: the advantage of an adversary against the HSM-augmented protocol is at most $\epsilon_{\text{threshold}} + \epsilon_{\text{HSM}}$, and the bound holds even when $\epsilon_{\text{HSM}} = 1$ (complete HSM compromise). □

4.1 Supported HSM Backends

Backend	Key Type	Attestation Latency
AWS CloudHSM	ECDSA P-256	3.2 ms
GCP Cloud HSM	ECDSA P-256	4.1 ms
Zymbit HSM6	ECDSA secp256k1	1.8 ms
YubiHSM 2	Ed25519	2.4 ms

Table 2: HSM attestation latency by backend.

5 Key Rotation

Definition 5.1 (Proactive Resharing)**.** *Given an existing t-of-n sharing of key x on polynomial f, a reshare protocol produces a new t'-of-n' sharing on a fresh polynomial g with $g(0) = f(0) = x$. The new shares are computed by having each old signer contribute an additive share of zero (a random polynomial h_i with $h_i(0) = 0$). The new shares are $g(j) = \sum_{i \in S} \lambda_i f(i) + \sum_{i \in S} h_i(j)$ for each new party j.*

Theorem 5.1 (Reshare Safety)**.** *Proactive resharing satisfies:*

1. ***Public key preservation.*** *The aggregate public key $X = x \cdot G$ is unchanged after reshare.*
2. ***Old share invalidation.*** *Old shares lie on polynomial f; new shares lie on polynomial $g \neq f$. Old shares cannot be combined with the new protocol state.*
3. ***No key reconstruction.*** *The secret x is never materialized during the reshare. Each party sees only its own old share plus received shares-of-zero.*

Proof. (1) follows because $g(0) = f(0) = x$, so $x \cdot G$ is unchanged. (2) follows because g is a uniformly random polynomial conditioned on $g(0) = x$, so the evaluations $g(j)$ are independent of $f(i)$ for $j \neq i$. Mixing old and new shares produces interpolation over inconsistent polynomials, which yields an incorrect secret. (3) follows because each old signer generates their share-of-zero locally (using fresh randomness), and the zero-secret polynomial h_i has $h_i(0) = 0$. The sum $\sum h_i(j)$ re-randomizes the polynomial without any party knowing the full randomness. □

Key rotation is performed periodically (default: every 7 days) to limit the exposure window. An attacker must compromise t shares within a single rotation epoch; shares from different epochs are incompatible.

6 Settlement Flow

The full settlement lifecycle for a securities trade on the Liquid EVM:

1. **Order.** User submits a trade order via the ATS API.
2. **Match.** The CEX matching engine (Lux CEX) matches the order against the order book.
3. **Record.** The matched trade is recorded in PostgreSQL (source of truth for trade data).
4. **Queue.** The trade is queued in the `pending_settlements` table.
5. **Sign.** A settlement cron (30s interval) picks up pending trades, constructs the on-chain transactions (mint/burn of SecurityToken), and initiates MPC signing.
6. **MPC.** The user's device share ($\mathbf{s}_1$) and the operator's share ($\mathbf{s}_2$) cooperate to produce a threshold signature.
7. **HSM.** The HSM co-signs the transaction, attesting that the signing environment is uncompromised.
8. **Broadcast.** The signed transaction is broadcast to the Liquid EVM.
9. **Confirm.** The chain confirms the transaction. The indexer picks up the event and updates the read cache.
10. **Notify.** The user receives a settlement confirmation.

If the chain is temporarily unavailable, trades continue to match and record in PostgreSQL. The settlement cron retries pending settlements until chain connectivity is restored. The database is the durable record; the chain is the authoritative record once settlement confirms.

7 Integration with EVM Precompiles

The Liquid EVM includes native precompiles for the cryptographic primitives used in threshold signing:

Precompile	Operation	Gas
FROST (`0x0300...0005`)	Verify Schnorr signature	3,000
SR25519 (`0x0300...0001`)	Verify SR25519 signature	3,000
Ed25519 (`0x0300...0002`)	Verify Ed25519 signature	3,000
CGGMP21 (`0x0300...0006`)	Verify threshold ECDSA	3,000
secp256r1 (`0x0300...0003`)	Verify P-256 signature	3,500

Table 3: Signature verification precompiles on the Liquid EVM.

These precompiles enable on-chain verification of threshold signatures at the same gas cost as standard ECDSA verification. Smart contracts can verify that a settlement transaction was properly authorized by the threshold scheme without needing to implement the verification logic in Solidity.

8 Formal Verification

The core properties are mechanized in Lean 4:

- `Crypto.Threshold.CGGMP21`: keygen correctness, identifiable abort, presignature one-time use, offline/online separation, key never reconstructed.
- `Crypto.FROST`: correctness, unforgeability, robustness, majority threshold safety, share refresh.
- `Crypto.Threshold.Reshare`: public key preservation, old share invalidation, threshold maintenance, no key reconstruction, Byzantine safety.
- `Crypto.NonCustodial`: 2-of-3 non-custodial guarantee, liveness (any 2 can sign), single-party impotence.

9 Conclusion

Threshold signing eliminates the single-key custody problem for digital securities. The 2-of-3 architecture ensures that no single party can move assets (non-custodial), any two parties can sign (liveness), and key rotation preserves the on-chain address (operational continuity). HSM attestation adds a defense-in-depth layer without weakening the threshold guarantee. The construction is deployed on the Liquid EVM with native precompile support for efficient on-chain verification.

References

[1] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, U. Peled. *UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts*. CCS 2020.

[2] C. Komlo, I. Goldberg. *FROST: Flexible Round-Optimized Schnorr Threshold Signatures*. SAC 2020.

[3] A. Shamir. *How to Share a Secret*. Communications of the ACM, 22(11):612–613, 1979.

[4] T. Pedersen. *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*. CRYPTO 1991.

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

[6] D. Boneh, X. Boyen, E.-J. Goh. *Hierarchical Identity Based Encryption with Constant Size Ciphertext*. EUROCRYPT 2005.