title

Zach Kelling

Satschel, Inc.

April 2026

Abstract

Regulated broker-dealers and transfer agents operating under SEC and FINRA jurisdiction must maintain the ability to produce records on lawful request. This requirement is fundamentally incompatible with sovereign key models where the operator cannot decrypt user data. We describe the key architecture deployed by Satschel, Inc. for the Liquidity ATS, transfer agent, and broker-dealer services. The system uses a KMS-held master key to derive per-organization data encryption keys via HMAC-SHA256, applies AES-256-GCM field-level encryption to sensitive data at rest, and employs CGGMP21/FROST threshold MPC for on-chain signing where no single party reconstructs the full private key. The blockchain serves as a WORM-compliant audit trail for all securities operations. We formalize the regulatory access model: the operator holds decrypt capability for data, but transaction signing requires threshold cooperation.

Contents

1 Regulatory Requirements

The architecture is constrained by three regulatory regimes.

1.1 SEC Rule 17a-4: Record Production

SEC Rule 17a-4 [1] requires broker-dealers to preserve records in a non-rewritable, non-erasable format (WORM) and to produce them upon SEC examination or subpoena. Subsection (f) permits electronic storage if the system:

1. Preserves records exclusively in a non-rewritable, non-erasable format.
2. Verifies automatically the quality and accuracy of the storage media.
3. Serializes the original and duplicate copies of records.
4. Has the capacity to readily download indexes and records.
5. Includes an audit system for downtime, failures, and record changes.

The operator must be able to decrypt and produce any record within the retention period. A system where the operator cannot decrypt (e.g., end-to-end encryption with user-held keys) would violate Rule 17a-4(f)(2)(ii)(A), which requires the broker-dealer to have the ability to produce records to regulators.

1.2 FINRA Rule 4511: Retention and Format

FINRA Rule 4511 [2] requires members to make and preserve books and records as required under FINRA rules, the Exchange Act, and applicable Exchange Act rules. Records must be retained for at least six years, with the first two years in an easily accessible place. The retention obligation extends to electronic communications, trade confirmations, account statements, and all documents relating to the business of the member.

1.3 Bank Secrecy Act and AML

The Bank Secrecy Act [3] and FinCEN regulations require broker-dealers to maintain records of customer identification (CIP), suspicious activity reports (SARs), and currency transaction reports (CTRs). These records must be producible on regulatory demand. Customer identification data—including SSN, tax ID, and identity document numbers—must be retained and producible for at least five years after account closure.

Invariant 1.1 (Operator Decrypt Capability)**.** *For every encrypted field f stored in the system, the operator must possess the capability to derive the decryption key for f within the retention period.*

2 Key Hierarchy

2.1 Master Key

The master key is a 256-bit symmetric key stored in Google Cloud KMS, backed by HSM (Cloud HSM with FIPS 140-2 Level 3 certification in production). The key never leaves the KMS boundary in plaintext. In the deployed system, the environment variable `TENANT_ENCRYPTION_KEY` holds the master key material, loaded from KMS at service startup.

In development, the master key may be unset, in which case field-level encryption is disabled and a warning is logged:

Listing 1: orgdb.go: encryption disabled warning

```
if masterKey == "" {
    slog.Warn("TENANT_ENCRYPTION_KEY not set -- " +
```

```
        "per-tenant field encryption disabled (dev mode)")
    return
}
```

2.2 Per-Organization Data Encryption Key

Each tenant organization receives a unique 256-bit data encryption key (DEK) derived deterministically from the master key and the organization's slug:

$$\text{DEK}_{\text{org}} = \text{HMAC-SHA256}(\text{masterKey}, \text{orgSlug}) \tag{1}$$

The derivation is implemented in `orgdb.go` as:

Listing 2: orgdb.go: per-org DEK derivation

```
func (o *orgEncryptor) OrgDEK(tenantSlug string) string {
    if len(o.masterKey) == 0 {
        return ""
    }
    mac := hmac.New(sha256.New, o.masterKey)
    mac.Write([]byte(tenantSlug))
    return hex.EncodeToString(mac.Sum(nil))
}
```

This produces a 32-byte (256-bit) key unique to each organization. The derivation is deterministic: the same master key and slug always produce the same DEK, which means no per-org key storage is required.

Property 2.1 (Key Isolation)**.** *For distinct organization slugs $s_1 \neq s_2$, $\Pr[DEK(s_1) = DEK(s_2)] \leq 2^{-256}$ under the assumption that HMAC-SHA256 is a pseudorandom function.*

2.3 Field-Level Encryption

Individual record fields are encrypted using AES-256-GCM with a random 12-byte nonce generated per encryption operation. The encrypted fields are defined per collection:

Collection	Encrypted Fields
securities	cusip, isin
kyc_submissions	ssn, document_number, tax_id
kyc_documents	document_data
stakeholders	tax_id, ssn, ein
transfer_requests	account_number

Table 1: Collections and fields subject to AES-256-GCM encryption.

The encryption is applied via Base record hooks. On `OnRecordCreate` and `OnRecordUpdate`, each field in the encrypted set is replaced with its ciphertext prefixed by `enc:org:`. On read, the prefix is detected and the field is decrypted transparently.

Definition 2.1 (Ciphertext Format)**.** *For plaintext p and org DEK k:*

$$\texttt{enc:org:}\ \textit{Base64}(\textit{nonce} \| \textit{AES-256-GCM}(k, \textit{nonce}, p))$$

where nonce is 12 bytes sampled uniformly at random.

2.4 FHE-Ready Fields

A subset of fields is designated for future fully homomorphic encryption, enabling compliance checks (e.g., accreditation threshold verification) on encrypted data without decryption:

Listing 3: orgdb.go: FHE-designated fields

```
var fheEncryptedFields = map[string]map[string]struct{}{
    "kyc_submissions": {"investment_amount": {},
                        "income": {}, "net_worth": {}},
    "stakeholders":    {"investment_amount": {},
                        "income": {}, "net_worth": {}},
}
```

These fields currently use AES-256-GCM as a fallback. When the `luxfi/fhe` library is integrated, they will transition to lattice-based FHE ciphertexts supporting homomorphic comparison (Section 4).

2.5 Post-Quantum Signatures on Records

Collections containing securities-critical records receive a post-quantum signature on creation. The current implementation uses HMAC-SHA256 as a placeholder; the target is ML-DSA-87 (FIPS 204 [6]) via `luxfi/crypto/mldsa`:

Listing 4: orgdb.go: PQ-signed collections

```
var pqSignedCollections = map[string]struct{}{
    "transfers":          {},
    "cap_table_entries": {},
    "securities":         {},
}
```

3 MPC Signing Architecture

Transaction signing uses threshold MPC protocols. The full private key is *never reconstructed* in any single process's memory.

3.1 Protocol Selection

Curve/Scheme	Protocol	Use Case
secp256k1 (ECDSA)	CGGMP21	EVM transaction signing
Ed25519 (EdDSA)	FROST	Non-EVM chains, attestations

Table 2: Threshold signing protocols by curve.

CGGMP21 [4] provides threshold ECDSA without trusted dealer setup. FROST [5] provides threshold Schnorr/EdDSA signatures. Both protocols guarantee that no coalition of fewer than t parties can produce a valid signature.

3.2 Threshold Configuration

The MPC network is configured as t-of-n where t and n are set per network deployment. From `cli/pkg/mpc/node.go`:

Listing 5: mpc/node.go: threshold configuration

```
type NodeConfig struct {
    NodeID      string `json:"nodeId"`
    NodeIndex   int    `json:"nodeIndex"`
    Threshold   int    `json:"threshold"`
    TotalNodes  int    `json:"totalNodes"`
    Network     string `json:"network"`
    Peers       []string `json:"peers"`
    // ...
}
```

Production deployment uses 2-of-3: the ATS operator holds one shard, a co-signing HSM holds one shard, and a recovery shard is stored in encrypted cold storage. The operator alone cannot sign.

3.3 Write Path

All securities operations follow the chain-first write path implemented in `ta/chainwire.go`:

1. API receives request (e.g., mint shares, transfer, burn).
2. `ChainWire` constructs the transaction payload.
3. Payload is sent to the MPC service: `POST {MPC_URL}/v1/deploy` or `POST {MPC_URL}/v1/execute`.
4. MPC nodes perform threshold signing (CGGMP21 for ECDSA).
5. Signed transaction is broadcast to the EVM chain.
6. Chain event is emitted; indexer updates the local SQLite/Base cache.

Listing 6: chainwire.go: write path comment

```
// Write path:
//   API -> ChainWire -> MPC sign -> on-chain tx
//       -> chain event -> indexer updates SQLite
// Read path:
//   API -> SQLite (fast, always re-derivable from chain)
```

3.4 Separation of Concerns

MPC signing and data encryption serve different purposes and use different key material:

Function	Key Material	Holder
Data encryption (PII)	KMS master key $\rightarrow$ HMAC DEK	Operator (KMS)
Transaction signing	MPC key shares	Threshold (t-of-n)
Record integrity	PQ signing key (ML-DSA-87)	Operator

Table 3: Key material separation.

The operator can decrypt PII data (required by regulation) but cannot unilaterally sign transactions (prevented by threshold).

4 Chain as Audit Trail

4.1 Securities Operations On-Chain

Every securities lifecycle event is recorded on-chain via ERC-3643 compliant contracts:

- **SecurityToken** (ERC-20): deploy, mint, burn, transfer per security.
- **SettlementRegistry**: maps trade UUIDs to transaction hashes.
- **ComplianceRegistry**: whitelist, lockup periods, jurisdiction restrictions.

- **USDL**: stablecoin (ERC-20) for cash-leg settlement, 1:1 with USD.

From `ats/onchain.go`, the ATS creates MPC intents for every financial action:

Listing 7: onchain.go: MPC intent creation

```
// Every financial action (deposit, withdrawal, trade)
// has an immediate on-chain counterpart.
// The chain is the ledger; PostgreSQL is the audit log.
```

4.2 WORM Compliance

The blockchain satisfies the WORM (Write Once, Read Many) requirement of SEC Rule 17a-4:

1. **Non-rewritable**: transactions are immutable once finalized.
2. **Non-erasable**: no mechanism exists to delete confirmed transactions.
3. **Timestamped**: block timestamps provide ordering.
4. **Serialized**: block numbers provide sequential indexing.
5. **Auditable**: any party with chain access can verify the full history.

4.3 Off-Chain Index

The local database (Base/SQLite in development, PostgreSQL in production) is a read cache of chain state. It is never authoritative:

> *"SQLite is never authoritative—chain always wins on conflict."* — `ta/chainwire.go`

If the index diverges from chain state, the indexer replays chain events to reconstruct the correct state. This ensures that even complete database loss does not result in loss of securities records.

5 Regulatory Access Model

5.1 Decryption on Lawful Request

The operator decrypts PII data only in response to lawful requests:

- SEC examination under Section 17(a) of the Exchange Act.
- FINRA audit under Rule 8210 (information requests).
- Subpoena from a court of competent jurisdiction.
- FinCEN request under the Bank Secrecy Act.

The operator never volunteers decrypted data. Production only what is specifically requested, scoped to the requesting authority's jurisdiction and the specific records identified.

5.2 Retention Schedule

Record Type	Regulation	Period	Access
Trade blotters	SEC 17a-4(b)(1)	6 years	2 years immediate
Customer accounts	SEC 17a-4(b)(5)	6 years	2 years immediate
Communications	SEC 17a-4(b)(4)	3 years	2 years immediate
CIP/AML records	BSA/31 CFR 1023	5 years after close	Immediate
SAR filings	FinCEN	5 years	Immediate

Table 4: Record retention requirements.

5.3 Backup Architecture

1. **Primary**: Base/SQLite per-tenant files (encrypted at rest).
2. **Replication**: PostgreSQL streaming replication (production).
3. **Object storage**: Daily encrypted snapshots to S3-compatible storage (Hanzo S3 → GCS Archive tier).
4. **Chain**: immutable on-chain records serve as ultimate backup for all securities operations.

Backup encryption uses the same KMS master key. Recovery requires KMS access, which is restricted to production service accounts and break-glass procedures.

6 Threat Model

6.1 KMS Master Key Compromise

Impact: all per-org DEKs are derivable; all encrypted PII is exposed.

Mitigation: Google Cloud KMS with HSM backing (FIPS 140-2 Level 3). Key access is restricted to service accounts with IAM conditions (IP range, time-of-day). Key rotation is supported: new master key encrypts new records; old records are re-encrypted during a rotation window.

Detection: Cloud Audit Logs on all KMS operations; alerts on anomalous access patterns.

6.2 MPC Signer Compromise

Impact: if $\geq t$ shards are compromised, an attacker can sign arbitrary transactions (mint, transfer, burn securities).

Mitigation: 2-of-3 threshold with shards on separate infrastructure. One shard in operator service, one in co-signing HSM, one in encrypted cold storage. Compromising any single shard is insufficient.

Detection: all MPC signing operations emit chain events. Unauthorized transactions are visible on-chain immediately.

6.3 Insider Threat

Impact: a privileged insider with KMS access can decrypt PII. An insider with MPC access to one shard cannot sign transactions alone.

Mitigation: separation of duties. KMS access and MPC shard access require different IAM roles. The on-chain audit trail is immutable—an insider cannot erase evidence of transactions.

Detection: all KMS access is logged. All chain transactions are public within the network.

6.4 Regulatory Overreach

Impact: a request exceeding lawful scope could expose data beyond what is required.

Mitigation: the operator scopes decryption to specific records identified in the request. Per-org key derivation means decrypting one organization's data does not expose another's. Legal counsel reviews all production requests.

6.5 Database Loss

Impact: loss of the local database (Base/SQLite or PostgreSQL).

Mitigation: the chain is the source of truth for all securities operations. The database is a rebuildable index. PII records are backed up to encrypted object storage daily.

7 Formal Properties

We state the key security properties of the architecture.

Theorem 7.1 (Regulatory Producibility)**.** *For any encrypted field f stored in collection C with organization slug s, the operator can produce the plaintext of f given access to the KMS master key:*

$$Decrypt(DEK(s), f) = p \quad where\ f = Encrypt(DEK(s), p)$$

This holds for the entire retention period as long as the KMS master key is available and the ciphertext has not been corrupted.

Theorem 7.2 (Signing Non-Unilaterality)**.** *No single party holding fewer than t MPC key shares can produce a valid ECDSA or EdDSA signature. In the 2-of-3 production configuration, the operator holds exactly 1 share, which is below the threshold.*

Theorem 7.3 (Audit Immutability)**.** *For any securities operation o recorded on-chain at block height h, no party (including the operator) can modify or delete o without producing a chain fork visible to all network participants.*

8 Conclusion

The Liquidity key architecture resolves the tension between data protection and regulatory access. Sensitive data is encrypted at rest with per-organization keys, but the operator retains decrypt capability as required by SEC Rule 17a-4 and FINRA Rule 4511. Transaction signing is separated from data encryption: MPC threshold signing prevents unilateral operator control over securities operations, while the blockchain provides an immutable audit trail. The architecture satisfies both the "must be able to produce records" requirement of securities regulation and the "should not have unilateral control" principle of non-custodial design.

References

[1] Securities and Exchange Commission, "Rule 17a-4: Records to be preserved by certain exchange members, brokers, and dealers," 17 CFR §240.17a-4, as amended 2023.

[2] Financial Industry Regulatory Authority, "Rule 4511: General requirements (books and records)," FINRA Manual, 2011.

[3] Bank Secrecy Act, 31 U.S.C. §§5311–5332; implementing regulations at 31 CFR Chapter X.

[4] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, U. Peled, "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts," in *Proceedings of the 2020 ACM CCS*, 2020.

[5] C. Komlo and I. Goldberg, "FROST: Flexible Round-Optimized Schnorr Threshold Signatures," in *Selected Areas in Cryptography (SAC)*, 2020.

[6] National Institute of Standards and Technology, "FIPS 204: Module-Lattice-Based Digital Signature Standard (ML-DSA)," August 2024.

[7] National Institute of Standards and Technology, "FIPS 197: Advanced Encryption Standard (AES)," November 2001.