Nihilium

Slashable Conditional Key Release

Technical Specification v0.5

Abstract

Nihilium is a crypto-economic protocol for conditional secret recovery. It approximates the functional properties of witness encryption - binding a secret to an NP condition such that satisfying the condition is necessary and sufficient for recovery - using a combination of homomorphic two-party key generation, client-side zero-knowledge proofs, and economic slashing enforcement.

The protocol introduces no new cryptographic primitives. Its contribution is a novel composition of existing techniques that achieves context-agnostic, censorship-resistant, and privacy-preserving secret recovery without a trusted custodian. Cryptographically, the construction provides threshold guarantees comparable to existing MPC systems. The security improvement over those systems comes from an economic enforcement layer: on-chain slashing makes collusion attributable and punishable, forced execution eliminates liveness failures, and processor misbehavior results in permanent total stake loss. MPC systems operate permanently at the trust level that Nihilium falls to only under catastrophic chain failure. The protocol raises the floor of practical security high enough to enable real-world UX without sacrificing self-custody.

Target Audience: Protocol developers and technical integrators.

Table of Contents

1. Design Principles

1.1 Security Model: Floor, Not Ceiling

Nihilium is a crypto-economic construction, not a purely cryptographic one. The distinction matters and should be held clearly throughout this document.

The protocol does not provide the mathematical guarantee that witness encryption would: that decryption without a valid witness is computationally infeasible regardless of party behavior. Instead it provides an economic guarantee: that decryption without a valid witness is irrational, because doing so transfers permanent slash capability to whoever receives the disclosed key. The cryptography enforces condition verification. The economics enforce honest operator behavior.

This framing has a precise implication: **the protocol raises the floor of security, not the ceiling.** It is designed to make self-sovereign secret recovery viable at scale - for users who currently have no recovery mechanism at all - not to replace strong cryptographic guarantees where they are available. For high-value assets requiring institutional-grade protection, dedicated custody infrastructure with hardware security modules, air-gapped ceremonies, and multi-institutional governance should be used. Nihilium targets the vast majority of users for whom the realistic alternative is not sophisticated threshold cryptography but rather nothing: a seed phrase in a drawer, a master password in memory, or complete dependence on a custodian.

The threat model Nihilium is designed to address is opportunistic and economically motivated misbehavior, not targeted state-level attacks against specific high-value secrets. Against that threat model, economic enforcement is not merely adequate - it is structurally superior to the trust assumptions of existing alternatives, as detailed in Section 13.

1.2 Primitive Composition

Nihilium composes existing techniques in a novel configuration:

Technique	Role in Nihilium
Combinatorial Threshold Encryption	Redundancy, liveness, oracle-resistant threshold recovery, and DLEQ-attributable collusion evidence
Homomorphic Encryption (ECElGamal)	Two-party key construction without plaintext exposure
Zero-Knowledge Proofs (Groth16)	Client-side condition proof generation
Blockchain (EVM)	Enforcement only - slashing, registration, forced execution
Staking	Reputation capital, not per-seal collateral
Merkle Trees	Timestamping and absence proofs via datastream
TEEs	Optional operational security for processors

No new mathematics are introduced. All primitives use established libraries.

ZK Proof System. The protocol currently uses Groth16 for its compact proof size (~200 bytes) and efficient on-chain verification. Groth16 requires a trusted setup ceremony: a universal Powers of Tau (Phase 1, reusable across circuits) and a circuit-specific Phase 2 that generates per-circuit parameters. Both phases should be conducted via multi-party computation ceremonies to ensure that no single participant can compromise soundness. Circuit changes require new Phase 2 ceremonies. The protocol architecture is

ZK-system-agnostic - migration to alternative proof systems (PLONK, STARK, or future constructions) requires no changes to the protocol design, only to the proof generation and verification components.

Curve Choice. The protocol currently uses bn254 (BN128) for ZK proofs and Baby Jubjub for HE and signatures. bn254 provides approximately 100–102 bits of security following the Kim-Barbulescu extended Tower Number Field Sieve result - below the NIST 128-bit recommendation but sufficient for the floor-level threat model the protocol targets. Migration to BLS12-381 (128-bit security) is straightforward and requires only library-level changes, as the protocol's cryptographic operations are curve-agnostic at the architectural level. For deployments requiring longer-term security margins, BLS12-381 is recommended.

1.3 Value Decomposition

The protocol does not attempt 1:1 economic backing between processor stake and sealed asset value. Instead, what is sealed is never the asset itself - only a component that is worthless in isolation. A password without its encrypted file has zero standalone value. An encrypted file without its password has zero standalone value. Their combination has the full asset value.

This means processor stake needs only to exceed the value an attacker could extract from the sealed component alone - which is zero under correct usage. The stake-to-asset-value comparison that initially appears problematic is therefore the wrong frame.

Value decomposition is strongly advised but not a protocol requirement. Without it, the sealed component carries extractable value and the economic security model still applies - it simply requires that processor stake exceeds that extractable value for the attack to remain irrational. The protocol functions identically in both cases; the difference is purely in the economics of the security guarantee.

Operational requirement: Value decomposition holds if and only if the encrypted counterpart is stored in a security domain independent from any processor's jurisdiction. The user is responsible for maintaining this separation. Concretely: if the encrypted file is stored locally on a personal device, and the password is sealed with Nihilium across processors in different jurisdictions, an attacker must independently breach the local device *and* compromise k processors to gain access to the decrypted data. If both the encrypted file and the sealed password are accessible through the same legal or technical attack surface - for example, both stored with cloud providers subject to the same jurisdiction - the decomposition weakens proportionally.

Integrators should design recovery flows that enforce this separation by default: encrypted data stored client-side or in a different trust domain than the Nihilium-sealed recovery key.

1.4 Client-Side Proof Burden

Unlike MPC and threshold systems where servers evaluate conditions, Nihilium inverts responsibility: **the client proves, the processor validates.** This inversion has several consequences:

- Processors remain stateless and lightweight
- The condition language is infinitely extensible without processor software updates
- Condition contents remain hidden from processors until unsealing
- Proof generation cost is borne by the party with the strongest interest in correctness

2. Recovery Landscape & Motivation

2.1 The Recovery Gap

The current key management landscape offers users a binary choice: full self-sovereignty with no recovery, or custodial solutions that sacrifice autonomy for convenience. There is no widely deployed middle ground.

The custodial axis has matured significantly. Institutional MPC custody providers manage key shares and operational complexity on behalf of clients. Consumer-facing solutions use social login flows that split keys between the user's device, a cloud share, and the provider's infrastructure. These work well within their constraints, but they are custodial in the meaningful sense: the user's ability to recover depends on a specific counterparty continuing to exist, remaining cooperative, and not being compelled or incentivized to deny service.

The non-custodial axis has seen progress in *authorization* - controlling who can sign transactions - but remains largely stagnant for *recovery*. Smart contract wallets offer multisig and modular access control. ERC-4337 account abstraction enables flexible signing policies and social recovery modules. Passkey-based wallets remove seed phrases from the user's cognitive burden. These are genuine improvements in how keys are *used*, but they assume the keys continue to exist.

The recovery mechanisms available to non-custodial users remain essentially unchanged: write down a seed phrase, store it somewhere, hope it survives. Social recovery via smart contract guardians sees minimal adoption due to coordination burden. Shamir-based seed splitting distributes fragments across physical locations. These approaches require near-perfect operational security from users who, by definition, are not security professionals.

2.2 The Encryption Recovery Problem

The gap becomes critical as the ecosystem moves toward encrypted state as a default. Private transactions, encrypted mempools, confidential documents, and end-to-end encrypted messaging with blockchain identity all create encrypted data that cannot be recovered through signing key rotation alone. If your encryption key is lost and your data is encrypted, no amount of smart contract sophistication helps. You need the actual key back, or something that can conditionally reconstruct it.

This is the problem Nihilium is designed to solve: conditional recovery of encryption keys and secrets, without custodial dependency, at a cost and latency that enables default integration by application developers. The target audience is developers building wallets, password managers, and encrypted applications - not end users directly.

2.3 Why Existing Threshold Systems Are Insufficient

Existing threshold decryption systems use distributed key generation (DKG) and multi-party computation (MPC) to avoid single points of key compromise. These systems provide cryptographic threshold guarantees but suffer from structural limitations that Nihilium addresses:

- **Undetectable and unpunishable collusion.** If k node operators in a DKG-based system collude, they reconstruct the full key with zero on-chain evidence. No slashing, no forensics, no accountability. This is a recognized unsolved problem in deployed threshold systems.
- **No enforcement of liveness.** MPC node operators can refuse to participate in decryption for any reason - legal pressure, commercial disagreement, operational policy - and the user has no recourse.
- **Coordination burden.** DKG requires interactive, multi-round setup ceremonies. Adding or removing participants requires re-sharing protocols. For long-lived secrets, the operator set must be actively maintained over time.
- **Hardware trust dependencies.** Some deployed MPC implementations run key shares inside TEEs (Intel SGX / AMD SEV), introducing supply-chain centralization risk that no amount of node decentralization addresses.
- **Limited condition extensibility.** Conditions are evaluated server-side, requiring the node network to support each condition type's execution environment.

A detailed comparison is provided in Section 13.

3. System Actors

3.1 The Client

The client holds secrets and bears full responsibility for cryptographic operations:

- Defines unseal conditions and computes the `unseal_root` commitment
- Generates all zero-knowledge proofs
- Executes homomorphic encryption operations within ZK circuits
- Initiates both sealing and unsealing interactions

3.2 The Processor

Independent operators that commit to conditional decryption via economic stake. Processors are not nodes in a consensus network - they operate independently, without coordination or communication with each other.

Holds:

- EdDSA signing key (Baby Jubjub) for commitment signatures
- ECElGamal decryption key (Baby Jubjub) for homomorphic ciphertext decryption

Responsibilities:

- Signs sealing commitments binding it to specific unseal conditions
- Validates chained proof execution at unsealing time
- Decrypts homomorphic ciphertexts upon successful proof validation
- Faces slashing for premature disclosure or refusal to serve valid requests

Operational properties:

- Entirely stateless (in-memory processing)
- Scales horizontally without coordination
- Optionally runs within TEEs for operational security hardening (not required for security model)
- Each processor independently slashable

Economic property: Processor stake is reputation capital at risk across all commitments, not per-seal collateral. A single stake covers all seals the processor has committed to. Any single provable misbehavior forfeits the entire stake permanently.

Longevity requirement: Processors must remain operational for the lifetime of all seals they have committed to, as each processor holds an EC-ElGamal decryption key that cannot be transferred or reconstructed. This is comparable to the longevity dependency in any custodial or threshold system - the difference is not in the dependency itself but in the enforcement mechanism that governs processor behavior during that lifetime (see Section 8).

Selection criteria for integrators: When selecting processors, developers should consider stake size, operational track record, grace period length, and - critically - **jurisdictional diversity**. For recovery configurations intended to remain viable over extended time horizons (years or decades), the threshold parameter k should be set low enough relative to n that the expected surviving processor set comfortably

exceeds k after accounting for processor mortality (offline, slashed, ceased operations). A 3-of-15 configuration where half the processors are expected to disappear over a decade still provides C(7,3) = 35 recovery paths.

3.3 The Datastream

A timestamping and data anchoring mechanism backed by stake. Multiple independent datastream operators may exist; clients select which to anchor to at sealing time and may distribute across multiple operators for redundancy.

Architecture:

- Two-tier Merkle tree (top-level + sub-trees)
- Smart contract per operator, updated each Ethereum block
- Sub-trees mirrored via IPFS for public availability
- Periodic updates anchored to Ethereum block height

Functions:

- Anchors the reveal value that initiates unsealing - creating mandatory public observability of all unsealing attempts
- Anchors values used in time-based unseal conditions
- Produces stake-backed exclusion claims: signed assertions that a value does not exist in the datastream within a specified time window. If disproven, the operator is slashed.
- Provides a forced inclusion mechanism for censorship resistance

Inclusion commitment mechanism: Upon receiving data for inclusion, the datastream operator signs a commitment specifying the data and the insertion deadline (the next Merkle tree update). If the data does not appear in the committed update, the signed commitment constitutes slashing evidence. This converts the datastream from a trusted timestamping service into an economically bound one - the operator cannot silently drop data because they have signed a commitment to include it. Insertion proofs can also be force-requested through on-chain contracts, providing a permissionless fallback if the operator refuses to sign inclusion commitments.

The datastream's temporal anchor is Ethereum block height - the same chain on which slashing is enforced. All temporal claims, condition verification, and economic enforcement share a single source of truth.

Dual implementation:

1. Off-chain aggregated version (primary, efficient)
2. Permissionless on-chain version (liveness guarantee, higher cost)

The on-chain version ensures that even if every datastream operator censors or goes offline, the user can fall back to direct on-chain submission at higher cost. This provides genuine censorship resistance: degraded performance, but no liveness failure.

3.4 The Blockchain (EVM)

Permissionless enforcement layer and registry of last resort:

- Processor and datastream stake registration and slashing
- Public key registration (with proof of knowledge - see Section 2.2)
- Unseal condition verification contract registry
- Forced execution mechanism

In normal operation, only the datastream requires regular on-chain transactions. Clients and processors interact off-chain unless slashing or forced execution is triggered. The chain exists as a backstop for adversarial cases, not as a workflow dependency.

3.5 The Committee (Transitional)

Initially centralized governance, planned transition to DAO:

Powers:

- Determines slashing reward recipients (handles edge cases: self-slashing, protocol bugs)
- Maintains registry of eligible unseal condition contracts (validates immutability, prevents bait-and-switch contracts)

Explicit limitations:

- Cannot deny processor or datastream registration
- Cannot intervene in slashing mechanics once triggered
- Cannot modify rules retroactively

The committee sets rules but cannot intervene in enforcement once those rules are triggered.

4. Severed Commitment Scheme

4.1 Problem

During sealing, processors must sign commitments that later prove their obligations. Standard commitment schemes create linkability: signatures, timing, and parameters tie sealing interactions to unsealing requests, making deanonymization possible.

Requirement: Every unsealing request must be indistinguishable from every other, at the protocol level, such that processors cannot correlate sealing and unsealing interactions.

4.2 Construction

Sealing phase:

1. Client generates random nonce `PI`
2. Client computes `PIh = hash(PI)`
3. Client sends `PIh` to processor
4. Processor generates random `R`
5. Processor computes signature `S = sign(PIh, R)` with signing key
6. Processor returns `S`, `R`, public key `PK`

ZK transformation (executed in circuit):

1. Client proves knowledge of `PI` (preimage of `PIh`)
2. Validates `S` using `PIh`, `R`, `PK`
3. Computes severed commitment: `SC = hash(PI, R)`
4. Circuit outputs: `SC`, `PK`

Unsealing phase:

- Client presents `SC` and ZK proof to processor
- Processor validates commitment exists; cannot link to original sealing interaction
- All identifying metadata has been severed

4.3 Properties

Property	Description
Deterministic	Processor controls `R`; client cannot generate multiple severed commits from one interaction
Unlinkable	`SC` is computationally unlinkable to `PIh` without knowledge of `PI`
Protocol-level privacy	Combined with network-layer anonymity (Tor/proxies), requests become indistinguishable

The severed commitment also generates the reveal value used to initiate unsealing: `RV = hash(PI, rand1)`, which must be inserted into the datastream to begin the unsealing flow.

5. Chained Proofs & Unseal Conditions

5.1 Condition Interface

Unseal conditions are on-chain contracts satisfying:

```
verify(rawBytes, publicSignals) → boolean
```

Required properties:

- Pure function (same input always produces same output)
- Deterministic (no randomness)
- Stateless (no on-chain state access during verification)
- Statically callable (external view functions only)
- Non-upgradeable (immutable logic)

Conditions are not limited to ZK proof verifiers. Any immutable on-chain contract satisfying the interface is a valid condition step. ZK verifiers and conventional contracts compose freely within the same pipeline.

Registration: Conditions must be registered in the committee-maintained registry to be eligible for slashing enforcement.

5.2 Chained Proof Pipeline

Chained proofs are a composable condition pipeline with typed signal passing between steps. The client commits to the entire pipeline at sealing time via a hash chain:

```
unseal_root = hash(
hash(hash(0, ACTION_1, params_1), ACTION_2, params_2),
ACTION_3, params_3 ...
)
```

Each action extends the root hash. Random values mixed into the construction prevent the processor from inferring condition contents from the root alone.

Core actions:

Action	Description
`PREPARE_NEXT_PROOF`	Loads verifier address, proof bytes, public signals (first invocation serves as the opening proof)
`STATIC_INPUT`	Compares public signal to hardcoded value
`PASS_SIGNAL`	Asserts equality of two public signals across steps
`VERIFY_PROOF`	Executes verification function at registered contract address
`VALIDATE_DATA_ROOT`	Validates Merkle root exists in datastream history

The pipeline supports **forking**: a step may branch based on prior output, enabling OR-logic (e.g., "valid before timestamp T OR signed by key K") within a single committed root.

5.3 Opening Proof (Mandatory)

Every unsealing flow begins with an opening proof that validates the reveal value. The reveal value **must** be inserted into the datastream before unsealing can proceed. This is not optional: it creates mandatory public observability of every unsealing attempt. The attempt is observable before decryption occurs, enabling real-time monitoring by the secret owner or any designated watcher without trusting any party to report it.

The protocol starts with a single opening proof but is not fundamentally limited to one. The current opening proof requires the reveal value to be anchored in the datastream, which enables time-based conditions to be validated against the anchored timestamp. However, when there is no timing element, a signed inclusion commitment from a datastream operator may be sufficient. The opening proof is currently implemented in Circom, but multiple proof systems can be supported simultaneously. They compose well: additional opening proof versions can be added to the on-chain registry and do not require processor updates, following the same dynamic extensibility model as all other condition modules (see Section 5.5).

Note on observability and front-running: The mandatory reveal value creates a window between public observability and decryption. For most recovery use cases, this observability is a feature - it enables monitoring and anomaly detection. However, in contexts where unsealing intent itself is sensitive, integrators should be aware that block builders and MEV searchers on Ethereum can observe reveal value insertions (particularly via the on-chain fallback path). The protocol deliberately prioritizes observability over intent privacy; use cases requiring the opposite tradeoff should consider additional application-layer protections.

5.4 Protocol-Native Condition Examples

The following conditions are available at launch. The condition library will grow over time as new verification modules are deployed to the on-chain registry. Conditions can be composed visually using the condition editor at editor.nihilium.io.

- Time-locks: `reveal_timestamp < threshold`
- Unseal delays: `current_time > reveal_time + N days`
- ZKEmail proof (email address ownership)
- ZKPassport proof (identity attribute verification)
- EdDSA/ECDSA signature verification
- Threshold signatures: `k-of-n signed(reveal_value)`
- Revocation checks: `no revocation_signal between T and T+N` (via datastream exclusion claim)
- Merkle inclusion/exclusion proofs
- Hash preimage checks
- Oracle value assertions

5.5 Dynamic Extensibility

Processors pull verification bytecode from the blockchain registry at runtime. This means:

- No preloaded verification functions required at processor deployment
- Protocol extends without processor software updates
- Teams deploy custom condition modules independently
- Compliance frameworks can standardize condition libraries

A ZK coprocessor could compile the entire chain execution to a single constant-size on-chain validity proof, making verification cost independent of chain complexity. This is not yet implemented but the architecture supports it.

6. Sealing Protocol

6.1 Pre-Sealing: Unseal Root Construction

Before interacting with a processor, the client computes `unseal_root` by dry-running the chained proof validation process with chosen conditions. This produces a cryptographic commitment to the entire unsealing procedure without revealing the conditions themselves.

6.2 Protocol

Objective: Generate a key pair where:

- The public key is available to the client for encrypting data
- The private key exists only as a homomorphic ciphertext
- No party holds the private key in plaintext at any point
- The ciphertext is cryptographically bound to: the processor, `unseal_root`, and the public key
- The client exclusively holds the sealed package

Client → Processor:

Sends `hash(unseal_root)` and `hash(PI)` (severed commitment preimage hash).

Processor actions:

1. Validates client authorization (payment)
2. Generates random `rand1` for severed commitment
3. Generates random BJJ EC private key `K1` (248-bit), derives `pk1`
4. Chunks `K1` into 8 × 31-bit pieces
5. Encrypts each chunk with ECElGamal under `pHEpk` → 8 ciphertexts `ci1`, 8 ephemeral keys `ek1`
6. Computes `ci1H = hash(ci1)`, `ek1H = hash(ek1)`
7. Computes `comH = hash(rand1, hash(ci1H, ek1H), PIh, pk1, hash(unseal_root))`
8. Signs `comH` with BJJ signing key → `comHSig`
9. Returns: `comHSig`, `ci1`, `ek1`, `rand1`

Client ZK circuit inputs:

`comHSig`, `rand1`, `pHEpk`, `ci1`, `ek1`, `unseal_root`, `PI`, `pk1`, 8 nonces

Inside ZK circuit:

- Hashes `unseal_root` and `PI` per severed commitment scheme
- Reconstructs `comH` and validates `comHSig` against processor's public key
- Generates client private key `K2`, derives `pk2`
- Computes final public key: `pkF = pk1 + pk2` (EC addition)
- Chunks `K2` into 8 × 31-bit pieces, HE-encrypts → `ci2`, `ek2`
- Homomorphically adds: `ciF = ci1 + ci2`, `ekF = ek1 + ek2`
- Generates reveal value: `RV = hash(PI, rand1)`

Mathematical foundation:

```
K1 + K2 = KF (private key addition)
pk1 + pk2 = pkF (corresponding public key addition)
```

The homomorphic addition ensures `ciF` decrypts to `KF` without `KF` existing in plaintext at any point during construction.

Circuit outputs:

`ciF`, `ekF`, `pkF`, `signVK`, `pHEpk`, `unseal_root`, `RV`

Note on processor honesty: A processor could provide `pk1` inconsistent with the encrypted `K1`. This is a slashable offense - if the processor cannot decrypt correctly at unsealing, the client can prove misbehavior on-chain. Generating a ZK proof of correct key generation is therefore redundant overhead and is not required.

6.3 Package Outputs

Private package (mandatory):

- Unseal conditions (chained proof construction)
- HE-encrypted ciphertexts (`ciF`, `ekF`)
- Static parameters and metadata
- ZK opening proof
- Encrypted data or pointer to encrypted data

Public observer package:

- Reveal value (`RV`)
- Condition metadata (time-lock thresholds, condition types - no private data)
- Purpose: enables monitoring of datastream for unsealing attempts without revealing sealed content

Slashing package (optional):

- Skimmed opening proof (commitment values only, no private data)
- Metadata for initiating slashing challenges
- Should be stored separately from private package

7. Unsealing Protocol

7.1 Preconditions

Before unsealing, the client must:

1. Insert the reveal value into the datastream - this triggers mandatory public observability
2. Collect all proofs required by the unseal conditions
3. Generate ZK proofs satisfying the chained proof pipeline

Collection time is condition-dependent: instant for time-locks; longer for ZKPassport or ZKEmail proofs.

7.2 Protocol

Client → Processor:

Sends complete proof chain and HE-encrypted ciphertexts.

Processor actions:

1. Pulls verification bytecode from blockchain registry for each condition
2. Executes chained proof validation: - Opening proof: validates reveal value and anchoring via `PREPARE_NEXT_PROOF` - Validates reveal value exists in datastream - Executes all additional conditions in committed order - Validates that executed actions reproduce the committed `unseal_root`
3. On successful validation: - Decrypts 8 ECElGamal ciphertexts (one per 31-bit chunk) - Solves discrete logarithm for each chunk (curve point → integer) - Combines 8 chunks → `tKF`
4. Returns `tKF` to client

A processor holding a valid unsealing proof is immune to slashing challenges for that specific unsealing.

Client final step:

Reconstructs `KF` from `tKF` by removing the local shielding entropy (which prevented the processor from learning the actual secret during decryption), then decrypts originally sealed data.

Note: Discrete log solving (~1 second) could be moved client-side with precomputed tables. Current implementation places this at the processor.

8. Economic Security Model

8.1 Staking Model

Processor stake is reputation capital at risk across all commitments, not per-seal collateral. A single stake secures all seals the processor has committed to. Any single provable misbehavior forfeits the entire stake permanently (one-strike-out).

Consequences:

- Economic security scales with processor reputation, not seal quantity
- Long-term honest operation maximizes lifetime earnings
- Higher stakes attract more traffic (clients prefer high-stake processors)
- Processors compete on stake size, reliability, and uptime history

8.2 Slashing: Premature Disclosure

Trigger: The private key corresponding to a commitment is revealed before unseal conditions are satisfied.

Process:

1. Challenger deposits bond (covers on-chain gas costs)
2. Processor has a time window to present a valid unsealing proof
3. If processor cannot provide proof: stake slashed, bond returned, processor permanently banned
4. If processor provides valid proof: stake retained, processor receives bond

Attribution property: The processor is the only actor capable of producing the decrypted key. If the key surfaces outside a valid unsealing flow, attribution is unambiguous. The challenge mechanism is permissionless - anyone observing a leaked key can initiate a challenge, not only the original client.

Coercion resistance (comparative). No system provides absolute protection against state-level coercion. The relevant question is: what happens to a coerced party across different architectures?

In standard threshold MPC, a coerced party hands over their share, the state collects k shares, reconstructs the secret, and nobody faces any consequence. The coercion is clean and costless for both parties.

In Nihilium, a coerced processor hands over its private key, and the coercer now holds an indefinite slashing weapon against that processor's entire staked capital. Compliance does not end the relationship - it guarantees that the coercing party holds permanent economic leverage over the processor. The processor has a strong structural incentive to resist, relocate, or litigate, because cooperation creates an ongoing vulnerability rather than a clean resolution.

This does not prevent coercion. It makes coercion structurally more expensive and operationally messier than in any comparable system. Combined with geographic and jurisdictional diversity across a threshold of processors (see Section 9.9), coordinated coercion requires simultaneously managing this dynamic across multiple independent actors in incompatible legal frameworks.

8.3 Slashing: Refusal to Serve (Forced Execution)

Trigger: Processor refuses to serve a valid unsealing request.

Process:

1. Client publishes chained proof and ciphertexts on-chain
2. Processor must respond within time limit with: ZK proof of decryption + re-encryption under client-provided key (for privacy)
3. If processor fails to respond: stake slashed, client compensated
4. If processor responds: client receives result, processor retains stake

The enforcement is symmetric: the same proof mechanism that punishes premature disclosure also compels delivery. A processor faces binary ruin for either violation.

This is a significant advantage over MPC-based systems, where node operators can refuse requests for any reason with no recourse mechanism. The forced execution path means that the chain acts as a backstop for the adversarial case - processors cooperate because refusal is economically equivalent to premature disclosure.

8.4 Grace Periods & Exit

- Processor sets grace period at stake deposit
- Grace period changes apply only after current period expires
- Intent to unstake must be signaled publicly on-chain
- During grace period, processor must continue serving all committed requests
- Longer grace periods signal stronger commitment; attract more users

8.5 Payment Model

Sealing: Prepaid. Client pays processor and datastream upon commitment. Once issued, commitments are irrevocable. At launch, Nihilium will provide a marketplace where processors can sell capacity to developers directly without requiring separate commercial agreements, facilitated by payment channels. However, the protocol does not force use of the marketplace. Direct API key arrangements with individual processors, backed by paper agreements or any other commercial terms, are equally supported.

Unsealing: Free. No additional payment. No marketplace involvement. Processor committed at sealing time. This eliminates pay-to-unseal censorship and extraction vectors.

8.6 Tokenless Design

The protocol operates without a native token by explicit design. Token-based staking ties security to market perception: a single incident can cause price collapse, which reduces staking incentives, which weakens security. Nihilium accepts only native ETH or ERC-20 tokens without blacklisting or whitelisting capabilities as stake assets. Stablecoins are explicitly excluded: their issuers can blacklist the staking contract at any time, which would create external leverage over the protocol's development and undermine the neutrality of the enforcement layer. Only assets whose transfer logic cannot be selectively frozen or censored by a third party are considered suitable for staking. Governance token consideration remains open but is not required for protocol operation.

8.7 Private Deployments

Entities requiring private governance (banks, insurers, regulated industries) may deploy isolated instances with internal dispute resolution replacing public slashing. Cryptographic guarantees are identical; governance model differs.

9. Threshold Recovery

9.1 Single-Processor Risk

The base protocol (one seal, one processor) has liveness risks: processor offline, slashed for unrelated misbehavior, or operationally compromised. A threshold construction is required so that any sufficient subset of processors can enable recovery, without introducing new attack vectors.

9.2 Why Not Shamir Secret Sharing

The naive approach to threshold recovery is Shamir Secret Sharing (SSS): split an AES key into shares, seal each share independently with a different processor, and reconstruct from any k-of-n shares. This introduces a critical vulnerability: the **decryption oracle attack**.

In the SSS construction, each share is encrypted under a combined public key (`pkF_i`) produced by the sealing protocol. The corresponding private key exists only inside the processor's HE ciphertext. To recover a share, the processor must decrypt the HE ciphertext and return the private key - a slashable action. However, the client can circumvent this: instead of requesting the private key, the client sends the encrypted share alongside the HE-encrypted private key and asks the processor to decrypt the share directly. The processor returns the plaintext share without ever revealing the private key. No slashing is triggered because the private key was never disclosed - it was used transiently as a decryption oracle inside the processor's environment.

An attacker exploiting this can approach k processors independently, obtain k plaintext shares through oracle queries, and reconstruct the secret. Each processor performed a legitimate-looking decryption operation. No private key was leaked. No slashing condition was met.

This attack is fundamental to any construction where separable shares are encrypted under individual keys: the processor can always be used as a decryption oracle for the share without revealing the key itself. The protocol's security guarantee - that the only path to decryption is full private key disclosure - is violated.

9.3 Combinatorial Threshold Encryption

The protocol uses Combinatorial Threshold Encryption to eliminate the oracle attack entirely. Instead of splitting a secret into shares encrypted under individual keys, the secret is encrypted under all possible threshold-qualifying combinations of public keys. Decryption requires the summed private key of a qualifying combination, which can only be obtained by full disclosure of each component private key.

Why this construction exists: Each sealing interaction is a 1-on-1 exchange between client and processor, producing an independently generated key pair. There is no DKG ceremony, no interactive setup, no communication between processors. The combinatorial construction achieves threshold recovery from independently generated keys - which is a fundamentally different setup than standard threshold cryptography, where all parties must coordinate to produce a shared key. The C(n,k) cost is the price of non-interactive threshold recovery, not an inefficiency.

Construction (k-of-n threshold, example: 5-of-10):

1. Client completes the sealing protocol with n = 10 independent processors, obtaining 10 combined public keys: `pkF_1, pkF_2, ..., pkF_10`
2. Client generates a random symmetric key `S`
3. Client enumerates all C(n,k) = C(10,5) = 252 combinations of 5 public keys
4. For each combination `{i,j,k,l,m}`, client computes the combined public key: `PK_combo = pkF_i + pkF_j + pkF_k + pkF_l + pkF_m` (EC point addition)
5. Client selects a single random ephemeral scalar `r` and computes `C1 = r · G`
6. For each combination, client computes the shared secret `r · PK_combo`, derives a symmetric key via KDF, and encrypts `S` under it
7. Client encrypts the payload under `S`, then discards `S` and `r`

The sealed package contains `C1`, the 252 encrypted copies of `S`, and the AES-encrypted payload.

Recovery:

1. Client satisfies unseal conditions for at least 5 processors
2. Each processor returns its private key component (slashable action)
3. Client sums the 5 private key components to obtain the combined private scalar for one of the 252 combinations
4. Client computes the shared secret using `sk_combined · C1`, derives the symmetric key, and decrypts `S`
5. Client decrypts the payload with `S`

Why this eliminates the oracle attack: There are no separable shares. The symmetric key `S` is encrypted directly under combined public keys. To decrypt any single copy of `S`, the client needs the summed private scalar of 5 processors. Obtaining that sum requires each processor to disclose its individual private key - a slashable action. There is no intermediate computation a processor can perform that yields useful information without full key disclosure.

Why this enables clean slashing boundaries: In the combinatorial construction, the processor never produces partial decryptions during legitimate operation - it either reveals the full private key after validating the proof chain, or it doesn't. Any partial decryption appearing anywhere is inherently unauthorized. In a standard threshold system with DKG, producing partial decryptions *is* normal operation, making it difficult to distinguish authorized from unauthorized decryptions on-chain.

9.4 Resistance to Processor Collusion via MPC

The combinatorial construction prevents the client from using processors as decryption oracles. A separate attack vector remains: colluding processors could attempt to decrypt cooperatively using Multi-Party Computation (MPC), avoiding individual key disclosure.

9.4.1 Simple MPC (Partial Decryptions)

In standard threshold EC-ElGamal decryption, each party computes a partial decryption `D_i = sk_i · C1` and shares it. The partial decryptions are combined to recover the shared secret. No individual private key is revealed.

However, each partial decryption `D_i` is cryptographically attributable to the corresponding public key `pkF_i` via discrete log equality (DLEQ): the relationship `dlog_G(pkF_i) = dlog_{C1}(D_i)` holds, and a DLEQ proof can be verified publicly. Publishing `D_i` together with the known `C1` and `pkF_i` constitutes the same quality of slashing evidence as publishing the private key itself - it is a value that only the holder of `sk_i` could have produced.

Furthermore, colluding processors require DLEQ proofs from each other to verify the correctness of contributed partial decryptions. Without these proofs, any participant could contribute garbage values, causing decryption to fail with no way to identify who cheated. The proof material necessary for reliable collusion is therefore the same material that enables tracing and slashing.

A single self-interested participant in the colluding group can capture another participant's partial decryption and submit it as slashing evidence - claiming the slashing reward and eliminating a competitor. The on-chain slashing contract verifies the DLEQ relationship and slashes the identified processor.

9.4.2 Advanced MPC (Garbled Circuits / SPDZ)

Sophisticated colluders could attempt to use generic MPC protocols (garbled circuits, SPDZ) where the entire decryption is computed on encrypted inputs. In this model, each processor feeds its private key into the MPC as an encrypted input, and only the final plaintext emerges - no partial decryptions or intermediate values are produced.

This eliminates the DLEQ tracing vector. However, it introduces a different deterrent: each colluding processor must feed its actual private key - the material whose disclosure triggers permanent stake forfeiture - into software provided by or coordinated with the party attempting to break the protocol. The processor must trust that:

- The software genuinely implements the claimed MPC protocol and does not exfiltrate the key
- No backdoors exist that leak intermediate values or the key itself
- Other participants are not running modified versions that capture inputs
- The implementation has no bugs that accidentally produce attributable evidence

Auditing MPC implementations for the absence of key exfiltration is a non-trivial task even for experienced cryptographers. A rational processor is being asked to stake its entire economic position on the correctness and honesty of unauditable software provided by an adversary. The incentive structure is inverted: the collusion scheme demands more trust among colluders than the honest protocol demands of anyone.

9.4.3 Collusion Instability

The combination of these two layers - DLEQ-based slashing for simple MPC, and the unauditable-trust problem for advanced MPC - produces a structural instability in any collusion attempt: defection by a single participant is individually rational and sufficient to break the collusion.

Against simple MPC, any participant can capture another's partial decryption and submit it as slashing evidence - collecting the slashing reward while destroying the colluder's stake. Against advanced MPC, any participant can refuse to join (preventing the collusion from reaching threshold) or join and defect (reporting the attempt, potentially with key-derived proof of participation).

The collusion must therefore maintain unanimous cooperation among all k participants, where every participant has a direct financial incentive to betray the group. This is a strong coordination requirement. Standard threshold cryptography requires that at most k-1 parties out of n are compromised - a trust assumption about the *majority*. Nihilium's threshold construction requires only that the collusion cannot maintain unanimous internal loyalty - a stability assumption about the *weakest link*. Whether any individual participant will defect is not a question of honesty but of rational self-interest: the slashing reward for betrayal is immediate and guaranteed, while the proceeds of collusion must be shared and depend on sustained mutual trust among parties who are, by definition, already acting dishonestly.

9.5 Availability, Security, and Longevity Tradeoffs

The security of the construction is determined by the threshold k, not the pool size n. An attacker must always compromise k processors regardless of how many total processors exist. Increasing n increases availability (more possible qualifying subsets) without changing the cryptographic difficulty of the attack.

This availability property is fundamentally stronger than what classical threshold schemes offer. In standard SSS or DKG-based threshold decryption, adding participants requires re-sharing the secret or running a new distributed key generation ceremony - an interactive, coordinated process. In Combinatorial Threshold Encryption, the client unilaterally adds processors at sealing time. A 3-of-20 configuration provides C(20,3) = 1,140 possible recovery paths, compared to C(6,3) = 20 for a 3-of-6 setup, with no additional coordination, no re-sharing protocol, and no interaction between processors.

For secrets intended to remain sealed over long time horizons - years or decades - this is a significant advantage. Processor liveness over extended periods is uncertain: operators may go offline, be slashed for unrelated misbehavior, cease operations, or lose infrastructure. A large n with a modest k ensures that recovery remains viable even as individual processors drop out of the network over time.

However, increasing n does increase the social attack surface: more processors exist as potential targets for coercion, bribery, or compromise. The probability of finding k compromisable processors grows with the pool size. The slashing mechanism mitigates this - the economic cost of compromising k processors scales linearly with k × stake - but the risk profile differs between, for example, 3-of-6 and 3-of-20. Geographic and jurisdictional diversification of the processor set is the primary mitigation for this risk (see Section 9.9).

A successful attack requires the conjunction of three conditions: the attacker obtains the sealed package, the attacker compromises k processors, and - when value decomposition is applied correctly - the attacker independently obtains the valuable counterpart that gives the sealed component its worth. The sealed package alone contains only an encrypted key or password; the encrypted data it unlocks must be obtained separately. A password without its encrypted file has zero standalone value. An encrypted file without its password has zero standalone value. The attacker must breach all three barriers simultaneously: acquire the sealed package, acquire the encrypted data, and compromise k processors willing to accept slashing.

9.6 Practical Parameters

Storage and computation: Reusing the same ephemeral scalar r across all C(n,k) combinations, each combination requires only one encrypted symmetric key (one field element). The computation per combination is one EC scalar multiplication plus symmetric key derivation and encryption.

Configuration	Combinations C(n,k)	Approx. Storage	Approx. Computation
2-of-5	10	< 1 KB	< 1 second
3-of-10	120	~4 KB	~2 seconds
5-of-10	252	~8 KB	~5 seconds
3-of-20	1,140	~36 KB	~20 seconds
5-of-15	3,003	~94 KB	~50 seconds
10-of-20	184,756	~5.8 MB	~5 minutes

Configurations beyond C(n,k) ≈ 200,000 become impractical. For the floor-level use cases the protocol targets (wallet recovery, password management), k ≤ 5 with n ≤ 20 provides a practical operating range. Use cases requiring larger committees should evaluate whether the combinatorial ceiling is acceptable or whether alternative threshold mechanisms are more appropriate.

Scalar field safety: The protocol uses 248-bit private scalars on Baby Jubjub (scalar field order ≈ 2^251). Summing k scalars of 248 bits each produces a maximum value of k × 2^248. For k = 5, this is approximately 2^250.3, which remains within the field order. For k ≥ 7, overflow risk increases and either smaller per-scalar bit sizes or modular reduction should be used.

9.7 Combinatorial Scaling: Network Complexity vs. Compute and Storage

The C(n,k) combinatorial blowup is the most visible cost of the threshold construction. At 5-of-10 (252 combinations) the cost is trivial. At 10-of-20 (184,756 combinations) it is manageable. Beyond roughly C(n,k) ≈ 200,000, client-side computation and sealed package size become impractical. This ceiling is real and defines the protocol's operational boundary for threshold parameters.

However, two properties make this ceiling less constraining in practice than it initially appears.

The computation is embarrassingly parallel. Each combination's encryption - one EC scalar multiplication, one KDF derivation, one symmetric encryption - is independent of every other combination. The workload distributes linearly across available cores with no synchronization, no shared state, and no communication between threads. A modern 8-core mobile processor handles all 252 combinations of a 5-of-10 configuration in under a second. A 16-core desktop handles 3,003 combinations (5-of-15) in under a minute. WebWorker-based parallelism in browser environments makes this accessible even for web applications. The computation is a one-time cost at sealing time, not a recurring overhead.

Higher parameters correspond to higher-value use cases where the cost is proportionate. A 10-of-20 configuration producing 184,756 combinations and a ~5.8 MB sealed package is not a consumer wallet recovery setup. It is an institutional custody configuration protecting assets where five minutes of client-side computation and a few megabytes of storage are negligible relative to the value secured. The parameter ceiling binds at exactly the boundary where compute and storage costs stop mattering. For the floor-level consumer use cases the protocol targets (2-of-5, 3-of-10), the combinatorial cost is imperceptible.

DKG/MPC systems face an analogous scaling constraint, but in a less reliable dimension. Distributed key generation protocols require $O(n^2)$ network messages during the ceremony - every participant must communicate with every other participant, exchanging commitments, shares, and verification proofs. A DKG ceremony with 20 participants requires on the order of 400 pairwise communications, each of which must complete successfully within timeout windows. This is quadratic scaling in network overhead: the dimension most vulnerable to latency, packet loss, NAT traversal failures, firewall restrictions, and participant availability. A single unresponsive participant can stall or abort an entire ceremony, requiring restart.

Re-sharing protocols for operator rotation carry the same $O(n^2)$ network cost and the same fragility. Adding a single participant to a DKG-based threshold system requires a coordinated ceremony involving all existing participants - every one of whom must be online, reachable, and responsive simultaneously.

Nihilium's combinatorial construction trades network complexity for local compute and storage. The C(n,k) blowup is entirely client-side: deterministic, parallelizable, performed once, and dependent on no external parties. There is no network round, no coordination, no timeout, no possibility of a stalled ceremony. The client computes independently and the result is correct by construction.

This is a fundamentally more reliable scaling dimension. Compute and storage are predictable, locally controlled, and monotonically improving with hardware generations. Network coordination across independent parties is unpredictable, subject to external failures, and does not improve with hardware. A DKG ceremony that fails at step 18 of 20 due to a network timeout wastes all prior computation. A combinatorial encryption that completes 18 of 20 processor seals has 18 usable seals regardless of what happens to the remaining two.

Deployed systems confirm that DKG's scaling constraint is the binding one, not Nihilium's. The combinatorial ceiling is sometimes perceived as a limitation unique to this construction. In practice, no deployed threshold system operates at parameters anywhere near Nihilium's ceiling. The largest actively deployed DKG-based threshold cohorts are:

System	Total nodes (n)	Threshold (k)	Architecture
Largest deployed DKG + TEE system	7–8	5–6	DKG + TEE
Largest deployed DKG-only system	7 (target 21)	~5 (target ~14)	DKG
Institutional MPC custody	3–5	2–3	MPC-CMP + TEE
Consumer 2-party MPC	2	2	2-party MPC

The most decentralised actively deployed threshold system in production operates with 8 nodes. Its developers explicitly identified the DKG scaling wall as a fundamental constraint, developing a splicing mechanism to create multiple independent small cohorts rather than growing a single threshold group - because DKG communication overhead increases quadratically with cohort size, inversely impacting throughput.

Nihilium's comfortable operating range of $k \leq 5$, $n \leq 20$ already exceeds the threshold parameters of every deployed MPC system. A 3-of-10 Nihilium configuration provides 120 recovery paths across 10 independent processors - more redundancy and a larger operator set than any production DKG deployment. A 5-of-15 configuration (3,003 combinations, ~50 seconds client-side computation) operates in a regime that DKG-based systems have not attempted in production. The combinatorial ceiling of ~200,000 combinations sits well above the practical ceiling that DKG coordination overhead imposes on real-world deployments.

The tradeoff favours Nihilium for any threshold parameter within the practical ceiling, and both ceilings - combinatorial and DKG - are ultimately defined by the same underlying constraint: the quadratic cost of increasing participant count. The difference is where that cost falls. DKG pays it in network coordination (unreliable, synchronous, failure-prone). Nihilium pays it in local computation (reliable, parallelizable, deterministic). For any parameter range that real-world systems actually deploy at, Nihilium's approach is strictly more practical.

9.8 Comparison to Social Recovery

This construction produces social recovery where the guardians are contracts, not contacts. Traditional social recovery distributes trust across human guardians who may be unavailable, uncooperative, or colluding. Nihilium replaces human guardians with economically staked independent operators. The orchestration burden - unavailable guardians, lost contact - is replaced by a threshold of independent actors, any subset of whom can serve the request. Recovery is a proof, not a conversation.

Property	Traditional Social Recovery	Nihilium Threshold
Trust basis	Social relationship	Economic stake
Coordination required	Yes (manual)	No (client proves)
Collusion resistance	Low	High (slashing + collusion instability)
Oracle resistance	N/A	Combinatorial encryption eliminates oracles
Guardian rotation	Difficult	Re-seal with new processors
Recovery attempt privacy	None (guardians aware)	Severed commitments

9.9 Counterparty Selection and Geographic Arbitrage

Nihilium's architecture gives the client unilateral control over processor selection. The client independently chooses which processors to seal with, how many to include, and what threshold to set. There is no cohort to join, no committee to petition, and no network-determined assignment. This is a structural property of the non-interactive threshold construction: because processors operate independently and never coordinate with each other, the client's selection is unconstrained.

This contrasts with how counterparty selection works in DKG-based MPC systems. In a DKG system, the cohort is determined by the network - nodes participate in a DKG ceremony together, and the resulting threshold key is bound to that specific set. A client using a deployed MPC implementation does not choose which nodes protect their key. The network has one active cohort (or a small number of realms), and all clients use the same set. In theory, a client could choose between competing MPC networks, but in practice the options are few, cohort composition is determined by staking contests or governance processes, and switching requires re-encryption. The selection is non-elastic.

This freedom of selection enables geographic arbitrage as a coercion resistance strategy - if processors are operated across multiple legal jurisdictions with incompatible legal frameworks. A 3-of-10 threshold across five jurisdictions makes coordinated legal compulsion more difficult because cross-border enforcement requires mutual legal assistance treaties (MLAT), which are slow and subject to bilateral diplomatic considerations. This is a possibility that the protocol's architecture enables but that depends on processors actually being operated in diverse jurisdictions.

Different use cases can demand different processor sets. A consumer wallet recovery seal might prioritise low-cost, high-availability processors. A corporate compliance seal might select high-stake processors with long grace periods. An inheritance seal might prioritise jurisdictional diversity and organisational stability over price. The client makes these choices at seal time, per seal, with no negotiation and no dependency on network governance decisions. In shared-cohort MPC systems, this kind of per-client risk tuning is structurally impossible - all clients inherit the same node operators and the same jurisdictional profile.

9.10 Relationship to Prior Work

Boneh, Partap, and Rotem (CRYPTO 2024) address the core accountability problem in threshold decryption - that colluding parties can construct a decoder that cannot be traced to any individual participant - using threshold traitor tracing. Their construction adds post-hoc forensic tracing: if a decoder is produced, a tracing algorithm can identify at least one contributor by feeding malformed ciphertexts and observing the response pattern.

The Nihilium construction differs in approach. Rather than detecting misbehavior after the fact, it prevents the formation of a useful decoder by construction. There is no partial decryption operation - the only path to decryption is full private key disclosure, which is immediately and automatically slashable. The traitor tracing approach is more general (it handles arbitrarily obfuscated decoders), while the combinatorial approach is simpler and preventive, at the cost of the C(n,k) combinatorial ceiling on practical parameters.

Neither approach addresses full-scale state coercion where the coercer compels disclosure and uses the material off-chain. BPR's traitor tracing identifies the coerced party after the fact, but a state actor operating under legal authority does not care about being identified - the tracing result is forensically interesting and practically irrelevant to the coercer. Nihilium's permanent leverage property creates structural resistance to coercion without solving it.

Cassiopeia (FC Workshops 2023) implements a related concept: on-chain witness encryption using a trusted committee, PVSS for secret splitting, zkSNARK proofs, and economic slashing for misbehavior. The architectural parallel is notable. However, Cassiopeia requires on-chain transactions as part of normal operation (every seal and unseal involves gas costs and on-chain state), depends on committee coordination for unsealing (introducing liveness risk), and creates permanent on-chain metadata trails. Nihilium's off-chain-by-default architecture, independent processors, client-side proof burden, and severed commitment privacy represent fundamental design differences that enable consumer-scale deployment rather than low-frequency high-value operations. The systems occupy different points in the design space: Cassiopeia demonstrates that crypto-economic witness encryption approximation is feasible on-chain; Nihilium attempts to make it deployable at scale.

10. Use Cases

The following examples illustrate the protocol pattern for various recovery and conditional access scenarios. Nihilium is designed for low-friction use cases where the alternative is typically no recovery mechanism at all. For high-value assets requiring institutional-grade protection, dedicated custody infrastructure should be used.

10.1 Wallet Key Recovery ("Forgot Password")

Setup: Wallet encrypts seed phrase with a strong random password. Password is sealed under ZKEmail or ZKPassport condition. Encrypted seed phrase stored locally on user's device.

Recovery: User proves email ownership or passport identity → Nihilium unseals password → password decrypts seed phrase.

Value decomposition applies: the sealed password has zero standalone value; the encrypted seed phrase on the user's device has zero standalone value. Their combination is the full wallet. The two components exist in independent security domains.

10.2 Password Manager Master Key

Seal master password under identity proof condition. Password manager stores encrypted vault locally. Recovery: user proves identity → unseals master key.

10.3 Identity-Verified Document Delivery

Seal a document encryption key under recipient identity condition (ZKPassport attribute). Sender cannot access after sealing. Recipient proves identity → unseals key → decrypts document. Every delivery attempt is publicly observable via datastream.

10.4 Healthcare: Emergency Access with Audit Trail

Seal medical record decryption key under: `practitioner_license_proof AND emergency_department_signature AND NOT revocation_signal`.

Patient can monitor all unsealing attempts via public observer package. Every access is auditable without trusting any third party to report it.

Note: Healthcare use cases may involve high-value data and sophisticated adversaries (institutional, regulatory). Integrators should assess whether the floor-level security model is appropriate for their specific compliance and threat environment, or whether additional protections are warranted.

10.5 Compliance: Break-Glass Procedures

Seal critical keys under: `k-of-n executive signatures AND NOT legal_hold_signal AND time_delay`.

Access is cryptographically gated rather than policy-gated. Every attempt is on-chain observable. Slashing prevents executives from bypassing conditions through collusion.

Note: As with healthcare, break-glass procedures may govern access to high-value or regulated assets. The protocol provides the pattern; integrators must evaluate the threat model.

10.6 Dead Man's Switch with Revocation

Seal data under: `time_delay AND NOT revocation_signal_in_window`.

Owner inserts a revocation signal periodically to prevent release. Failure to insert (e.g., incapacitation) allows unsealing after the window. Revocation window is enforced via datastream exclusion claim, which is stake-backed and verifiable.

10.7 Regulatory Viewing Keys with Verifiable Non-Access

Seal a viewing key (read-only decryption key for encrypted data) under: `court_order_proof AND regulatory_authority_signature AND time_delay`.

Regulator can unseal the viewing key by satisfying conditions. The mandatory reveal value insertion means every access attempt is publicly observable before decryption occurs, so the private party can independently verify that no unsealing has ever been initiated.

This inverts the typical transparency model: instead of trusting an escrow agent not to inspect the key, the private party monitors the datastream for access attempts. The regulator retains a credible access path. Neither side depends on the other's honesty.

11. Performance

All figures from NodeJS reference implementation. Rust implementation expected to improve significantly.

11.1 Measured Latencies

Operation	Environment	Latency
Sealing ZK circuit	Desktop WASM	2.8s average
Chained proof construction	Desktop WASM	0.3s per module
Total sealing (client)	Desktop WASM	3–15s
Commitment generation	16-core server	7ms
HE encryption (8 chunks)	16-core server	3ms
Commitment signing	16-core server	1ms
Total sealing (processor)	16-core server	~11ms
Chained proof verification	16-core server	800ms
HE decryption (8 chunks)	16-core server	100ms
Discrete log solving (8 chunks)	16-core server	1200ms
Total unsealing (processor)	16-core server	~2.1s
Datastream update	On-chain	15–30s (block-dependent)

11.2 On-Chain Costs

Operation	Gas	Notes
Datastream update	~69,500	24-depth tree, 60-block history
ZK proof verification	~250,000	Per Groth16 proof (Solidity verifier)
Slashing challenge	Variable	Bond covers gas; awarded to challenger on success

12. Cryptographic Primitives

Primitive	Specification	Notes
ZK proof system	Groth16	Requires trusted setup (Phase 1 + Phase 2 MPC ceremonies); architecture supports migration to PLONK/STARK
ZK circuit framework	Circom	
Elliptic curve	bn254 (BN128) for ZK; Baby Jubjub (BJJ) for HE and signatures	~100-bit security; migration to BLS12-381 (128-bit) straightforward
HE scheme	ECElGamal on Baby Jubjub	CPA-secure; 31-bit chunk BSGS decryption
HE chunk size	31-bit chunks; 8 chunks per 248-bit key	
Signature scheme	EdDSA on Baby Jubjub	
Merkle hash function	Keccak256 (EVM-native)	
Merkle tree depth	24 (configurable for sub-trees)	
ZK proof size	~200 bytes	
Merkle proof size	~768 bytes (24-depth tree)	
Signature size	64 bytes	

13. Comparison to Existing Systems

13.1 Nihilium vs. Threshold MPC Systems

Property	Nihilium	Threshold MPC
Collusion detection	DLEQ-attributable partial decryptions; single defector breaks collusion	Undetectable; no on-chain evidence
Collusion punishment	Permanent total stake loss	None (cannot attribute)
Liveness enforcement	Forced execution via chain; refusal = slashing	No enforcement; nodes can refuse for any reason
Condition extensibility	Client-side proofs; processors pull verifier bytecode at runtime; infinitely extensible	Server-side evaluation; requires node network support
Operator coordination	None; independent 1-on-1 sealing	DKG ceremony required; re-sharing for membership changes
Hardware dependency	None (TEEs optional)	SGX/SEV required for key share isolation
On-chain footprint (normal)	Datastream update only, amortized across all users	Varies by implementation
Privacy	Severed commitments; unlinkable sealing/unsealing	Session linkability; on-chain metadata trails
Single-party compromise	Sealed material encrypted under combined threshold keys; individual key useless for recovery	Key share information-theoretically useless
Blockchain dependency	Required for enforcement; without it, degrades to standard MPC trust assumptions	None (but this means no enforcement exists at all)
Trust model	Economic: collusion requires unanimous internal loyalty	Cryptographic: k-1 honest out of n required

Key insight: MPC systems operate permanently at the trust level that Nihilium falls to only under catastrophic chain failure. Nihilium's normal operating state provides enforcement that MPC systems lack entirely.

13.2 Nihilium vs. Custodial Recovery

Property	Nihilium	Custodial Recovery
Counterparty dependency	Processor set (threshold, replaceable)	Specific provider (single point of failure)
Terms of service	Cryptographic commitment, irrevocable	Contractual, changeable
Censorship resistance	Forced execution; refusal = slashing	Provider can deny service
Provider shutdown risk	Threshold survivability across processor set	Complete loss of access
Regulatory seizure	Geographic arbitrage across jurisdictions	Single-jurisdiction vulnerability

13.3 Nihilium vs. Social Recovery

See Section 9.8.

13.4 Nihilium vs. Witness Encryption

General-purpose witness encryption - binding a secret to an NP condition with purely cryptographic enforcement - remains impractical for deployment. The original multilinear map construction is broken, and candidate lattice-based constructions (Tsabary, CRYPTO 2022) are not yet deployable.

Special-purpose witness encryption is increasingly practical for narrow condition types (e.g., Garg, Kolonelos, Policharla, and Wang (CRYPTO 2024) for threshold BLS signatures). However, these handle fixed condition types, not the arbitrary extensible conditions Nihilium supports. Witness encryption also does not enforce liveness - it guarantees decryption *capability* with a valid witness but not that any party will *help* decrypt.

Nihilium is designed so that the processor role could be replaced by a deployable witness encryption construction if one becomes available (see Section 14.4).

14. Open Questions

The following represent unresolved problems relevant to protocol security and design:

14.1 Detectability strengthening. The slashing guarantee requires that disclosed key material eventually surfaces in a context where it can be used to initiate a challenge. If a state actor compels disclosure and the material is never used publicly, slashing does not occur. This is an inherent boundary of economic enforcement - no system using economic deterrence can punish undetectable misbehavior. Whether this assumption can be strengthened cryptographically - for example, through commitment schemes that make private use detectable - is an open problem. Geographic arbitrage across processor jurisdictions (Section 9.9) can mitigate this if processors are operated in diverse jurisdictions.

14.2 Formal security definition. No formal security definition currently exists for slashable conditional key release as a standalone primitive, distinct from witness encryption and MPC. A formal treatment would clarify the precise threat model, the conditions under which economic guarantees hold, and the relationship to existing primitives. This is a documentation maturity gap, not a design limitation - the construction is intended to be formally analyzed as a companion effort.

14.3 ZK coprocessor compilation. The chained proof pipeline currently executes off-chain at the processor, with on-chain contracts serving only as the challenge layer. Compiling the full chain execution to a single constant-size ZK validity proof via a ZK coprocessor would make on-chain verification cost independent of chain complexity while preserving composability. The feasibility and performance characteristics of this compilation path are not yet established.

14.4 Processor replacement. The architecture is designed so that the processor role could be replaced by a cryptographic primitive - specifically, a deployable witness encryption construction - if one becomes available. The conditions under which this substitution is sound, and what interface such a primitive would need to satisfy, are not formally specified.

15. Post-Quantum Migration Path

The protocol's cryptographic operations are structurally simple relative to the broader landscape of applied cryptography. The core algebraic operations reduce to: EC point additions (key combination, homomorphic ciphertext addition, threshold public key construction), EC scalar multiplications (ElGamal encryption, threshold shared secret derivation, EdDSA signatures), and hash evaluations (commitments, Merkle trees). No pairings, multilinear maps, or lattice-hard operations are used in the current construction.

This operational simplicity creates a natural migration path to post-quantum security. Each core operation has a direct lattice-based equivalent:

Additively homomorphic encryption. The two-party key construction - where the client and processor independently encrypt key components and homomorphically add the ciphertexts (`ciF = ci1 + ci2`) such that the result decrypts to the sum of plaintexts - is the central cryptographic mechanism of the protocol. LWE-based encryption is natively additively homomorphic: adding two LWE ciphertexts produces a valid encryption of the sum of the underlying plaintexts, with noise growth bounded linearly. This is a direct structural match. The Regev cryptosystem and its ring variants (RLWE) support this property without modification, and lattice-based additively homomorphic encryption has been demonstrated in practical systems including privacy-preserving machine learning and electronic voting.

Signatures. EdDSA on Baby Jubjub can be replaced by ML-DSA (Dilithium), which is a NIST-standardized lattice-based signature scheme. The protocol uses signatures only for commitment binding and processor authentication, with no algebraic dependence on the signature's internal structure.

Zero-knowledge proofs. The protocol architecture is already ZK-system-agnostic (see Section 1.2). Groth16 can be replaced by any proof system capable of proving the required relations. STARKs, which rely on hash-based commitments rather than elliptic curve assumptions, are already post-quantum under conservative assumptions and are deployed in production systems. Lattice-based ZK proof systems are maturing in parallel: LatticeFold provides lattice-based folding for recursive proof composition, the LaZer/LaBRADOR library offers practical lattice-based succinct proofs with a usable API, and recent work on lattice commitment schemes preserves the additive homomorphism needed to prove linear relations between hidden values. Migration to STARKs provides immediate post-quantum ZK capability at the cost of larger proof sizes; lattice-based ZK systems offer a path to more compact post-quantum proofs as the field matures.

Threshold construction. The combinatorial threshold construction (Section 9.3) uses EC point addition to combine public keys and ECDH-style shared secret derivation for symmetric key encapsulation. Both operations have lattice analogues: public key combination maps to vector or polynomial addition in the lattice setting, and key encapsulation is supported by ML-KEM (Kyber), also NIST-standardized.

Collusion attribution. The DLEQ-based collusion detection mechanism (Section 9.4) relies on the discrete logarithm structure of elliptic curves to cryptographically attribute partial decryptions to specific processors. Lattices lack a direct DLEQ equivalent, but recent lattice threshold cryptography (2024-2025) achieves the same identifiable abort property through protocol structure rather than discrete-log algebra. In LWE-based threshold decryption, partial decryptions are linear functions of the processor's short secret share. Each processor's public commitment (published during sealing) binds it to its share, and any leaked partial is verifiable against that commitment using lightweight algebraic checks or correlation-based detection - no full ZKPoK or NIZK is required. A challenger submits the leaked partial together with the relevant public commitment to the on-chain slashing contract, which runs the verification and slashes the identified processor. This provides the same quality of attributable evidence as DLEQ in the elliptic curve construction. For advanced MPC collusion (garbled circuits, SPDZ), the economic instability arguments in Section 9.4.2 and 9.4.3 apply unchanged and are strengthened: any leaked intermediate from the MPC execution can be

accompanied by the same attribution check, tying it back to a specific processor's published key share.

Assessment. The protocol does not depend on any cryptographic structure that is intrinsically bound to elliptic curves or the discrete logarithm problem. The core construction - homomorphic two-party key generation, client-side zero-knowledge proofs, and economic enforcement via on-chain slashing - transfers to a lattice-based instantiation without architectural changes. Collusion attribution, initially identified as an open gap, is addressed by lattice identifiable abort techniques that are already published and used in related threshold constructions. The required lattice primitives (additively homomorphic encryption, signatures, key encapsulation) are NIST-standardized, and post-quantum ZK systems are deployable today via STARKs. A lattice-native instantiation represents an engineering effort, not a cryptographic open problem.

Nihilium Protocol · nihilium.io · contact@nihilium.io · x.com/nihiliumio

Document Version: 0.5 | License: CC-BY-4.0