NIHILIUM

Web3 ‘forget your password’ made possible
A new access-control primitive

Summary

Nihilium is a cryptographic protocol for universal, uncensorable key recovery — the Web3 equivalent of “forgot my password.”

Today, private key recovery is one of the biggest hurdles to mass adoption of self-custody and privacy-preserving applications [1][2][3]. Existing approaches suffer from three fatal flaws:

- **Context-locked**: Recovery is tied to a single wallet, blockchain, or service. Your Ethereum wallet recovery doesn’t help you access an encrypted backup, or vice versa.

- **Orchestration burden**: Solutions like social recovery or seed phrases require users to coordinate backups or validators manually. In practice, this is where the UX problems originate from.

- **Censorship-prone**: Custodial recovery, KYC checks, or simply service denial introduce points where access can be denied. Under regulatory or geopolitical pressure, this risk becomes critical.

Nihilium solves all three problems. It uses cryptographic commitments and game-theoretic incentives — the same mechanisms that secure blockchains — to ensure recovery is context-agnostic, user-friendly, and resistant to censorship.

By removing these barriers, Nihilium makes **self-sovereign, UX-friendly key recovery** possible for the first time.

The core concept

Sealed Packages

At the core of Nihilium is the **sealed package**: a public key that encrypts a secret, paired with a private key that no party has ever seen. Recovering the secret — *unsealing* — is only possible when the requesting party proves that specific conditions are met.

Because processors/nodes cryptographically commit to these conditions, they are bound to execute when valid proofs are provided. If a private key leaks prematurely, or a processor refuses to act, it triggers a **slashable event** — provable misbehavior that destroys their stake. Crucially, even a single processor is enough: the rules enforce censorship resistance regardless of network size.

Enforcement Layer

Enforcement lives on-chain, but it is only invoked when something goes wrong. In normal operation, processors and clients never need to touch the blockchain. The only constant transaction requirement is the public anchoring of an *unseal initiation* — a timestamped marker that makes every recovery attempt observable. As long as processors behave honestly, the protocol runs entirely off-chain and can scale indefinitely. On-chain execution is the *last line of defense* if a slashable event occurs or when a processor refuses to execute.

Unseal Conditions

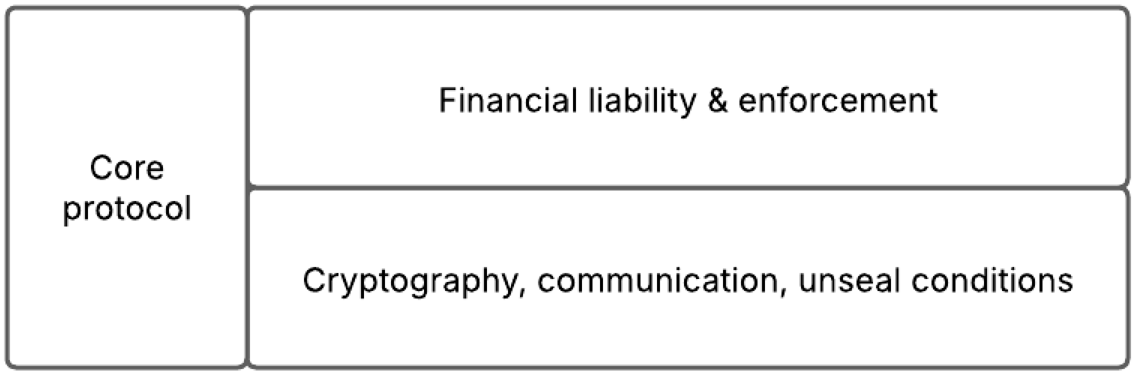
Unseal conditions are modular and application-specific. As long as a proof can be verified on-chain, it can serve as a gate. Examples include:

- Ownership of an email account (via ZKEmail)
- Proof of public key possession
- Timelocks
- Credentials signed by another authority
- TLS-Notary attestations
- ZKPassport proofs of identity fields (e.g., date of birth)

Unlike custodial/gated systems, the **onus of proof lies with the recovering party**, not with the processor. This keeps processors lightweight and stateless, reducing complexity and attack surface.

Architecture

Nihilium’s architecture revolves around three components: **Clients, Processors, and the Enforcement Layer**, bound together by modular **Unseal Conditions**.


Core protocol
Financial liability & enforcement
Cryptography, communication, unseal conditions

Clients

Clients initiate sealing and carry most of the cryptographic workload. To seal a secret, the client produces a **Severed Commitment** — a zero-knowledge proof that transforms commitments in a way that generates the unseal initiation value. This ensures that:

- The unseal conditions themselves are never revealed to processors.
- The sealed and unsealed states cannot be linked, preserving unlinkability and privacy.

Because the client defines the unseal conditions, responsibility for recovery lies entirely with the recovering party — not with custodians or intermediaries.

Redundancy with Shamir Secret Sharing

For resilience, the client SDK supports recovery thresholds using Shamir Secret Sharing. A sealed secret can be split into multiple shares distributed across different processors, so that recovery remains possible even if some processors are unavailable or go offline. This is purely a redundancy mechanism: it strengthens liveness but does not alter the core security model. Nihilium’s censorship-resistance and enforcement guarantees hold even with a single sealed package.

Processors

Processors commit to unsealing when the conditions are met. Their role is intentionally lightweight:

- They generate random data, sign the client's request, and become bound to act.
- They anchor activity through their own data-stream, an efficient batching method for publishing commitments on-chain, scaling to millions of inserts per block-time.
- When an unseal request arrives, processors pull validation bytecode directly from the blockchain and execute it in memory. This guarantees alignment with the enforcement layer and allows the protocol to evolve without processor software updates.

To ensure **long-term liveness**, processors are economically incentivized through fixed staking periods and mandatory shutdown signals. Each processor operates within a *must-act window*: if it stops responding abruptly, its stake can be slashed. If it intends to exit, it must follow a predefined shutdown schedule, giving clients and the network time to adjust. These mechanics discourage short-lived or unreliable processors and encourage stable, committed participation.

For operational security, processors can run inside TEEs (trusted execution environments), but the protocol does not depend on this assumption.

Enforcement Layer

The blockchain is the arbiter of last resort. Its functions are:

- Register processors and hold their stake.
- Maintain the repository of validation contracts for unseal conditions.
- Enable challenges when secrets leak prematurely.
- Force processors to act and reply on-chain, with ZK proofs ensuring verifiability.

If everything runs smoothly, the chain is almost dormant, used only for processor onboarding and staking updates. On-chain execution is only required when a slashable event occurs. This design allows the network to scale indefinitely while remaining censorship-resistant.

Unseal Conditions

Unseal conditions are the most flexible part of the protocol. They form an execution path — the **unseal condition root** — which the client commits to at sealing. To unseal, the client must gather and prove all steps in this path.

The framework supports a small set of primitives:

- Prepare proof (with verification code and public signals)
- Pass signals
- Validate static data
- Validate on-chain data
- Verify proof

From these building blocks, complex applications emerge. Examples include verifying ownership of an email address via ZKEmail, proving identity attributes via ZKPassport, checking historical NFT ownership, or even validating TLS-Notary attestations. Because conditions must be verifiable at any time after they become valid, designs naturally support cross-chain proofs and interoperable recovery logic. Nihilium's core protocol will provide an ever extending set of unseal conditions and invites developers to build their own.

Financials

Nihilium's incentive model reinforces censorship resistance.

- **Sealing** is prepaid, covering processor costs and staking requirements.
- **Unsealing** is free — once a commitment exists, processors are economically bound to execute.

This asymmetry ensures that no recovering party can ever be extorted with "pay-to-unseal" fees. A processor that refuses to act risks slashing, while an honest processor faces no incentive misalignment.

Applications

Web3 “Forgot My Password”

The clearest application of Nihilium is universal key recovery. A client seals a secret — a password, seed phrase, or private key — under chosen unseal conditions. To recover, the client **must publish the unseal initiation value on-chain, making the attempt observable**. The sealed package itself can be stored anywhere, even by a developer or third party, because only the rightful user can satisfy the unseal conditions (e.g., ZKEmail proof). The flow mirrors the familiar Web2 “forgot my password” experience, but without custodians or common trust assumptions.

Secure File Transfers

Our live application at transfer.nihilium.io demonstrates sealed file exchange. A file is encrypted, and its decryption key is placed in a sealed package. When the recipient unseals, both parties gain cryptographic proof that the file was accessed — without relying on a central authority. Unlike traditional file transfer services that enforce access centrally, Nihilium shifts access control into a decentralized proof layer. Developers can use the SDK to plug sealed transfers into any backend — S3, IPFS, or custom storage.

Emergency Access to Medical Data

Privacy is vital in healthcare, yet data must be accessible in emergencies. Nihilium enables patients to store critical medical data (e.g., allergies, medications) in sealed packages. Authorized staff can access the data during emergencies, but every unseal attempt is publicly observable. Patients can thus guarantee their data remains private until truly needed — and know if it was ever accessed prematurely.

Compliance & Oversight

Unseal conditions can be written to enforce due process especially in regulated contexts. For example, a root certificate might be stored under a “break-glass” procedure requiring multiple verifiable proofs before access. Or law enforcement could be given the ability to request access (as with a doorbell camera), but every access attempt would be logged and provable. We call this *keeping honest people honest*: even powerful actors cannot bypass the rules without detection.

An Application-Agnostic Primitive

These applications are only a few examples. Because Nihilium is built around sealed packages and verifiable unseal conditions, it is not tied to any single use case or ecosystem. Any condition that can be proven — from identity attestations to cross-chain state proofs — can serve as the key to unlock a secret. This makes Nihilium an **application-agnostic primitive**: a foundation for censorship-resistant recovery, coordination, and access control across domains well beyond Web3.

Differentiation

Nihilium is not just another key-recovery scheme or access-control tool. Several approaches exist today, but all fall short of combining censorship resistance, universality, and usability:

- **Not Social Recovery**
  Social recovery schemes rely on peers or guardians coordinating to return access. This creates orchestration overhead, privacy risks, and potential collusion. Nihilium requires no coordination beyond the user presenting proofs.

- **Not Custodial Recovery**
  Custodians can deny service, shut down, or fall under regulatory capture. In Nihilium, processors cannot selectively refuse requests without provable slashing, and users can store sealed packages anywhere without depending on a central operator.

- **Not Just MPC**
  Multi-party computation focuses on distributing trust across participants, but it usually locks recovery into a single context (e.g., one wallet). Nihilium's sealed packages are context-agnostic: the same primitive works for wallets, files, credentials, or identity proofs.

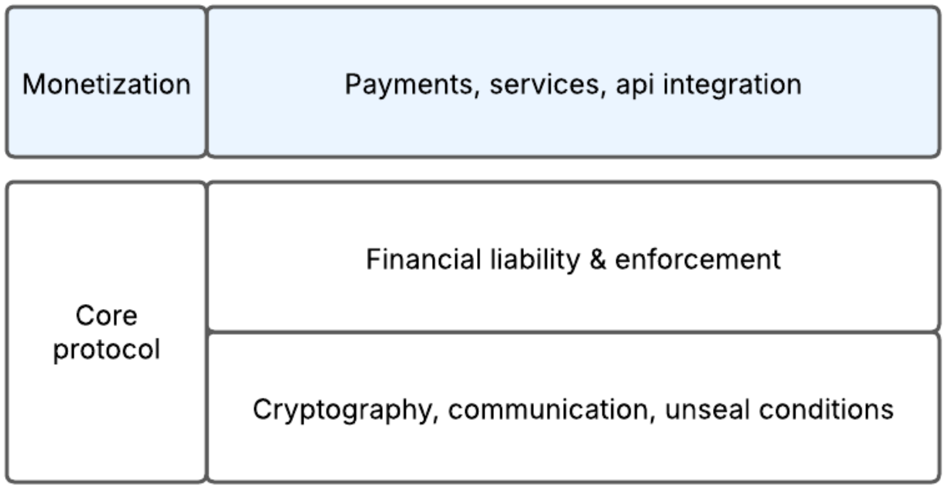
- **Proofs Are Client-Side**
  In many systems, servers or nodes are responsible for producing or orchestrating proofs. In Nihilium, the burden is inverted: the client generates all proofs, and processors only validate them. This keeps processors lightweight and stateless, reduces trust assumptions, and ensures the onus of proof always lies with the party seeking recovery.

What sets Nihilium apart is the **combination**:

- Context-agnostic sealed packages.
- Proof-driven recovery with no orchestration burden.
- Incentive-aligned processors bound by slashable commitments.
- Client-side proof generation with processor-side validation only.
- Minimal reliance on-chain, scaling indefinitely without bottlenecks.

Together, these elements make Nihilium a new primitive: **sealed access control**, capable of supporting recovery, compliance, and coordination without trust in intermediaries.

The business model


Monetization
Payments, services, api integration
Core protocol
Financial liability & enforcement
Cryptography, communication, unseal conditions

Nihilium’s business model is service-focused. The protocol itself is permissionless: anyone can run a processor, and any client can create sealed packages. But without coordination, developers would need to negotiate directly with processors — a fragmented and unworkable experience.

To solve this, **Nihilium the company will operate the sealing marketplace**. Developers can register, purchase capacity from processors, and integrate through familiar workflows using API keys and SDKs. This smooths developer adoption without compromising protocol guarantees: the client remains in full control, seals stay unlinkable, and unsealing is always free.

On top of this, Nihilium will provide **observability services**. Because sealed packages contain a public component, initiation events can be monitored externally. This enables notification services ranging from simple email alerts to app push notifications or even proactive outreach in critical scenarios.

Finally, Nihilium will also be an **early user of its own network**. Our first application, Nihilium Transfer, demonstrates sealed file transfers in production. Over time, we intend to launch and co-found additional products that leverage this primitive.

There is **no token planned** for the network initially. We believe payments to processors should remain an independent economic activity, and that the asset used for staking should not derive its value from speculative network dynamics.

In short: Nihilium makes a censorship-resistant primitive developer-friendly — by operating the marketplace, tools, and services that bring sealed packages into real applications at scale.

𝕏 @nihiliumio
Email: contact@nihilium.io