CIPNFT on GenesisL1: Cryptographic Information Protocol NFTs for Confidential Intellectual Property Tokenization and Exchange

A Post-Quantum-Aware, Ledger-Agnostic Architecture for DeSci Data Governance and Secure IP Transfer

Decentralized Science Labs

(reference implementation: browser-only client + on-chain encrypted metadata NFT contract)

February 2026

Abstract

The current reference deployment is implemented and operated exclusively on **GenesisL1**, an **EVM-capable Layer-1 blockchain** chosen to run this class of on-chain encrypted-metadata systems without depending on L2-specific trust or off-chain custodians.

Decentralized Science (DeSci) and broader intellectual property (IP) markets are increasingly constrained by a structural tension: open infrastructures excel at coordination, timestamping, and settlement, yet most valuable scientific artifacts—negative results, private datasets, trade secrets, pre-publication manuscripts, and commercialization pathways—require controlled disclosure. This paper introduces CIPNFT (Cryptographic Information Protocol Non-Fungible Token), a protocol and reference dApp for *fully on-chain* encrypted metadata NFTs enabling confidential data custody, verifiable integrity, and buyer-safe transfer with escrowed payments. CIPNFT stores ciphertext on-chain and maintains a compact, updatable *key envelope* that is re-wrapped to the new owner upon transfer, so only the current owner (or an optional view-key holder) can decrypt. Unlike conventional NFT designs that outsource metadata to mutable web servers or rely on opaque custodians, CIPNFT makes the ciphertext a first-class, consensus-protected object. In this sense, CIPNFT functions as an *IP tokenization protocol*: it tokenizes a cryptographic capability (the right to decrypt a specific encrypted knowledge capsule) rather than attempting to tokenize information itself.

The protocol supports two envelope modes: (i) a standard public-key "sealed box" envelope (based on X25519 and a proven AEAD construction), and (ii) an optional post-quantum envelope based on ML-KEM-768 combined with XChaCha20-Poly1305 to protect confidentiality against adversaries with large-scale quantum computers. Crucially, CIPNFT includes a two-phase sale mechanism (*deliver* then *finalize*) that forces the buyer to verify decryption correctness *before* releasing escrowed payment, addressing a fundamental fairness failure of encrypted-asset transfers on blockchains.

We provide a precise specification of the on-chain data model, the client cryptographic pipeline, and the deal-flow state machine, including a complete Alice-and-Bob example for IP sale finalization. We analyze security, limitations (notably the impossibility of cryptographic revocation after disclosure), and offer a forward-looking discussion on ledger survivability: how CIPNFT can outlive any particular blockchain by preserving a self-describing cryptographic "data capsule" that can be migrated or re-instantiated without forfeiting confidentiality guarantees.

Keywords: encrypted NFTs, intellectual property, IP tokenization, tokenization, confidential data exchange, DeSci, escrow, post-quantum cryptography, ML-KEM-768, XChaCha20-Poly1305, key envelopes, on-chain metadata, ledger migration.

1 Introduction

CIPNFT as IP tokenization. At a protocol level, CIPNFT can be viewed as *IP tokenization* for confidential knowledge: the NFT represents a transferable, verifiable capability to decrypt a specific ciphertext capsule, and market operations (listing, escrowed offers, delivery, finalization) provide an auditable mechanism for licensing or sale. Hence, the "token" is not a claim that the underlying idea becomes scarce; it is a portable, consensus-tracked handle over a cryptographic access right.

Blockchains provide public verifiability, globally consistent ordering, and programmable settlement. For science and IP markets, these primitives are attractive for: timestamped provenance, reproducible audit trails, automated licensing, and new funding mechanisms. Yet public ledgers also create an immediate confidentiality problem: if research artifacts or proprietary datasets are published in plaintext, they are instantly and irrevocably disclosed.

Most NFT infrastructures respond by storing only a pointer (URL) to metadata and placing the data behind a private server, a cloud bucket, or a gated API. This approach reintroduces trusted intermediaries and introduces fragility: link rot, centralized censorship, mutable metadata, and uncertain long-term availability. Recent analyses show significant centralization of NFT metadata hosting in practice, undermining the permanence narrative of NFTs.

CIPNFT addresses this tension directly by elevating ciphertext to a native on-chain object. The protocol is built around a simple but powerful idea:

> *Store only ciphertext (and integrity commitments) on-chain, while transferring decryption capability*

through an updatable key envelope that is tied to token ownership.

This design makes confidentiality a *cryptographic* property rather than an access-control policy enforced by a server. The blockchain acts as a durable, censorship-resistant publication layer for encrypted artifacts, while access is mediated by transferable cryptographic capability.

1.1 Contributions

This paper contributes:

1. A formalized architecture for on-chain encrypted metadata NFTs, including precise byte-level layout and a commitment-based verification mechanism for envelope correctness.

2. A buyer-safe escrowed sale design with a two-step transfer (*deliver* then *finalize*) that prevents payment release for incorrect key re-wrapping.

3. An optional post-quantum envelope mode (ML-KEM-768) that can be used for long-lived IP confidentiality under plausible quantum adversaries.

4. A DeSci-focused framework for secure knowledge transfer and private dataset governance, including licensing and deal finalization flows.

5. A ledger-survivability analysis showing how CIPNFT can be re-instantiated on a different chain without abandoning the core confidentiality model.

1.2 Scope and disclaimers

The reference implementation described here consists of: (i) a Solidity smart contract implementing an ERC-721-like interface with deliberate transfer restrictions to preserve envelope correctness, and (ii) a browser-only client using EIP-1193 wallet connectivity and local cryptography (`libsodium` and an ML-KEM-768 implementation).

This paper is technical, not legal advice. CIPNFT can encode licensing terms and acceptance proofs, but enforceability depends on jurisdiction, contracting, and governance design. The implementation is a reference design and must be professionally audited before production use.

2 Problem Statement

We consider a recurring pattern in DeSci and IP markets:

1. An owner holds valuable information M (e.g., a dataset manifest, a protocol description, an invention disclosure, an unpublished manuscript).

2. The owner wants to timestamp and commit to M in a public, verifiable system.

3. The owner may wish to sell or license M to a counterparty in a way that:

   (a) hides M from everyone else,

   (b) preserves integrity and provenance,

   (c) provides a fair exchange (no payment without correct access; no access without payment),

   (d) is compatible with long-term confidentiality needs.

A naïve approach is to encrypt M off-chain and store ciphertext C on a server; the seller then transfers the decryption key to the buyer after receiving payment. This fails under adversarial conditions: the buyer cannot verify correctness pre-payment; the seller cannot credibly commit to the encrypted content; and the availability depends on server uptime.

We therefore seek a design in which:

- The ciphertext C is published on-chain under consensus durability.
- The decryption capability is transferable and bound to NFT ownership.
- The deal flow is *fair* with respect to a rational adversary.

3 Design Goals and Non-goals

3.1 Goals

G1: On-chain ciphertext durability. The ciphertext should be stored *fully on-chain* (not merely referenced), ensuring availability and immutability as long as the ledger persists.

G2: Confidentiality under public observation. Anyone can read the chain state; confidentiality must come from cryptography, not access control.

G3: Transferable decryption capability. Ownership transfer should correspond to the ability to decrypt, via a re-wrapped envelope.

G4: Buyer-safe exchange. A buyer depositing payment into escrow should not be forced to pay unless decryption correctness can be verified.

G5: Post-quantum option. The protocol should support a post-quantum envelope mode for long-lived secrets.

G6: Ledger survivability. The protocol should be instantiable across ledgers; data capsules should be migratable without losing confidentiality guarantees.

3.2 Non-goals

N1: Revocation / DRM. If a party legitimately learns M, cryptography cannot prevent them from copying it. CIPNFT is *not* digital rights management.

N2: Perfect unlinkability. On-chain ciphertext sizes, listing prices, and transaction patterns can leak metadata.

N3: On-chain correctness verification of envelope decryption. Without suitable cryptographic precompiles, a typical EVM cannot verify X25519 or ML-KEM-768 decapsulation; verification is performed client-side.

4 System Model and Notation

4.1 Entities

We use the following entities:

- **Owner / Seller** (Alice): holds private key material and can decrypt and re-wrap the token's DEK.
- **Buyer** (Bob): wishes to acquire decryptability through escrowed purchase.
- **Blockchain / Contract** ($\mathcal{L}$): a ledger that stores ciphertext and enforces transfer / escrow rules.
- **Client** (C): browser-only software that performs cryptography locally.

4.2 Threat model

The adversary can:

- read all on-chain state and transactions,
- submit transactions and observe mempool activity,
- attempt to purchase, front-run, or spam offers,
- compromise user browsers (outside the scope of cryptography).

We assume standard cryptographic hardness of the chosen primitives and that users protect their private seeds.

4.3 Notation

Let:

- M be plaintext metadata (UTF-8 bytes), bounded in size.
- $\mathsf{DEK} \leftarrow \{0,1\}^{256}$ be a random 32-byte data-encryption key.
- $\mathsf{Keccak256}(\cdot)$ be Keccak-256 (as used by Ethereum), producing a 32-byte digest.
- $\mathsf{XChaCha20\text{-}Poly1305}$ be an AEAD scheme with 24-byte nonce and 16-byte tag.
- $\mathsf{Seal}(\cdot)$ denote anonymous public-key encryption (sealed box) to a recipient public key.

5 Cryptographic Construction

5.1 Core encapsulation principle

A CIPNFT token is a *cryptographic capsule*:

$$\mathcal{T} = \left(C, E_{\text{owner}}, W_{\text{view}}, h\right)$$

where:

- C is on-chain ciphertext of M under DEK,
- E_{owner} is an owner-only envelope containing DEK,
- W_{view} is an optional "view-key wrap" of DEK,
- $h = \mathsf{Keccak256}(\mathsf{DEK})$ is a commitment to the DEK.

5.2 Associated data binding

To prevent replay of ciphertext across contracts or chains, the reference implementation uses associated data

$$\mathsf{AAD} \leftarrow \mathsf{Encode}(\texttt{chainId}, \texttt{contractAddress}).$$

Then encryption uses

$$C \leftarrow \mathsf{nonce} \parallel \mathsf{XChaCha20\text{-}Poly1305.Enc}_{\mathsf{DEK}}(M; \mathsf{AAD}).$$

Decryption checks the tag under the same AAD.

Interpretation. This does not "secure the blockchain"; rather it domains-separates ciphertext so an adversary cannot copy ciphertext and claim it belongs to a different contract context without detection. For ledger migration, CIPNFT can either (i) re-encrypt under the new AAD, or (ii) treat the original AAD as part of the capsule context; see Section 13.

5.3 Key envelope modes

CIPNFT supports two mutually exclusive envelope modes for E_{owner}.

5.3.1 Mode 0: Classic sealed box

Let $(\mathsf{pk}_u, \mathsf{sk}_u)$ be the user encryption key pair, with $\mathsf{pk}_u \in \{0,1\}^{256}$ (32 bytes) in X25519 format. Then

$$E_{\text{owner}} \leftarrow \mathsf{Seal}_{\mathsf{pk}_{\text{owner}}}(\mathsf{DEK}),$$

implemented as `crypto_box_seal` in `libsodium`. This is an anonymous envelope: the recipient can decrypt but cannot identify the sender by cryptographic means.

Size. For a 32-byte message, the sealed box has 48 bytes of overhead, producing an 80-byte envelope.

5.3.2 Mode 1: Post-quantum envelope

Let $\mathsf{pk}_u^{\mathrm{PQC}}$ be an ML-KEM-768 public key (1184 bytes in the reference implementation). Encapsulation produces:

$$(\mathsf{ct}, \mathsf{ss}) \leftarrow \mathsf{KEM.}(\mathsf{pk}_{\mathrm{owner}}^{\mathrm{PQC}}),$$

where $\mathsf{ss} \in \{0,1\}^{256}$ is a 32-byte shared secret. Then DEK is wrapped under ss using AEAD with the same AAD:

$$E_{\mathrm{owner}} \leftarrow \mathsf{ct} \parallel \mathsf{nonce}_e \parallel \mathsf{XChaCha20\text{-}Poly1305.Enc}_{\mathsf{ss}}(\mathsf{DEK}; \mathsf{AAD}).$$

Size. In ML-KEM-768, ct is 1088 bytes. The AEAD wrap adds 24-byte nonce and 16-byte tag for a 32-byte message, totaling:

$$|E_{\mathrm{owner}}| = 1088 + 24 + (32 + 16) = 1160 \text{ bytes}.$$

5.4 Optional view key wrap

A view key enables controlled disclosure to non-owners without changing token ownership. Let v be a human-shareable view key string. Derive a symmetric key via a hash:

$$K_v \leftarrow \mathsf{H}(v),$$

where the reference uses BLAKE2b-256 (libsodium `crypto_generichash`) for $K_v \in \{0,1\}^{256}$. Then:

$$W_{\mathrm{view}} \leftarrow \mathsf{nonce}_v \parallel \mathsf{XChaCha20\text{-}Poly1305.Enc}_{K_v}(\mathsf{DEK}; \varnothing).$$

Anyone who learns v can recover DEK and decrypt C. The view key can be rotated by the current owner by rewriting W_{view}.

5.5 Commitment for envelope correctness

Because the chain cannot verify that a provided E_{owner} decrypts to the correct DEK, we store:

$$h \leftarrow \mathsf{Keccak256}(\mathsf{DEK}).$$

Clients validate a decrypted DEK by checking $\mathsf{Keccak256}(\mathsf{DEK}) = h$ before attempting metadata decryption.

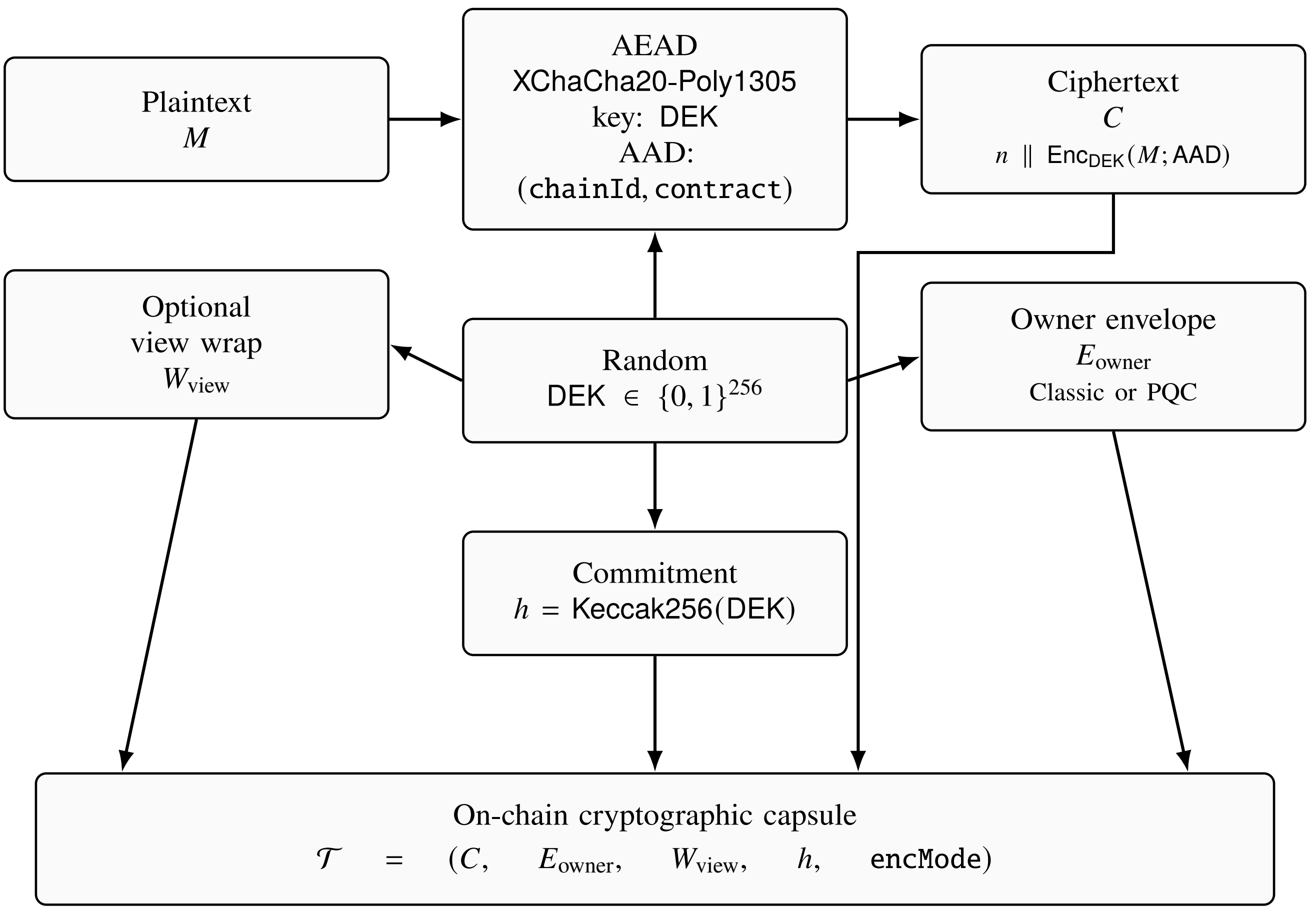

Figure 1: Capsule schematic (full page): ciphertext is consensus-durable on GenesisL1; decryptability is a transferable capability via the owner envelope.

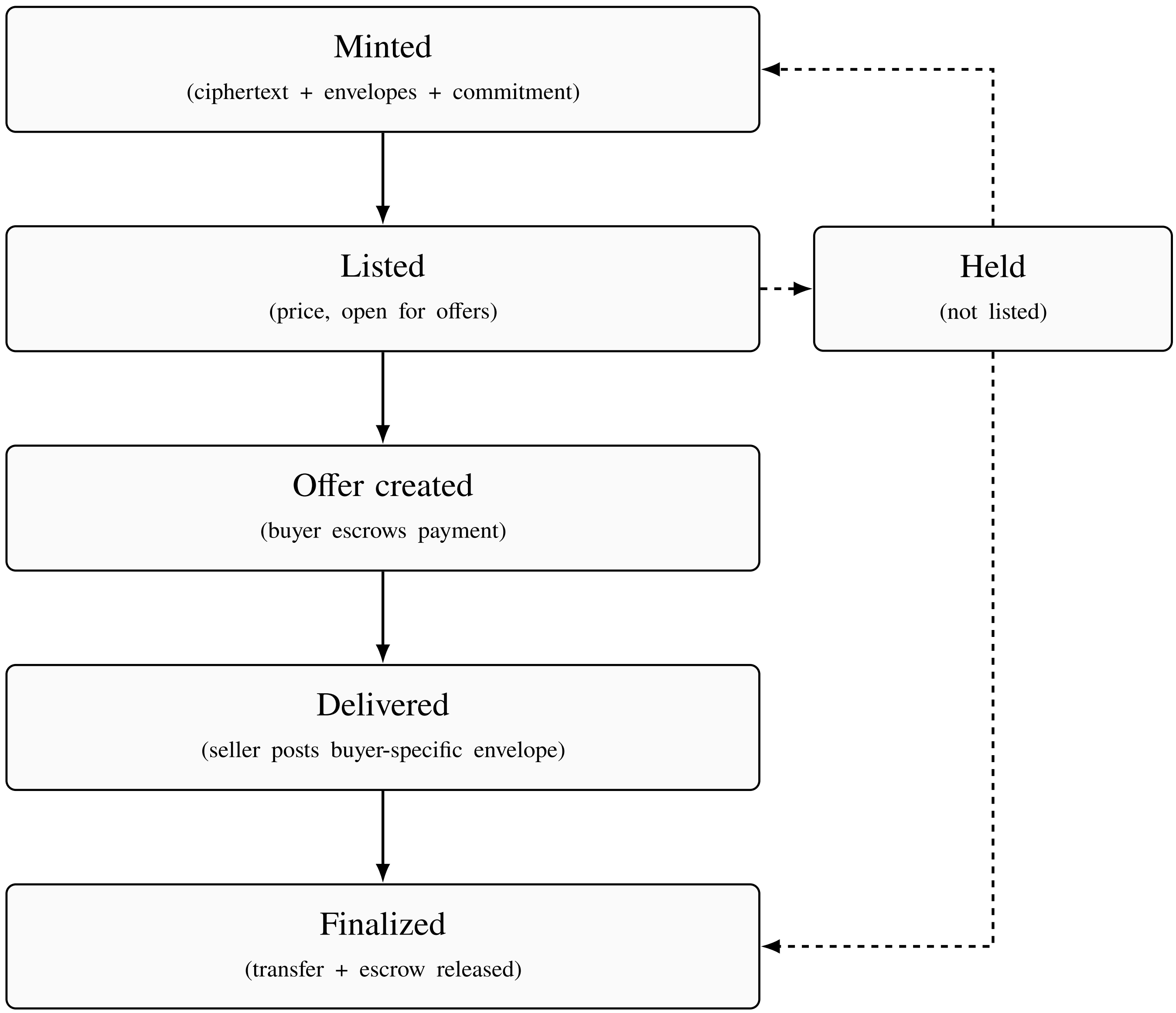

Figure 2: Token market state machine (GenesisL1 reference deployment): transfers are mediated by escrow so the buyer can verify decryptability before funds are released.

1. Bob creates an offer on GenesisL1 (on-chain).
Buyer calls `createOffer(tokenId,expiry)` and escrows the exact asking price.

2. Alice prepares delivery (off-chain).
Alice decrypts the current owner envelope to recover DEK, then re-wraps DEK to Bob's registered public key.

3. Alice delivers on GenesisL1 (on-chain).
Seller posts `deliverOffer(tokenId,buyer,newOwnerEncDEK)`. No payment is released yet.

4. Bob verifies decryptability (off-chain).
Bob decrypts `newOwnerEncDEK` $\Rightarrow \mathsf{DEK}$, checks $\mathsf{Keccak256}(\mathsf{DEK}) = h$, then decrypts ciphertext C to obtain M.

5. Bob finalizes on GenesisL1 (on-chain).
Buyer calls `finalizeOffer(tokenId)`. Contract transfers the NFT and releases escrow to Alice.

Figure 3: Buyer-safe deal flow (GenesisL1): deliver first, then finalize only after buyer validation.

6 On-Chain Protocol: Data Model and State Machine

6.1 Token data layout (reference implementation)

The reference contract stores the following per-token fields:

Field	Meaning
`metaCipher`	Bytes: n $\|$ $\mathsf{XChaCha20\text{-}Poly1305.Enc}_{\mathsf{DEK}}(M; \mathsf{AAD})$; $\lvert M\rvert \le 64$ KiB
`ownerEncDEK`	Owner envelope (80B classic or 1160B PQC)
`viewWrap`	Optional view-key wrap of DEK (72B)
`dekHash`	$h = \mathsf{Keccak256}(\mathsf{DEK})$ commitment
`encMode`	0=classic sealed-box, 1=PQC envelope

Additionally, the token includes: (i) a short human-readable title stored in plaintext (bounded by 80 bytes and 5 words), and (ii) the Terms-of-Service version required at mint time.

6.2 Why transfers are restricted

Standard ERC-721 allows arbitrary transfers via `transferFrom` and approvals. CIPNFT intentionally disables these because transferring ownership without updating `ownerEncDEK` would strand ciphertext under the wrong envelope, breaking the "owner can decrypt" invariant. Therefore, the reference contract redefines transfer semantics through an escrowed offer flow that updates the envelope in the same logical deal process.

6.3 Listings, offers, delivery, finalization

The protocol implements a discoverable on-chain marketplace:

- An owner may *list* a token with an asking price.
- A buyer can *create an offer* by escrow-depositing the exact asking price until expiry.
- The seller *delivers* by posting a re-wrapped envelope for that buyer.
- The buyer *finalizes* after verifying correctness, transferring the token and releasing escrow.

This two-phase design is essential: the seller cannot be paid until the buyer has an opportunity to decrypt and validate.

6.4 State machine

7 Client Architecture: Browser-Only Cryptography

7.1 Rationale

Placing encryption in the browser reduces trust in centralized backends. The client:

- generates or derives encryption identities locally,
- performs AEAD encryption/decryption,
- re-wraps envelopes during sale,
- verifies envelope correctness using $h = \mathsf{Keccak256}(\mathsf{DEK})$.

No plaintext needs to be sent to a server.

7.2 Key identities

The client maintains two independent encryption identities:

1. **Classic identity:** X25519 key pair derived from a 32-byte seed.
2. **PQC identity:** ML-KEM-768 key pair derived from a raw seed (library-defined size).

Wallet-signature-derived seeds. For usability, the reference implementation can derive seeds from an -191 style wallet signature over a domain-separated message containing origin, account, chain ID, and contract address. This reduces friction but introduces an important caveat: some wallets may not produce deterministic signatures. Therefore, the client supports explicit seed export/import and on-chain key registration.

8 Protocol Flows

8.1 Flow 1: Key registration

Before minting or receiving encrypted tokens, a user registers their public encryption key on-chain:

- Classic mode registers a 32-byte X25519 public key.
- PQC mode registers the 1184-byte ML-KEM-768 public key.

This registration binds the wallet address to an encryption public key used in envelopes.

8.2 Flow 2: Minting an encrypted IP token

To mint, the owner (Alice) selects an envelope mode and provides plaintext metadata M. The client computes $(C, E_{\text{owner}}, W_{\text{view}}, h)$ and submits them to the contract with required fees. The chain stores ciphertext and envelope; M never appears on-chain.

Listing 1: Algorithm: Encrypt-and-mint (client-side)

```
Inputs: plaintext M, title T, owner pubkey pk_owner, mode
     {classic,pqc}
Parameters: AAD = Encode(chainId, contractAddress)

1. DEK $ {0,1}^256
2. C  nonce || XChaCha20-Poly1305.Enc_DEK(M; AAD)
3. if mode = classic:
      E_owner  Seal_{pk_owner}(DEK) // 80 bytes
   else if mode = pqc:
      (ct, ss)  ML-KEM-768.Encaps(pk_owner) // ct 1088B,
           ss 32B
      E_owner  ct || nonce_e || XChaCha20-Poly1305.Enc_ss(
           DEK; AAD) // 1160B
4. h  Keccak256(DEK)
5. Optional: W_view  nonce_v || XChaCha20-Poly1305.Enc_{H(
     viewKey)}(DEK)
6. Call contract.mint(T, C, E_owner, h, W_view, encMode)
     with fee
```

8.3 Flow 3: Viewing/decrypting

A party can decrypt either:

1. as the owner: open E_{owner} with their private key to obtain DEK, or
2. as a view-key holder: open W_{view} using the view key to obtain DEK.

Then verify $\mathsf{Keccak256}(\mathsf{DEK}) = h$ and decrypt C under DEK.

9 Deal Design: Buyer-Safe Finalization

9.1 Motivation

Encrypted asset markets face a fundamental asymmetry:

> If the ciphertext is public, only the seller can produce a key-wrapped envelope for the buyer—and the chain cannot verify it is correct.

A seller could provide an incorrect envelope and still attempt to receive payment. Therefore CIPNFT introduces a *two-step* deal:

1. **Deliver**: seller posts the re-wrapped envelope on-chain (no transfer, no payment).
2. **Finalize**: buyer verifies decryption correctness off-chain, then calls finalize to transfer and pay.

9.2 Offer escrow

An offer is created by depositing exact asking price into escrow, along with:

- an expiry timestamp,
- a snapshot of the buyer's encryption public key reference.

Key snapshotting prevents the buyer from changing their registered key after the offer is made (which could otherwise invalidate delivery).

9.3 Deliver

The seller:

1. decrypts the current owner envelope to recover DEK,
2. produces a new envelope under the buyer's public key,
3. posts it via `deliverOffer(tokenId,buyer, newOwnerEncDEK,newViewWrap)`.

No funds are released.

9.4 Finalize

The buyer:

1. retrieves delivered envelope,
2. decrypts it using their private key to obtain DEK,
3. checks $\mathsf{Keccak256}(\mathsf{DEK}) = h$ and validates that C decrypts,
4. calls `finalizeOffer(tokenId)`.

The contract then updates the on-chain owner envelope, transfers the NFT, and pays the seller from escrow.

10 Alice–Bob Example: A Complete IP Sale

We now walk through a concrete scenario using the reference flow.

10.1 Setup

Alice possesses an invention disclosure and experimental protocol M and wants to sell exclusive access. She registers her encryption public key on-chain and accepts the current on-chain Terms.

She mints token #42 with:

- title: "Neuroprobe Microfluidic Method",
- ciphertext C of her disclosure under DEK,
- owner envelope E_{Alice} containing DEK,

- commitment $h = \mathsf{Keccak256}(\mathsf{DEK})$,
- no view key (exclusive sale).

She lists the token at 10 L1.

Bob is a biotech founder who wants to purchase. He registers his own encryption public key on-chain.

10.2 Offer creation

Bob calls `createOffer(42, expiry)` with exactly 10 L1 as escrow. The contract records:

- `amountWei` = 10 L1,
- `expiry` = now + 7 days (bounded by protocol rules),
- `buyerKeyRef` = (Bob's pubkey snapshot).

10.3 Delivery

Alice monitors offers and selects Bob. She fetches token data $(C, E_{\text{Alice}}, \varnothing, h)$ from chain, decrypts E_{Alice} to recover DEK, and checks $\mathsf{Keccak256}(\mathsf{DEK}) = h$.

She then produces:

$$E_{\text{Bob}} \leftarrow \mathsf{Seal}_{\mathsf{pk}_{\text{Bob}}}(\mathsf{DEK})$$

(or the PQC envelope if the token is in PQC mode). She submits `deliverOffer(42,Bob,E_Bob,newViewWrap)`. No transfer occurs and no payment is released.

10.4 Buyer verification

Bob retrieves the delivery from chain, decrypts E_{Bob} to get DEK, and checks $\mathsf{Keccak256}(\mathsf{DEK}) = h$. He then decrypts C under DEKand obtains plaintext M. Only once this succeeds does Bob proceed.

10.5 Finalization

Bob calls `finalizeOffer(42)`. The contract verifies offer validity and delivery presence, updates `ownerEncDEK` to E_{Bob}, transfers token ownership to Bob, and releases the escrowed 10 L1 to Alice.

Outcome. Alice gets paid *only after* Bob can decrypt. Bob gets the token (and future resale rights) only through a state transition that updates the envelope to his key.

11 IP Data Protection and DeSci Workflows

11.1 Why DeSci needs confidentiality

DeSci aims to improve coordination, funding, and incentives in scientific production. However, scientific artifacts routinely traverse phases: *private ideation → collaborative development → selective disclosure (review) → publication / commercialization*. Each phase has different confidentiality requirements.

CIPNFT supports these phases by treating an artifact as a cryptographic capsule whose decryptability is transferable or shareable via view keys.

11.2 Patterns enabled by CIPNFT

GenesisL1 ecosystem primitive: PEPTL and CIPMolNFTs. On GenesisL1, CIPNFT is intended to function as a reusable confidentiality primitive for domain-specific scientific assets. For example, in a PEPTL peptide design studio, candidate peptides (and associated design rationales, activity assays, or proprietary scoring models) can be tokenized as *molecular NFTs* (*CIPMolNFTs*) whose descriptive payloads are stored as CIPNFT encrypted capsules. This enables selective disclosure to collaborators, escrowed transfers to commercialization partners, and durable provenance for iterative molecular design workflows while preserving confidentiality prior to publication or patent filing.

P1: Private timestamping without disclosure. A researcher can commit to a detailed method M by minting ciphertext C on-chain. This creates a public timestamp and immutable commitment without revealing the content.

P2: Controlled peer review. A researcher can share a view key with reviewers (time-bounded operationally, not cryptographically), enabling decryption without transferring ownership.

P3: Collaboration and handoff. A lab can transfer ownership of the capsule to a partner institution, moving decryptability through the owner envelope without relying on shared passwords or centralized vaults.

P4: IP sale or licensing with escrow. Because the encrypted capsule is represented as an NFT, these workflows naturally support *tokenized IP transfer* and secondary-market licensing primitives on GenesisL1. The deliver/finalize mechanism supports fair exchange. Licenses can be included in M (e.g., a signed PDF hash and terms) and/or represented via auxiliary tokens.

P5: Secure dataset manifests. Large datasets can be stored off-chain (e.g., IPFS/Arweave/S3) while the on-chain plaintext contains only an encrypted *manifest*: content hashes, chunking strategy, access policy, and keys for encrypted blobs. The token becomes a transferable access capability to verifiable data.

11.3 Limits: disclosure is irreversible

CIPNFT does not prevent a legitimate recipient from redistributing M. Instead, it provides: (i) cryptographic integrity and provenance, (ii) controlled capability transfer, (iii) auditable deal completion. These are necessary but not sufficient components of a broader DeSci governance stack.

12 Post-Quantum Resistance

12.1 Quantum threat model

Large-scale quantum computers threaten classical public-key cryptography based on discrete logarithms and factoring. In particular, Shor's algorithm breaks elliptic-curve Diffie–Hellman and RSA in polynomial time. Therefore, any confidentiality that depends on X25519 may be compromised retroactively by a "store-now-decrypt-later" adversary.

Symmetric cryptography is more resilient: Grover's algorithm provides at most a quadratic speedup for brute force search, effectively halving security bits. Thus 256-bit symmetric keys remain conservative under quantum attack.

12.2 Why envelope mode matters

In CIPNFT

- The *payload* uses 256-bit symmetric AEAD (XChaCha20-Poly1305), which is robust.
- The *envelope* determines whether an adversary can obtain DEK.

Therefore the envelope is the main long-term confidentiality risk.

12.3 PQC envelope using ML-KEM-768

ML-KEM-768 is a module-lattice-based key encapsulation mechanism standardized by NIST (FIPS 203). It provides a public-key KEM that is believed secure against quantum adversaries under the Module-LWE assumption. CIPNFT uses ML-KEM-768 as follows:

$$\begin{aligned}
&(\mathsf{ct}, \mathsf{ss}) \leftarrow \mathsf{ML\text{-}KEM\text{-}768}.(\mathsf{pk}^{\mathrm{PQC}}_{\mathrm{owner}}),\\
&E_{\mathrm{owner}} := \mathsf{ct} \parallel \mathsf{nonce}_e \parallel\\
&\qquad\qquad \mathsf{XChaCha20\text{-}Poly1305.Enc}_{\mathsf{ss}}(\mathsf{DEK}; \mathsf{AAD}).
\end{aligned}$$

This is a KEM–DEM-style hybrid: the KEM provides a shared secret, and AEAD wraps the DEK.

12.4 Migration from classic to PQC

A key advantage of the envelope architecture is *upgradability*:

- If a token was minted in classic mode, the owner can decrypt DEKand re-wrap to a PQC public key, *but* the contract stores the mode per token.
- The reference implementation treats mode as immutable per token for simplicity; a future version can support dual envelopes or mode upgrades.

12.5 Hybrid envelopes (future direction)

For maximum robustness, a hybrid envelope can store both:

$$E = E^{\mathrm{classic}} \parallel E^{\mathrm{PQC}},$$

allowing decryption by either scheme during the transition era and preserving security if either classical or PQC assumptions fail. This increases on-chain storage costs but may be justified for high-value, long-lived IP.

13 Ledger Survivability and Protocol Migration

13.1 Protocol vs ledger

CIPNFT is a *representation* of a cryptographic capsule and a *set of state transition rules* for escrowed transfer. The protocol's longevity therefore depends on:

- the continued security of the cryptographic primitives,
- the ability to store bytes and enforce ownership rules on some ledger,
- the social/market recognition of continuity between instances.

13.2 Why CIPNFT can outlive a chain

GenesisL1 as canonical origin and design target. Even though the canonical instance is deployed exclusively on GenesisL1 today (an EVM-capable L1), the capsule representation is portable as bytes and does not inherently depend on GenesisL1's consensus rule set.Even if a particular chain becomes unusable or economically irrelevant, a CIPNFT token can be reconstructed from:

$$\mathcal{T} = (C, E_{\mathrm{owner}}, W_{\mathrm{view}}, h, \texttt{encMode}, \texttt{title}, \texttt{tosVersion}),$$

which are all on-chain artifacts. This tuple is portable as data. Hence, the capsule can be migrated or wrapped on a new ledger without requiring the old chain to remain economically dominant. The confidentiality of M depends on the secrecy of DEK and the security of the envelope, not on the chain's consensus algorithm.

13.3 Practical migration patterns

We outline two migration patterns.

Pattern A: Re-mint after decryption. The current owner decrypts M, then re-encrypts and re-mints on the new chain under a new AAD context. This provides clean domain separation but requires access to plaintext.

Pattern B: Capsule preservation with continuity certificate. The owner exports the capsule tuple $\mathcal{T}$ and mints a new token whose plaintext contains: (i) the full prior capsule tuple, and (ii) a signed statement linking old and new token IDs.

Let σ be a wallet signature on a message:

$$\begin{aligned}\mathsf{link} = \big(&\texttt{oldChain}, \texttt{oldContract}, \texttt{oldTokenId},\\ &\texttt{newChain}, \texttt{newContract}, \texttt{newTokenId},\\ &\mathsf{Keccak256}(C \parallel h)\big).\end{aligned}$$

This forms a portable audit trail. Clients can verify σ under the owner's on-chain address history. In this model, decryption uses the *original* AAD context; the migrated token acts as a recognized wrapper.

Observation. The reference implementation's use of $\mathsf{AAD} = (\texttt{chainId}, \texttt{contract})$ is a strong anti-replay defense within a chain ecosystem, but it means Pattern B requires clients to preserve the original AAD when decrypting legacy capsules. This is a solvable engineering issue: store the originating AAD context inside the migrated token and use it for decryption.

14 Security Analysis

14.1 Confidentiality

Assuming:

1. XChaCha20-Poly1305 is IND-CPA/IND-CCA secure as AEAD,
2. the envelope scheme is IND-CCA secure (classic sealed boxes) or KEM-secure (ML-KEM-768),

then an on-chain adversary who observes $(C, E_{\text{owner}}, W_{\text{view}}, h)$ learns nothing about M beyond length and traffic patterns, unless they obtain DEK. Hence confidentiality is reduced to protecting envelope secrecy under the chosen cryptographic assumptions.

14.2 Integrity and authenticity of ciphertext

AEAD ensures integrity of C under DEK and AAD. An adversary cannot modify C without detection.

14.3 Envelope correctness and buyer safety

The chain cannot cryptographically verify that delivered envelopes decrypt to DEK. Instead, correctness is verified by the buyer client using $h = \mathsf{Keccak256}(\mathsf{DEK})$ and actual ciphertext decryption. The two-phase deliver/finalize flow ensures the seller cannot extract escrow without buyer verification.

14.4 Key compromise

If a user's local seed is compromised, adversaries can decrypt owned tokens and potentially re-wrap envelopes. Users should:

- keep derived seeds offline and back them up securely,
- rotate envelopes by updating `ownerEncDEK` after key rotation (supported for owners),
- prefer PQC mode for long-lived secrets.

14.5 Impossibility of revocation

Once a buyer decrypts M, they can copy it. Thus confidentiality after disclosure is a legal and governance problem, not a cryptographic one. CIPNFT can, however, provide evidence of when a disclosure-capability changed hands.

14.6 State erasure

The reference contract includes a multi-transaction `eraseTokenData` procedure that scrubs *current* storage slots for C, envelopes, and commitments. This reduces exposure in the current state trie but cannot erase historical chain data or archival copies.

15 Implementation Notes (Reference Contract and Client)

GenesisL1 deployment note. The reference contract suite and browser client are deployed *exclusively* on **GenesisL1**, an **EVM-compatible L1** chosen to run large on-chain ciphertext payloads together with escrowed deal finalization. Other EVM ledgers can host compatible instances, but GenesisL1 is the canonical deployment referenced by this whitepaper.

15.1 Byte limits

The reference contract enforces:

- $|M| \le 64$ KiB.
- $|C| \le |M| + 24 + 16$.
- $|E_{\text{owner}}| = 80$ (classic) or 1160 (PQC).
- $|W_{\text{view}}| \in \{0, 72\}$.

15.2 Terms-of-service and fee model

A versioned Terms-of-Service text is stored on-chain, and users must accept the current version before minting. Mint fees are linear in plaintext size:

$$\mathsf{fee} = f_{\mathsf{flat}} + f_{\mathsf{byte}} \cdot |M|.$$

Fees accumulate for the administrator and are withdrawable without touching escrowed offers.

15.3 On-chain anti-spam rules

To reduce spam:

- listings must meet a minimum price,
- offers have bounded durations (default 7 days),
- offers escrow exact asking price.

16 Conclusion

Operational status. The system is deployed exclusively on **GenesisL1** because GenesisL1 is an **EVM Layer-1** capable blockchain suitable for running on-chain encrypted-capsule escrow logic at the protocol layer.CIPNFT provides a concrete, implementable design for confidential scientific and IP artifacts on public ledgers: ciphertext is durable and immutable on-chain; decryptability is transferable via key envelopes; and buyer-safe exchange is enforced by a deliver/finalize deal flow. By including a post-quantum envelope option and by framing the token as a portable cryptographic capsule, CIPNFT can support long-term confidentiality and can plausibly outlive any single ledger by migration or re-instantiation.

Disclaimer and Security Limitations

Informational only; no advice. This whitepaper is provided solely for informational, technical, and research purposes. Nothing herein constitutes legal, financial, investment, tax, regulatory, or security advice. You should obtain independent professional advice before acting on any information in this document.

Experimental; "AS IS"; no warranties. The protocol, reference implementation(s), smart contracts, and any accompanying materials are experimental and provided **"AS IS"** and **"AS AVAILABLE."** To the maximum extent permitted by applicable law, the authors and contributors disclaim all warranties of any kind, express or implied, including but not limited to merchantability, fitness for a particular purpose, non-infringement, title, accuracy, and security. No representation is made that the system is error-free, secure, or suitable for any use case.

High-risk; use at your own risk. Use may result in irreversible outcomes, including loss of funds or assets, permanent disclosure of confidential information, inability to decrypt data, transaction failures, or denial of service.

Security limitations (non-exhaustive):

1. **No revocation / no DRM:** once plaintext is disclosed to any party, it can be copied and redistributed; cryptography cannot prevent this.
2. **Endpoint compromise:** confidentiality depends on client devices, browsers, wallets, and dependencies; compromise may reveal keys and plaintext.
3. **Blockchain and infrastructure risk:** reorgs, forks, halts, censorship, fee spikes, or RPC failures can impair operation and availability.
4. **Smart contract and economic risk:** vulnerabilities, MEV/front-running, griefing/spam, or escrow edge cases may exist.
5. **"Erasure" is limited:** on-chain "erase" cannot remove historical chain data, archives, indexers, backups, or third-party caches.
6. **Cryptography may fail over time:** implementation flaws, new attacks, or quantum advances may weaken or break assumptions (including post-quantum schemes).

Limitation of liability. To the maximum extent permitted by law, in no event shall the authors or contributors be liable for any direct or indirect losses or damages (including loss of data, confidentiality, profits, or business opportunities) arising from use of, inability to use, or reliance on these materials, under any legal theory, even if advised of the possibility. Where limitations are restricted by applicable law, liability is limited to the minimum extent permitted.

No offer / no endorsement. This document is not an offer, solicitation, or recommendation regarding any asset, token, or investment.

Acknowledgments

This whitepaper synthesizes established cryptographic primitives and emerging DeSci insights. Any errors remain the responsibility of the authors.

References

[1] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.

[2] W. Entriken, D. Shirley, J. Evans, N. Sachs. *EIP-721: Non-Fungible Token Standard*, Ethereum Improvement Proposals, 2018. `https://eips.ethereum.org/EIPS/eip-721`.

[3] Ethereum community. *EIP-1193: Ethereum Provider JavaScript API*, 2019. `https://eips.ethereum.org/EIPS/eip-1193`.

[4] EIP authors. *EIP-6963: Multi Injected Provider Discovery*, 2023. `https://eips.ethereum.org/EIPS/eip-6963`.

[5] A. Langley, M. Hamburg, S. Turner. *RFC 7748: Elliptic Curves for Security*, IETF, 2016. `https://datatracker.ietf.org/doc/html/rfc7748`.

[6] Y. Nir, A. Langley. *RFC 8439: ChaCha20 and Poly1305 for IETF Protocols*, IETF, 2018. `https://www.rfc-editor.org/rfc/rfc8439.html`.

[7] CFRG. *draft-irtf-cfrg-xchacha: XChaCha: Extending the ChaCha20 Nonce*, IETF Datatracker, Internet-Draft. `https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-xchacha-03`.

[8] Frank Denis et al. *Libsodium Documentation: Sealed Boxes*. https://libsodium.gitbook.io/doc/public-key_cryptography/sealed_boxes.

[9] National Institute of Standards and Technology. *FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, 2015.

[10] NIST. *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)*, 2024. https://csrc.nist.gov/pubs/fips/203/final.

[11] NIST. *FIPS 204: Module-Lattice-Based Digital Signature Standard (ML-DSA)*, 2024. https://csrc.nist.gov/pubs/fips/204/final.

[12] NIST. *FIPS 205: Stateless Hash-Based Digital Signature Standard (SLH-DSA)*, 2024. https://csrc.nist.gov/pubs/fips/205/final.

[13] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schwabe, P. Seiler, D. Stehlé. *CRYSTALS–Kyber: A CCA-Secure Module-Lattice-Based KEM*, 2018.

[14] V. Lyubashevsky, L. Ducas, T. Prest et al. *CRYSTALS–Dilithium: Digital Signatures from Module Lattices*, 2018.

[15] D. Bernstein, A. Hülsing, S. Kölbl, et al. *SPHINCS+: Submission to the NIST Post-Quantum Cryptography Project*, 2019.

[16] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994.

[17] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 1996.

[18] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, C. Winnerlein. *BLAKE2: simpler, smaller, fast as MD5*, 2013.

[19] F. Díaz et al. Exploring the decentralized science ecosystem: insights on organizational structures, technology foundations, and funding mechanisms. *Frontiers in Blockchain*, 2025.

[20] A. Authors. Decentralized science (DeSci): definition, shared values, and guiding principles. 2024.

[21] A. Authors. Hidden Risks: The Centralization of NFT Metadata and Its Impact on Sustainability. arXiv:2408.13281, 2024.

[22] H. Li et al. Patent data access control and protection using blockchain technology. *Scientific Reports*, 2022.

[23] A. Peters. Blockchain-Based Intellectual Property Tokenization and Distributed Innovation Governance. 2025.

[24] G. M. Hickey et al. Web3 and the future of applied ecosystem governance. *Advances in Ecological Research*, 2023.