CIPNFT: A Post-Quantum Ciphertext NFT Protocol

Robust On-Chain Confidential IP Tokenization with Buyer-Safe Key Delivery

Decentralized Science Labs

February 2026

Abstract

Public ledgers provide globally verifiable provenance and programmable exchange, yet their transparency conflicts with intellectual property (IP) markets in which information must remain confidential while ownership and transferability remain public. This work presents CIPNFT (Cryptographic Information Protocol NFT, also Cryptographic Intellectual Property NFT, also "Cipher NFT"), a non-custodial protocol for *fully on-chain ciphertext NFTs.* Each token stores (i) a public title and (ii) an encrypted metadata payload up to tens of kilobytes, where confidentiality and integrity are provided by modern authenticated encryption with associated data (AEAD), and access control is realized through cryptographic key encapsulation rather than servers.

A central design objective is *long-horizon confidentiality.* On-chain ciphertext is permanently observable and can be harvested for "store-now, decrypt-later" attacks. CIPNFT therefore treats post-quantum (PQ) key-wrapping as a cornerstone: the Data Encryption Key (DEK) that decrypts the payload is wrapped to the owner (and later to a buyer) using the NIST-standardized module-lattice KEM ML-KEM (FIPS 203), while the payload itself uses 256-bit symmetric AEAD. This separation yields a protocol whose confidentiality can outlive the account-authentication cryptography of a particular chain: even if ledger-layer signatures are forged and token control is compromised, ciphertext confidentiality remains intact provided the PQ decapsulation secret is not compromised.

Encrypted asset exchange introduces a practical fairness gap: an EVM contract cannot efficiently verify that a seller delivered a correct DEK envelope for a buyer. CIPNFT resolves this via a buyer-safe two-step escrow protocol in which the seller posts the buyer-specific DEK envelope on-chain *before* payment is released; the buyer verifies correctness off-chain via an on-chain DEK commitment and only then finalizes the sale, atomically transferring the token and releasing escrow. CIPNFT also includes an on-chain policy layer in which versioned Terms-of-Service (TOS) text and a transparent fee schedule are stored on-chain and referenced at tokenization time. The result is an EVM-compatible data privacy primitive for confidential IP tokenization and decentralized science workflows.

1 Introduction

Non-fungible tokens (NFTs) provide a canonical representation of ownership for unique digital artifacts. Standard NFT metadata, however, is typically stored in cleartext or retrieved from public URLs. This is incompatible with workflows where the asset must remain confidential while still benefiting from public provenance and programmable transfer. Examples include: research protocols, experimental datasets, model weights, molecular designs, manufacturing recipes, CAD files, unreleased media masters, and licensing terms.

A common "unlockable content" pattern delegates confidentiality to centralized servers, access-control lists, or custodial key management, reintroducing a trusted operator who can censor, disappear, or change policy. CIPNFT targets a different point in the design space: *store ciphertext* *on-chain and enforce access via cryptography.* The protocol is non-custodial: only participants holding the relevant cryptographic secrets can decrypt.

Long-horizon confidentiality and post-quantum threat. Public ledgers are archival media: ciphertext and key envelopes written today can be collected indefinitely. If key-wrapping relies on classical public-key cryptography vulnerable to quantum attacks, an adversary can perform a delayed compromise ("store-now, decrypt-later"). CIPNFT therefore elevates PQ key encapsulation to a first-class requirement rather than an afterthought. The protocol's confidentiality layer is engineered to remain meaningful even if the underlying chain's account-authentication cryptography is later weakened.

Fair exchange gap. Encrypted assets also create a fairness problem: the seller must provide not only the token but also a correct decryption capability. On the EVM, verifying the correctness of a delivered key envelope is not feasible with native opcodes. CIPNFT resolves this via a two-step escrow protocol: deliver first, then finalize.

Contributions.

- A non-custodial ciphertext-NFT construction with fully on-chain encrypted metadata and public titles.
- A PQ-secure DEK handoff via ML-KEM (FIPS 203) combined with 256-bit symmetric AEAD for payload encryption.
- A buyer-safe escrow exchange separating key delivery from payment release, with correctness verified by a DEK commitment.
- An on-chain policy layer: versioned TOS text and a transparent fee schedule stored on-chain and referenced during tokenization.
- A browser-only reference client and a state-indexed listing/offer mechanism suitable for static hosting (no server).

2 System and Threat Model

2.1 Entities

We describe the protocol using two participants and a public ledger:

- Alice (current owner / seller) holds token t and can decrypt its payload.
- Bob (buyer) wishes to acquire t and obtain decryption capability.
- Contract C implements tokenization and escrow logic on an EVM-compatible chain.

2.2 Adversary

The adversary can:

- observe all on-chain state, calldata, and logs, and run unbounded offline computation;
- front-run and reorder transactions subject to consensus rules;

- act as a malicious seller or malicious buyer;
- *optionally* possess quantum computational capability sufficient to break classical public-key cryptography.

No trusted off-chain services are assumed. Client software (browser) is assumed correct but subject to ordinary endpoint risks (phishing, malicious extensions, malware).

2.3 Security goals

Let M be the plaintext metadata, C_M its ciphertext, and K the per-token DEK.

1. **Payload confidentiality:** Without K, C_M reveals no information about M beyond length.
2. **Integrity:** Any modification to ciphertext or key envelopes is detected (decryption yields $\perp$).
3. **Non-custodial access:** Only holders of the relevant secrets (owner key material or view key) can recover K.
4. **Buyer-safe exchange:** Escrowed funds cannot be released to the seller unless the buyer can verify decryptability.
5. **Post-quantum confidentiality:** Key-wrapping remains confidential against quantum adversaries under standard assumptions for ML-KEM, enabling long-horizon secrecy for on-chain archives.

2.4 Inherent limitations

- **No revocation:** any party who learns M or K can retain a copy.
- **No historical erasure:** current-state scrubbing cannot delete historical chain data from archives.
- **Verification gap:** the contract cannot feasibly verify ML-KEM decapsulation or AEAD decryption on-chain.

3 Cryptographic Primitives

3.1 Notation

- $\|$ denotes concatenation.
- keccak256$(\cdot)$ denotes Keccak-256 (on-chain hash).
- $\mathrm{AEAD.Enc}_K(N, M, A)$ and $\mathrm{AEAD.Dec}_K(N, C, A)$ denote AEAD encryption/decryption with associated data A.
- A KEM provides $\mathrm{KeyGen}, \mathrm{Encaps}, \mathrm{Decaps}$ producing ciphertext ct and shared secret ss.

3.2 Payload encryption (AEAD)

CIPNFT encrypts metadata using XChaCha20-Poly1305, an AEAD with 256-bit keys and 192-bit nonces. Associated data binds ciphertext to chain and contract identity:

$$A \leftarrow \mathrm{encode}(\mathrm{chainId}) \parallel \mathrm{encode}(C)$$

and encryption produces:

$$C_M \leftarrow N \parallel \mathrm{AEAD.Enc}_K(N, M, A)$$

where N is stored alongside ciphertext.

3.3 DEK commitment

The contract stores a DEK commitment:

$$h \leftarrow \mathrm{keccak256}(K)$$

Any party who recovers a candidate $\hat{K}$ checks $\mathrm{keccak256}(\hat{K}) = h$ before using it.

3.4 Post-quantum DEK wrapping (ML-KEM)

CIPNFT wraps the DEK K to an owner or buyer using ML-KEM (FIPS 203). Let the recipient's encapsulation key be ek_R and decapsulation key be dk_R.

Wrap.

1. $(ct, ss) \leftarrow \mathrm{ML\text{-}KEM.Encaps}(ek_R)$.
2. $K_w \leftarrow \mathrm{KDF}(ss, \texttt{"CIPNFT-DEK-WRAP"} \parallel \mathrm{chainId} \parallel C)$.
3. $W \leftarrow \mathrm{AEAD.Enc}_{K_w}(N_w, K, A)$ with fresh nonce N_w.
4. Output envelope $E_R \leftarrow ct \parallel N_w \parallel W$.

Open.

1. Parse E_R as (ct, N_w, W).
2. $ss \leftarrow \mathrm{ML\text{-}KEM.Decaps}(dk_R, ct)$.
3. $K_w \leftarrow \mathrm{KDF}(ss, \cdot)$ and recover $\hat{K} \leftarrow \mathrm{AEAD.Dec}_{K_w}(N_w, W, A)$.
4. Accept iff $\mathrm{keccak256}(\hat{K}) = h$.

3.5 View key wrapping

An optional view key v enables third-party decryption without changing ownership. The client derives $K_v \leftarrow \mathrm{Hash}_{256}(v)$ and wraps K under K_v with AEAD. Security reduces to the entropy of v; CIPNFT generates v as 32 random bytes and provides export tooling.

4 On-Chain State and Interfaces

Each token t stores:

- **Public title** title_t (bounded length / bounded word count).
- **Ciphertext** $\text{metaCipher}_t = C_M$ (up to a size bound, e.g., 64 KiB).
- **Owner envelope** $\text{ownerEncDEK}_t = E_{\text{owner}}$ (PQ envelope as in Section 3.4).
- **View wrapper** viewWrap_t (optional AEAD-wrapped DEK under view key).
- **Commitment** $\text{dekHash}_t = h$.
- **Policy version** tosVersionOf_t, the on-chain TOS version referenced at tokenization.

The contract maintains:

- **Listings:** per-token open flag and price in native currency.
- **Offers:** per (t, buyer) escrowed payment and expiry.
- **Deliveries:** seller-posted delivered envelopes per (t, buyer).
- **State indexes:** enumerators for listings, offers, and owned tokens to support serverless UIs without event scans.
- **On-chain policy text:** TOS text stored as versioned on-chain strings; tokenization is gated by acceptance of the current version.
- **Fee schedule:** flat fee plus per-byte fee stored on-chain and referenced at tokenization.

Non-custodial property. No on-chain state includes private keys, decapsulation keys, view keys, or plaintext metadata. The only decryption capability is represented by encrypted envelopes and ciphertext.

5 Protocols (Alice and Bob)

5.1 Tokenization ("Encrypt and Tokenize")

Alice prepares plaintext metadata M and a public title title. The client:

1. samples random DEK $K \leftarrow \{0,1\}^{256}$ and nonce N,
2. computes ciphertext $C_M \leftarrow N \parallel \text{AEAD.Enc}_K(N, M, A)$,
3. computes commitment $h \leftarrow \text{keccak256}(K)$,
4. wraps K to Alice's PQ encapsulation key ek_A to obtain envelope E_A,
5. optionally produces view wrapper viewWrap.

The tokenization transaction stores $(\text{title}, C_M, E_A, h, \text{viewWrap}, \text{tosVersion})$ on-chain, and pays the on-chain fee determined by plaintext size.

Algorithm 1 Tokenization (Alice)

```
Require: Plaintext metadata $M$, $|M| \leq 64$ KiB; public title title
Require: On-chain acceptance recorded for current TOS version
1: Sample $K \leftarrow \{0,1\}^{256}$, $N \leftarrow \{0,1\}^{192}$
2: $A \leftarrow \text{encode}(\text{chainId}) \parallel \text{encode}(C)$
3: $C_M \leftarrow N \parallel \text{AEAD.Enc}_K(N, M, A)$
4: $h \leftarrow \text{keccak256}(K)$
5: $E_A \leftarrow \text{PQWrap}(ek_A, K)$                    ▷ ML-KEM + AEAD wrap
6: Optional: generate view key $v$ and viewWrap
7: Call on-chain tokenize storing $(\text{title}, C_M, E_A, h, \text{viewWrap}, \text{tosVersion})$
```

5.2 Listing

The owner lists t for price p by setting $(\text{open} = \text{true}, \text{price} = p)$ in on-chain listing state. Delisting sets open = false. Listings are enumerated via on-chain state indexes to support browser-only discovery.

5.3 Offer (Bob)

To buy at price p, Bob escrows p by calling `createOffer` with `msg.value = p` and an expiry time. The offer is stored in state and is enumerated via on-chain indexes (no event scanning required for discovery).

5.4 Buyer-safe exchange: deliver then finalize

Step 1 (Alice): deliver envelope. Alice recovers K locally by opening ownerEncDEK_t with her PQ decapsulation key dk_A and checking $\text{keccak256}(K) = h$. She then wraps K to Bob's PQ encapsulation key ek_B to obtain E_B and posts a delivery:

$$\texttt{deliverOffer}(t, B, E_B, \text{viewWrap}')$$

No payment is released at this step.

Step 2 (Bob): verify and finalize. Bob opens E_B with dk_B to recover $\hat{K}$, checks $\text{keccak256}(\hat{K}) = h$, optionally test-decrypts C_M, and then calls:

$$\texttt{finalizeOffer}(t)$$

This transaction atomically updates the owner envelope, transfers the token, and releases escrowed payment to the seller.

Claim 1 (Buyer payout safety)**.** *Assuming correct contract execution, a seller cannot receive a buyer's escrowed payment for token t unless the buyer calls `finalizeOffer` for their offer.*

Proof sketch. Escrow is locked in contract state on `createOffer`. The only code path releasing escrow to the seller is `finalizeOffer`, which is callable only by the buyer address. Other paths (cancel/refund) return escrow to the buyer and delete offer state. Thus seller payout requires buyer-controlled finalization. □

6 Security Discussion

6.1 Confidentiality under classical and quantum adversaries

CIPNFT separates confidentiality into two layers:

1. **Symmetric layer:** payload ciphertext C_M is protected by 256-bit AEAD.

2. **Public-key layer:** DEK K is protected by PQ KEM wrapping (ML-KEM).

The public chain reveals ciphertext length and timing, but not plaintext.

Why symmetric AEAD remains viable. Quantum search (Grover) provides a quadratic speedup for brute force; using 256-bit symmetric keys provides a conservative margin (idealized cost $\sim 2^{128}$ for exhaustive search), making the public-key layer the dominant long-horizon risk.

6.2 Ledger compromise vs. ciphertext confidentiality

A distinctive property of CIPNFT is that confidentiality can remain meaningful even if ledger-layer authentication is later weakened.

Claim 2 (Confidentiality under signature compromise)**.** *Suppose the chain's account-authentication scheme is compromised so that an adversary can forge transactions as arbitrary addresses. If (i) the ML-KEM decapsulation secret dk of the rightful decryptor is not compromised and (ii) no view key is disclosed, then the adversary cannot recover the DEK K from on-chain state with non-negligible advantage beyond breaking ML-KEM or the symmetric AEAD.*

Proof sketch. Forging transactions enables control-plane actions (transfer, list/delist, state updates), but does not yield access to dk, which never appears on-chain. Recovering K from the envelope requires decapsulation and AEAD opening using keys derived from dk; absent dk (and absent view key), the adversary's advantage reduces to breaking ML-KEM confidentiality or AEAD security. Therefore ciphertext confidentiality is cryptographically insulated from ledger authentication. □

Interpretation. The claim does *not* assert that token control is safe under signature compromise—only that the ciphertext confidentiality layer can outlive the chain's authentication assumptions. This is the sense in which the protocol's privacy layer can be "future-proof": encrypted artifacts can remain confidential as archival data even if the surrounding ledger ecosystem evolves or degrades.

6.3 Integrity and correctness

Payload integrity follows from AEAD. Envelope integrity follows from AEAD within the PQ wrap. Correctness of seller delivery is verified off-chain via $h = \text{keccak256}(K)$, and buyer-safe finalization prevents payment release without buyer verification.

6.4 View key considerations

A view key provides intentional broad decryption. If v is low-entropy, it can be brute-forced offline. CIPNFT therefore generates v as 32 random bytes and treats it as a cryptographic secret.

7 Implementation and Engineering Notes

State indexing (no event scanning). The reference contract stores explicit on-chain indexes for listed tokens, owned tokens, and offers, enabling fully static UIs to discover protocol state by constant-time indexed reads rather than event scans.

State scrubbing. The contract supports incremental clearing of large byte arrays from *current* state to reduce ongoing storage footprint. This is not historical erasure.

Portability and longevity. Ciphertext C_M, commitment h, and envelopes E are ledger-agnostic objects built from standardized cryptography. They can be migrated across ledgers or archived independently of EVM compatibility. This portability underpins the claim that CIPNFT privacy semantics may outlive a particular chain: even if the tokenization substrate changes, the ciphertext and PQ envelopes remain meaningful.

8 Applications

CIPNFT enables:

- confidential IP tokenization for trade secrets and licensing;
- decentralized science artifact escrow (protocols, datasets, designs) with public provenance;
- verifiable handoff of proprietary assets without custodial infrastructure;
- selective disclosure via view keys for auditors, collaborators, or licensees.

9 Limitations and Future Work

- **On-chain verifiability:** efficient zero-knowledge proofs could allow a seller to prove that a delivered envelope opens to a DEK matching h without revealing K.
- **Hybrid envelopes:** deriving wrap keys from both classical and PQ shared secrets to hedge algorithmic risk.
- **PQ signatures / account models:** integrating PQ signature verification (e.g., via account abstraction) so both confidentiality and control become PQ-resilient.
- **Recipient privacy:** reducing linkability of offers and deliveries via stealth addressing or privacy layers.

10 Conclusion

CIPNFT provides a non-custodial ciphertext-NFT primitive for confidential IP tokenization on public ledgers. The protocol anchors long-horizon secrecy in post-quantum key encapsulation for DEK transfer, while using high-security symmetric AEAD for payload encryption. A buyer-safe two-step escrow mechanism resolves the on-chain verification gap by conditioning payout on buyer verification. Together, these components yield a robust EVM-compatible data privacy protocol whose confidentiality semantics can remain meaningful even under adverse future cryptographic conditions.

References

[1] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. https://bitcoin.org/bitcoin.pdf.

[2] G. Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger (Yellow Paper)*. 2025 (Shanghai version). https://ethereum.github.io/yellowpaper/paper.pdf.

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

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

[5] P. Rogaway. Authenticated-Encryption with Associated-Data. In *Proceedings of ACM CCS 2002 (Workshop on Security)* (AEAD formulation), 2002. https://web.cs.ucdavis.edu/~rogaway/papers/ad.pdf.

[6] D. J. Bernstein. ChaCha, a variant of Salsa20. In *Workshop Record of SASC 2008*, 2008. https://cr.yp.to/chacha/chacha-20080128.pdf.

[7] D. J. Bernstein. The Poly1305-AES message-authentication code. In *Fast Software Encryption (FSE 2005)*, Springer, 2005. https://cr.yp.to/mac/poly1305-20050329.pdf.

[8] F. Denis et al. *XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305*. Internet-Draft, IRTF CFRG, 2020. https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-xchacha-03.

[9] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. DOI: https://doi.org/10.1137/S0097539795293172.

[10] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of STOC 1996*, ACM, 1996. DOI: https://doi.org/10.1145/237814.237866.

[11] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of STOC 2005*, ACM, 2005. https://cims.nyu.edu/~regev/papers/qcrypto.pdf.

[12] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT 2010*, Springer, 2010. DOI: https://doi.org/10.1007/978-3-642-13190-5_1. ePrint: https://eprint.iacr.org/2010/269.

[13] C. Peikert. A Decade of Lattice Cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016. https://eprint.iacr.org/2015/939.

[14] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS–Kyber: A CCA-secure module-lattice-based KEM. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018. DOI: https://doi.org/10.1109/EuroSP.2018.00032. https://cryptojedi.org/papers/kyber-20170627.pdf.

[15] National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)*. Federal Information Processing Standards Publication 203, 2024. DOI: https://doi.org/10.6028/NIST.FIPS.203. https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.203.pdf.

[16] G. Alagic et al. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. NISTIR 8413, 2022. DOI: https://doi.org/10.6028/NIST.IR.8413. https://csrc.nist.gov/pubs/ir/8413/upd1/final.

[17] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. arXiv:1904.05234, 2019. https://arxiv.org/abs/1904.05234.