Five modes of ZK-based community chats around non-fungible tokens' owners

Oleksandr Kurbatov, Yaroslav Panasenko, Volodymyr Dubinin, and Yevhen Hrubiian

Distributed Lab, Kyiv, Ukraine

Abstract. We present Free-Delete, a censorship-resistant group-chat protocol whose membership is validated by non-fungible-token (NFT) ownership while user privacy ranges across *five* selectable modes. A single Groth16 circuit, anchored in a sparse Merkle tree of verifiable commitments, realises (i) Fully Anonymous messaging, (ii) Linkable Anonymous reputation building, (iii) Publicly Identified disclosure, (iv) Confidential end-to-end encryption, and (v) Rate-Limited Accountability that revokes keys on spam—*all* without moderator involvement or economic deposits. A black-paper prototype written in TypeScript, Circom 2, and Solidity achieves 0.35 s for register and 1.38 s for postMessage on consumer hardware; on-chain verification costs 3–5.5$\times 10^5$ gas per proof on Polygon. These results demonstrate that NFT-gated, stake-free, privacy-preserving communication can be deployed today on any EVM chain.

Keywords: NFT authentication, zero-knowledge proofs, decentralised chat, sparse Merkle tree, rate-limiting nullifiers, forward secrecy

1 Introduction

Mainstream chat platforms concentrate censorship power in the hands of a few administrators who can delete messages, ban members, or erase history. Communities that rely on public accountability—activist groups, decentralised-autonomous organisations (DAOs), research collectives—demand the opposite: an immutable yet privacy-respecting space where only bona-fide members may speak. Ownership of a particular NFT serves as a natural membership credential, but publishing that credential in the clear compromises anonymity and invites surveillance. Conversely, hiding completely invites spam and Sybil abuse.

Free-Delete closes this gap. Each wallet writes a commitment—its baby-JubJub public key hashed with the token identifier—into a sparse Merkle tree managed by an EVM contract. A single Groth16 circuit then lets the holder prove, in zero knowledge, that she controls (and still owns) exactly one such commitment while optionally exposing selected public flags. By toggling these flags, the same circuit offers five privacy tiers:

1. *Fully Anonymous* — unlinkable messages;
2. *Linkable Anonymous* — pseudonymous reputation without identity disclosure;
3. *Publicly Identified* — sender's key or address revealed;
4. *Confidential* — messages encrypted under epoch-ratcheted group keys;
5. *Rate-Limited Accountability* — stake-free RLN with key exposure on spam.

No moderator, coordinator, or deposit is required. Epoch-scoped nullifiers block replay, a per-epoch key schedule ensures forward secrecy, and a single trusted setup suffices for every mode and RLN degree. Our reference implementation compiles the largest circuit (54,520 constraints) in two seconds, generates proofs in under 1.4 s, and verifies them on-chain for roughly 400 k gas—well within the limits of inexpensive L2 networks.

The remainder of the paper is organised as follows. Section 2 reviews the cryptographic tools and prior art. Section 4 details the protocol flow and its five modes. Section 6

David C. Wyld et al. (Eds): CSIT, AMLA, IPDCA, NLPA, AIS, IPPR, SPTM – 2025
pp. 205 -219, 2025. CS & IT -CSCP 2025
DOI: 10.5121/csit.2025.151416

formalises the threat model and security guarantees. Implementation and performance metrics appear in Section 7. Section 8 outlines open problems—improved replay safety, higher-degree RLN, automated key rotation—and Section 9 summarises our contributions.

2 Preliminaries

Let $\mathsf{B} = \{0, 1\}$ and B^n a binary sequence with the length n. $\mathbb{F}_p$ is a finite field of primer order p. Let zkHash be the zk-friendly hash function that $\mathsf{zkHash} : \mathsf{B}^* \to \mathbb{F}_p$, while the hash_n be a regular cryptographic one-way function $\mathsf{hash}_n : \mathsf{B}^* \to \mathsf{B}^n$.

Let $\mathcal{T}$ be a Merkle Tree [1]. Each leave (element e) consists of a key-value pair (k, v). Merkle audit path $\mathsf{path}(e)$ is the shortest list of additional nodes that allows computing the root value $\mathsf{Root}_{\mathcal{T}}$:

$$\mathsf{proof}(e) = \begin{cases} \mathsf{MP} & \text{if } \mathsf{path}(e) \to \mathsf{Root}_{\mathcal{T}} \\ \mathsf{NMP} & \text{if } \mathsf{path}(e) \to \mathsf{Root}_{\mathcal{T}'} \neq \mathsf{Root}_{\mathcal{T}} \end{cases}$$

Let $\mathsf{S}(\mathsf{priv}^*, \mathsf{pub}^*, \mathsf{rel}^*)$ be a statement that sets the list of mathematical relations rel^* between private priv^* and public pub^* signals. Let $\mathsf{D} = \langle d_p, d_v \rangle$ be a tuple of keys needed for generating and proving zk-SNARK for the statement S using the trusted procedure[2].

$\mathsf{prove}(d_p, \mathsf{priv}^*, \mathsf{pub}^*) \to \mathsf{P}$ is a proving construction verified by the $\mathsf{verify}(d_v, \mathsf{pub}^*, \mathsf{P}) \to \mathsf{B}$, where 0 means the proof correctness, overwise 1.

Let $\mathsf{sig\_gen}(\mathsf{message}, \mathsf{sk})$ be an EdDSA signature generates algorithm top on baby jubjub curve[3], operating with $\mathsf{message}$ and a private key sk. $\mathsf{sig\_ver}(\mathsf{sig}, \mathsf{PK}, \mathsf{message}) \to \mathsf{bool}$ is the signature verification algorithm that takes $\mathsf{message}$, public key PK and signature sig as inputs and returns true or false value depending on the signature correctness.

2.1 Shamir's Secret Sharing

Shamir's Secret Sharing [4] is a cryptographic algorithm designed to divide a secret into multiple parts, giving each participant its unique part. To reconstruct the secret, a minimum number of parts is required. This scheme is also known as a (t, n)-threshold scheme, where t is the threshold number of parts needed to reconstruct the secret, and n is the total number of parts distributed.

Setup Let $s < p$ be the secret we want to share. Lets define a random polynomial $f(x)$ of degree $t - 1$ over $\mathbb{F}_p$:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1} \mod p$$

where $a_0 = s$ and $a_1, a_2, \ldots, a_{t-1}$ are randomly chosen coefficients from $\mathbb{F}_p$.

Shares distribution Each of the n participants is assigned a unique, non-zero value $x_i \in \mathbb{F}_p$, and they receive the corresponding share $(x_i, f(x_i))$.

Secret reconstruction At least t shares are needed to reconstruct the secret. Given t points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)\}$, we can use Lagrange interpolation to find the polynomial $f(x)$:

$$f(x) = \sum_{j=1}^{t} L_j(x) y_j \mod p, \quad L_j(x) = \prod_{\substack{1 \le m \le t \\ m \neq j}} \frac{x - x_m}{x_j - x_m} \mod p$$

The secret s is then $a_0 = f(0)$.

3 Related Work

Research on privacy–preserving social systems may be divided into three lines: anonymous signalling primitives, vote-oriented credential frameworks, and token-gated chat protocols.

3.1 Anonymous Signalling Primitives

Semaphore v1/v2 realises set membership proofs with a single Groth16 SNARK, yielding unlinkable "signals" but neither economic spam deterrence nor forward secrecy [5]. **Rate Limiting Nullifiers (RLN)** augments Semaphore with a bond–slash mechanism: surpassing a degree-d bound in an epoch discloses the user's secret and forfeits the stake [6]. The *Social Forest* prototype integrates RLN proofs into a live chat client, demonstrating practical latency yet confirming the usability cost of mandatory deposits and the privacy loss once a key is exposed.

3.2 Voting Frameworks and Credential Systems

MACI repurposes Semaphore for collusion-resistant voting but retains an off-chain coordinator that can still censor submissions. **Rarimo** formalises cross-chain NFT credentials, treating tokens as portable proofs of membership [8]. While powerful for gating, Rarimo delegates both spam control and message confidentiality to higher-level applications.

3.3 Token-Gated Chat Protocols

Several recent designs attempt end-to-end encrypted messaging restricted to token holders. **Push Protocol** introduces NFT-gated group channels on top of a custom P2P layer [10]. **Lens Protocol** implements encrypted publications whose keys are wrapped for holders of a designated NFT or ERC-20 asset [11]. **Waku v2** offers a modular, censorship-resistant transport already adopted by Status; token balance checks can be placed on its envelope topics [12]. None of these systems, however, provides a stake-free accountability mechanism or forward-secret unlinkability within the same universal SNARK circuit.

3.4 Free-Delete Algorithm's Distinct Contribution

Inspired by Rarimo's credential abstraction, the Free-Delete algorithm fuses dynamic token-based group formation with stake-free, forward-secret accountability:

- **Spam resistance** – a moderator quorum can trace an offending degree-d nullifier and penalise abuse without user deposits.
- **Censorship checks** – the tracing procedure requires threshold cooperation, precluding unilateral suppression.
- **Forward secrecy** – per-epoch keys keep past ciphertexts private even if a long-term key later leaks.

Thus Free-Delete is, to our knowledge, the first protocol that (i) forms anonymity sets via NFTs, (ii) eliminates economic friction in spam deterrence, and (iii) guarantees forward secrecy – closing the gap left by RLN and extending privacy beyond existing token-gated chat systems.

4 Protocol

Overview. FREE-DELETE lets a wallet that currently holds a qualifying token speak in a shared chat while keeping its owner hidden. The wallet first stores a commitment (a public key hashed with its token ID) in a sparse Merkle tree managed by the contract. Every message carries one Groth16 proof that shows (i) the sender controls that commitment, (ii) the token has not left the wallet, and (iii) any chosen rate-limit. The contract verifies the proof, checks a per-epoch nullifier to block replays, and then emits the message event; no administrator can erase or veto it. Five privacy modes: anonymous, linkable, identified, confidential, and rate-limited – are selected with public inputs, so a single trusted setup secures the entire system.

4.1 Keypair generation

First, the user needs to generate the baby jubjub key pair $(\mathsf{sk}, \mathsf{PK})$ that will be used to confirm their actions (sending messages). These keys will be used to verify signatures in circuits more efficiently.

4.2 Creating the verifiable commitment

After generating the key pair, the user must prove the NFT ownership and add the corresponding commitment to the tree. The structure of the data in the commitment is the following:

$$\mathsf{vc} = (\mathsf{contract\_id}, \mathsf{nft\_id}, \mathsf{owner\_eoa}, \mathsf{PK}, \mathsf{timestamp})$$

Then, the user creates the transaction that initiates adding the credential to the tree and includes the following proof π:

$$\left|\begin{array}{c} \textbf{pub\_signals:} \\ \mathsf{contract\_id}, \mathsf{nft\_id}, \mathsf{owner\_eoa}, \mathsf{vc\_id}, \mathsf{T}_c \\ \textbf{priv\_signals:} \\ \mathsf{PK}, \mathsf{timestamp} \\ \textbf{circuit\_logic:} \\ \mathsf{vc\_id} = \mathsf{zkHash}(\mathsf{contract\_id}, \mathsf{nft\_id}, \mathsf{owner\_eoa}, \mathsf{PK}, \mathsf{timestamp}) \\ \wedge \\ \mathsf{timestamp} \leq \mathsf{T}_c \end{array}\right| \tag{1}$$

The transaction is signed by the user and calls the method of FREE-DELETE contract to add the commitment vc to the tree $\mathcal{T}$ with the index $\mathsf{vc\_id}$. The contract:

1. Verifies the TX signature according to the EOA that initiated it
2. Verifies that EOA is an owner of the declared NFT on the declared contract
3. Verifies a zero-knowledge proof π according to the statement mentioned above

Let's note that the user can define any timestamp in the credential that is lower than the value of the current blockchain timestamp. Digging deeper, the user passes the current time value as a public signal and proves that the timestamp inside the commitment is less than the declared value. The smart contract checks that the time value does not exceed the timestamp of the blockchain itself ($\mathsf{T_c}$ must be less than $\mathsf{block.timestamp}$).

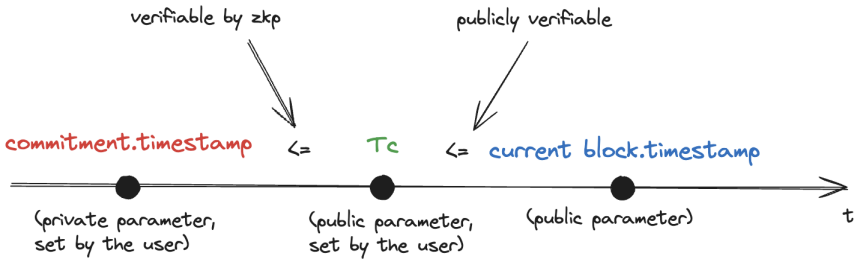

Fig. 1. Verifiable commitment timestamp rules

The commitment vc is added to the tree if all verifications are performed correctly. At this stage, it is worth noting that anyone can see which user created a commitment and what specific data they linked it to (except the public key for managing the commitment) — this information is validated by the contract and, therefore, is available to any party that owns the state machine. However, all further operations with the commitment are performed in hidden form, as described in the next section.

4.3 Authentication

When users want to connect to the chat and write messages, they need to prove the validity of their credentials. The chat settings define the validity rules. These criteria include the address(es) of the NFT contract, the list of token identifiers (if we need to provide chat access only to a limited set of NFT owners, not to all of them), and expiration time bounds (the owner of the particular NFT can be changed).

Depending on the working mode, there are some modifications in the proving mechanism (the mechanism of generation of the proof for message sending), but we can formalize the approach as follows:

1. User generates the signature over the message using their commitment key:

$$\mathsf{sig\_gen}(\mathsf{message}, \mathsf{sk}) \rightarrow \mathsf{sig}$$

2. User generate the proof π for the following statement:

$$\left|\begin{array}{c} \textbf{pub\_signals:} \\ \mathsf{contract\_id}, \mathsf{Root}_{\mathcal{T}}, \mathsf{message}, \mathsf{T}_t \\ \textbf{priv\_signals:} \\ \mathsf{vc}^*, \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)), \mathsf{sig} \\ \textbf{circuit\_logic:} \\ \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)) \rightarrow \mathsf{Root}_{\mathcal{T}} \\ \wedge \\ \mathsf{sig\_ver}(\mathsf{sig}, \mathsf{vc.PK}, \mathsf{message}) \rightarrow \mathsf{true} \\ \wedge \\ \mathsf{vc.timestamp} > \mathsf{T_t} \end{array}\right| \tag{2}$$

3. User sends the proof to the chat service

In other words, when connecting to the chat and sending messages, the user states that they once confirmed ownership of an NFT from the collection (an existing commitment is such a confirmation), and this confirmation has not yet expired (by the relation

$\mathsf{vc.timestamp} > \mathsf{T_t}$ we prove that the timestamp in the commitment exceeds the minimal threshold defined by the chat service).

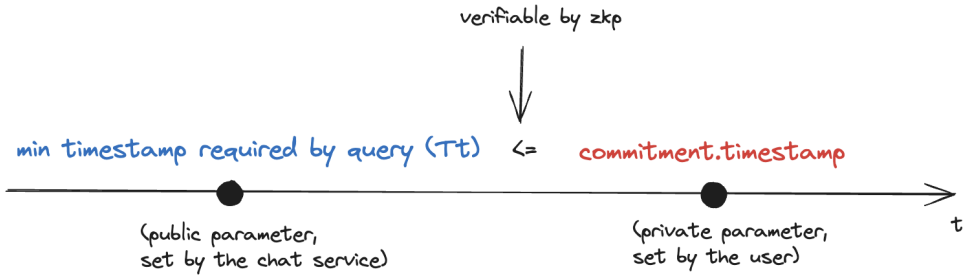

Fig. 2. Rules for satisfying a chat's query

The realization of the chat architecture is the topic of a separate paper; the approach proposed here should support options from centralized chat service to some federated social networks.

5 Modes

Every message in FREE-DELETE moves through the same three-step pipeline: the sender (1) chooses a set of public flags, (2) generates a single Groth16 proof that binds those flags to her live token commitment, and (3) relays the ciphertext plus proof to the contract, which verifies and logs it. Varying the flag vector does not change the circuit – it only flips which parts of the proof's public input are revealed. The resulting flag profiles are the protocol's *modes of operation*. The next subsections define each mode and the guarantees it offers.

5.1 Fully Anonymous Mode

This mode allows the user to be fully anonymous without even connecting messages sent by the same chat participant. It's the simplest and efficient approach that allows complete decentralized chaos without limitations and prohibitions.

To operate in Fully Anonymous Mode, the user should use the authentication approach mentioned in Section 3.3.

5.2 Linkable Anonymous Mode

This mode presumes users to be anonymous but connects message history to the profiles they were sent from. It allows private profiles to collect a provable reputation without disclosing a particular person who stays behind.

This scheme has a small modification extending the auth method with the nullifier constructed top on the sk. It modifies the algorithm in the following way:

1. User generates the signature over the message using their commitment key:

$$\mathsf{sig\_gen}(\mathsf{message}, \mathsf{sk}) \rightarrow \mathsf{sig}$$

2. User generates the nullifier as

$$\mathsf{zkHash}(\mathsf{sk}) \rightarrow \mathsf{nullifier}$$

3. User generate the proof π for the following statement:

$$\left|\begin{array}{c} \textbf{pub\_signals:} \\ \mathsf{contract\_id}, \mathsf{Root}_{\mathcal{T}}, \mathsf{message}, \mathsf{nullifier}, \mathsf{G}, \mathsf{T_t} \\ \textbf{priv\_signals:} \\ \mathsf{vc}^*, \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)), \mathsf{sig}, \mathsf{sk} \\ \textbf{circuit\_logic:} \\ \mathsf{vc.contract\_id} = \mathsf{contract\_id} \\ \wedge \\ \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)) \rightarrow \mathsf{Root}_{\mathcal{T}} \\ \wedge \\ \mathsf{sig\_ver}(\mathsf{sig}, \mathsf{vc.PK}, \mathsf{message}) \rightarrow \mathsf{true} \\ \wedge \\ \mathsf{vc.PK} = \mathsf{sk} \cdot \mathsf{G} \\ \wedge \\ \mathsf{nullifier} = \mathsf{zkHash}(\mathsf{sk}) \\ \wedge \\ \mathsf{vc.timestamp} > \mathsf{T_t} \end{array}\right| \tag{3}$$

4. User sends the proof to the chat service

This approach allows the aggregation of all messages with the same nullifier without revealing the owner of the appropriate secret key and is therefore the recommended configuration for Linkable Anonymous Mode.

5.3 Publicly Identified Mode

It's possible to make totally public and traceable all message senders (In this case, users have nothing to hide). That's not the main idea of the protocol, but no significant modifications are required to make it possible. The scheme of relations changes to:

$$\left|\begin{array}{c} \textbf{pub\_signals:} \\ \mathsf{contract\_id}, \mathsf{Root}_{\mathcal{T}}, \mathsf{message}, \mathsf{PK}, \mathsf{T_t} \\ \textbf{priv\_signals:} \\ \mathsf{vc}^*, \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)), \mathsf{sig} \\ \textbf{circuit\_logic:} \\ \mathsf{vc.contract\_id} = \mathsf{contract\_id} \\ \wedge \\ \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)) \rightarrow \mathsf{Root}_{\mathcal{T}} \\ \wedge \\ \mathsf{PK} = \mathsf{vc.PK} \\ \wedge \\ \mathsf{sig\_ver}(\mathsf{sig}, \mathsf{PK}, \mathsf{message}) \rightarrow \mathsf{true} \\ \wedge \\ \mathsf{vc.timestamp} > \mathsf{T_t} \end{array}\right| \tag{4}$$

It's not so difficult, right? This mode doesn't allow the non-community participant to send the message, so full verifiability and transparency are met. Using an EOA identifier instead of PK is also possible. It's up to you.

5.4 Confidential Messaging Mode

All previous approaches presumed messages transfer in the open form, making them publicly auditable. This approach doesn't suit private organizations, where the message content must only be available to community members.

For organizing confidential messaging, the most efficient approach is for the first chat participant to generate the secret key and then encrypt it using other users' public keys (asymmetric encryption). Users can use new keypairs for that and authenticate them using the signature generated by the commitment's key.

All processes with expiration and revocation of the encryption key depend on the chat members. They have enough to use different governance protocols that are managed by their commitment keys.

5.5 Rate-Limited Accountability Mode

Sometimes, it makes sense to ban spammers. But again, with no administrators, only using defined chat rules and cryptography. We can use rate-limiting nullifiers for this purpose[7]. It extends our verifiable commitment to the new field

$$\mathsf{sk\_null} = \mathsf{zkHash}(\mathsf{sk})$$

At the same time, let's define the linear polynomial $\mathsf{f}(\mathsf{x}) = \mathsf{a}\mathsf{x} + \mathsf{b}$, meaning the secret b can be reconstructed by having evaluations in only two points.

When the user wants to send a message, it should calculate the following point:

$$\mathsf{x} = \mathsf{zkHash}(\mathsf{message}), \quad \mathsf{y} = \mathsf{f}(\mathsf{x})$$

Then, the user sends this share with its validity proof π.

$$\left|\begin{array}{c} \textbf{pub\_signals:} \\ \mathsf{contract\_id}, \mathsf{Root}_{\mathcal{T}}, \mathsf{message}, \mathsf{topic}, \mathsf{y}, \mathsf{T_t} \\ \textbf{priv\_signals:} \\ \mathsf{vc}^*, \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)), \mathsf{sig}, \mathsf{sk} \\ \textbf{circuit\_logic:} \\ \mathsf{vc.contract\_id} = \mathsf{contract\_id} \\ \wedge \\ \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)) \rightarrow \mathsf{Root}_{\mathcal{T}} \\ \wedge \\ \mathsf{sig\_ver}(\mathsf{sig}, \mathsf{vc.PK}, \mathsf{message}) \rightarrow \mathsf{true} \\ \wedge \\ \mathsf{y} = (\mathsf{zkHash}(\mathsf{topic}, \mathsf{sk})) \cdot \mathsf{zkHash}(\mathsf{message}) + \mathsf{sk} \\ \wedge \\ \mathsf{sk\_null} = \mathsf{zkHash}(\mathsf{sk}) \\ \wedge \\ \mathsf{vc.timestamp} > \mathsf{T_t} \end{array}\right| \tag{5}$$

If the user wants to send another message for the same topic, its secret key will be corrupted. For example, we have two messages $\mathsf{m}_1, \mathsf{m}_2$ and corresponding shares:

$$\mathsf{x}_1 = \mathsf{zkHash}(\mathsf{m1}), \mathsf{y}_1 = \mathsf{zkHash}(\mathsf{topic}, \mathsf{sk}) \cdot \mathsf{zkHash}(\mathsf{m1}) + \mathsf{sk}$$

$$\mathsf{x}_2 = \mathsf{zkHash}(\mathsf{m2}), \mathsf{y}_2 = \mathsf{zkHash}(\mathsf{topic}, \mathsf{sk}) \cdot \mathsf{zkHash}(\mathsf{m2}) + \mathsf{sk}$$

Everyone can reconstruct the polynomial by:

$$f(x) = y_1 \cdot \frac{x - x_2}{x_1 - x_2} + y_2 \cdot \frac{x - x_1}{x_2 - x_1}$$

Wrapping $\mathsf{zkHash}(\mathsf{topic}, \mathsf{sk})$ to t we receive:

$$f(x) = (t \cdot x_1 + \mathsf{sk}) \cdot \frac{x - x_2}{x_1 - x_2} + (t \cdot x_2 + \mathsf{sk}) \cdot \frac{x - x_1}{x_2 - x_1}$$

$$f(x) = \frac{tx_1x - tx_1x_2}{x_1 - x_2} + \mathsf{sk} \cdot \frac{x - x_2}{x_1 - x_2} + \frac{tx_2x - tx_1x_2}{x_2 - x_1} + \mathsf{sk} \cdot \frac{x - x_1}{x_2 - x_1}$$

$$f(x) = \mathsf{sk} \cdot \left(\frac{x - x_2}{x_1 - x_2} - \frac{x - x_1}{x_1 - x_2}\right) + t \cdot \left(\frac{x_1x - x_1x_2}{x_1 - x_2} - \frac{x_2x - x_1x_2}{x_1 - x_2}\right)$$

$$f(x) = \mathsf{sk} + t \cdot x$$

So we see that the polynomial was reconstructed correctly with revealing the user's secret key. This approach doesn't limit the user's actions but allows them to steal an identity if the user violates the chat rules.

6 Security Model

6.1 Threat Model

All adversarial algorithms $\mathcal{A}$ run in PPT relative to the security parameter λ.

- **Observer** — full read access to the ledger and mempool; attempts deanonymisation.
- **Front-running relayer** — controls a mempool gateway; reorders, inserts, or delays transactions within latency Δ_{net}.
- **Colluding moderators** — any subset $\mathcal{M} \subseteq \{1, \ldots, M\}$ with $|\mathcal{M}| = k < M$; tries to censor messages or break privacy during tracing.

We assume chain liveness and eventual finality; *no* honest-majority of miners is needed for privacy or replay safety.

6.2 Formal Security Goals

For each property P we define an experiment $\mathsf{Exp}^{\mathsf{P}}_{\text{FREE-DELETE},\mathcal{A}}(\lambda)$; the protocol is secure if every PPT adversary wins with probability $\leq \mathsf{negl}(\lambda)$.

Unlinkability. $\mathcal{A}$ chooses (U_0, U_1); the challenger flips $b \leftarrow \{0, 1\}$, has U_b produce T_b, and returns T_b plus a random decoy T_{1-b}. $\mathcal{A}$ outputs b'.

$$\Pr\left[\mathsf{Exp}^{\text{Unlink}}_{\text{FREE-DELETE}} = 1\right] = \Pr[b' = b] \;\leq\; \tfrac{1}{2} + \mathsf{negl}(\lambda).$$

Censorship resistance. The challenger submits a valid T at height h. $\mathcal{A}$ controls the mempool. Success iff T is absent from all finalised blocks $< h + \Delta_E$.

Replay resistance (epoch-scoped). $\mathcal{A}$ outputs a transaction T^* *bit-for-bit* identical to a previously confirmed T with the *same epoch index* $e = \lfloor \texttt{block.number}/\mathsf{EPOCH\_LEN} \rfloor$. Game returns 1 if the contract accepts T^*.

Transfer safety. $\mathcal{A}$ produces a proof that is valid at generation time t_0 but submits it after ownership of `tokenId` changed ($t_1 > t_0$). Game returns 1 if the proof is accepted.

Forward secrecy. $\mathcal{A}$ learns the long-term key $\mathtt{SK}$ at t_1 and is given a ciphertext C issued at $t_0 < t_1$. Advantage is the ability to distinguish C from random.

Spam accountability. $\mathcal{A}$ exceeds degree-d rate limit in epoch e. Trace outputs η. Game succeeds if (i) η links all violating messages and (ii) reveals no secret keys.

6.3 Mechanism Mapping

- **zk-SNARK proof** — enforces unlinkability and ownership.
- **Epoch counter and nullifier** $\mathtt{nf} = \mathsf{Hash}(\mathtt{sk} \,\|\, e \,\|\, \mathtt{flags})$ — prevents replay *within* an epoch.
- **Sparse Merkle leaf** $\lambda = \mathsf{Hash}(\mathtt{tokenId} \,\|\, \mathtt{owner} \,\|\, e_{\mathrm{cr}})$ — binds proofs to the owner present when the commitment was written.
- **Auto-advancing epochs** — deletion of any single coordinator; yields censorship resistance.
- **Hierarchical keys + double ratchet** — provide forward secrecy.
- **Threshold trace on** $\mathtt{nf}$ — realises spam accountability without identity loss.

6.4 Epoch-Based Ownership Validation

Epoch definition. Let $\mathsf{EPOCH\_LEN} \in \mathbb{N}$ be a global parameter (default 100 blocks ≈20 min on Ethereum). The current epoch is

$$e_{\mathrm{cur}} \;=\; \left\lfloor \frac{\mathtt{block.number}}{\mathsf{EPOCH\_LEN}} \right\rfloor.$$

Epochs advance deterministically; no sequencer or oracle can halt progress.

Dynamic ownership leaf. For every NFT $\mathtt{tokenId}$ the sparse Merkle tree stores

$$\lambda \;=\; \mathsf{Hash}(\mathtt{tokenId} \,\|\, \mathtt{owner} \,\|\, e_{\mathrm{cr}}),$$

where e_{cr} is the epoch when the leaf was last (re)written.

Public inputs of a proof. A message proof exposes $\{\,\mathtt{root},\, e_{\mathrm{cur}},\, \mathtt{tokenId},\, \mathtt{nf}\,\}$. The circuit enforces:

(i) $e_{\mathrm{cur}} = \lfloor \mathtt{block.number}/\mathsf{EPOCH\_LEN} \rfloor$;
(ii) valid Merkle path for λ;
(iii) on-chain check $\mathtt{ownerOf(tokenId)} = \mathtt{msg.sender}$.

Nullifier space. Nullifiers are epoch-scoped:

$$\mathtt{nf} = \mathsf{Hash}(\mathtt{sk} \,\|\, e_{\mathrm{cur}} \,\|\, \mathtt{flags}), \quad \mathtt{NFspace}[e] = \{\mathtt{nf}\}.$$

Hence, cross-epoch replays are impossible by construction.

Race-condition immunity. If a proof is generated at t_0 (owner=Alice) but submitted after a transfer at t_1 (owner=Bob), it fails because $\mathtt{ownerOf(tokenId)} = \mathtt{Bob} \neq \mathtt{Alice}$.

6.5 Mode-Specific Guarantees

Let the mode flag vector be $\langle \mathsf{anon}, \mathsf{link}, \mathsf{id}, \mathsf{rln}, d \rangle$. Guarantees differ as follows:

Mode	Unlink	Linkability	Identity	RLN	FS
Fully Anonymous	✔	✗	✗	✗	✔
Linkable Anonymous	✗	✔	✗	✗	✔
Public ID	✗	✔	✔	✗	✔
RLN–Accountable	✗	✔	✗	✔(d)	✔

Here FS stands for forward secrecy; it is offered in all modes because encryption keys ratchet per epoch.

7 Implementation

1. Depending on the implementation of the time-checking logic on the contracts and circuits, different strategies for user eligibility can be considered. In this paper, the static approach is described when, in the public domain, such as the Ethereum network, on the contract, a specific timestamp is statically stored, which specifies after which point the credentials are valid. Also, it is possible to define the validity of credentials for a specific period, for example, one month.
2. During the registration period, the Sparse Merkle Tree is used. Therefore, adding a tree, both index and value, is required. As index the vc_id is used, and value is constructed as follows: (zkHash(contract_id, nft_id, owner_eoa)) This allows for possible query proofs, where the participant can prove that some of these values are in the tree without revealing them. This can be used to filter out desired participants based on the data in the value.
3. As we mentioned at the start of the article — NFT isn't the single artifact the user can be connected to. There can be verifiable commitments connecting to certain balances, network activity, attestations, and other credentials and commitments.

Scope and status. The implementation described below represents the *black-paper* prototype of our protocol – a working reference build intended to make every design choice explicit and measurable. Although we anticipate further optimisation and security hardening in future revisions, this version is already complete enough to serve as a running example for the architectural discussion that follows.

7.1 Prototype Setup

Our reference implementation is written in TypeScript, Circom 2, and Solidity. The prover runs on commodity hardware; all figures below were obtained on a single–threaded CPU-only virtual machine with 12 GB RAM. On-chain components target the Ethereum Virtual Machine (EVM) and are tested on a Polygon fork for realistic gas accounting.

7.2 Developer Benchmarks

Unit-test timing. End-to-end integration tests executed on a MacBook Pro (Apple M3 Pro, 32 GB RAM) show:

– register — proof generation & contract call: ≈ 0.35 s

– postMessage — proof generation & contract call: 1.38 s
– failure path (expired credential): 1.18 s

These figures include witness generation, proof construction, local EVM execution, and assertion checks.

Circuit complexity.

Circuit	Constraints	Compile time	Proving-key size
PostMessage	54 520	2.0 s	23 MiB
VerifiableCommitment	1 040	0.5 s	0.5 MiB

Trusted-setup parameters. The largest circuit requires a Powers-of-Tau file with exponent 2^{16} (exact ID 16); download and Groth16 key generation complete in under 25 s on the same machine.

Gas profile. A ganache-based gas reporter records the following averages (optimizer enabled, 0.8.x compiler):

Contract / Method	Gas (avg)	USD (0.04 $/M gas)
AuthenticationStorage.register	529 123	0.021
Chat.postMessage	406 099	0.016

On Polygon (2–4× cheaper than Ethereum mainnet), these costs remain well below the practical threshold for mass usage.

On-chain verification. Groth16 verification costs dominate the gas expenditure of the smart contract. Empirical measurements on Polygon yield $3.0–5.5 \times 10^5$ gas per proof, which comfortably fits within a single block and is significantly cheaper than on mainnet Ethereum. This confirms that large-scale deployments remain economically viable.

7.3 Sparse Merkle Tree Index

All membership proofs rely on a sparse Merkle tree whose leaves are defined as

$$\mathsf{leaf} \;=\; \mathsf{zkHash}\big(\mathsf{vc}\big), \quad \mathsf{vc} = (\mathit{contract\_id}, \mathit{nft\_id}, \mathit{owner}, \mathit{pk}, \mathit{ts}).$$

The tree depth d is logarithmic in the number N of commitments ($d = \lceil \log_2 N \rceil$). For $d = 20$ ($N \approx 10^6$), a leaf update requires exactly 20 hash evaluations, yielding $O(\log N)$ gas growth. On Polygon, the marginal cost per insertion is below 40 k gas, allowing over 100,000 insertions for under 4 M gas—a feasible batch size for a single transaction by a trusted relayer.

7.4 Rate-Limited Accountability

In the RLN mode we fix the polynomial degree to $k = 1$; hence each epoch permits one message per commitment. Doubling (or more generally increasing) k would raise the circuit size quasi-linearly. Our choice of $k = 1$ keeps proof generation within the 3s budget while still deterring spam via key-extraction on double posting.

7.5 Key Management for Confidential Chat

The first participant derives a symmetric session key K and transmits K encrypted under the recipients' long-term public keys. At present, the implementation lacks (i) automated key rotation upon NFT transfer and (ii) forward secrecy. Section 8 outlines how integrating an HPKE envelope with a double-ratchet can solve both issues without enlarging the zero-knowledge circuit.

7.6 Timestamp Enforcement

Each commitment carries a timestamp $ts \leq T_c$ where T_c is the block time at creation. When a message is posted, the proof additionally enforces $ts > T_t$ for an application-defined threshold T_t. Choosing T_t as "now minus 30 days" yields an effective Sybil-mitigation window while avoiding the need for explicit revocation lists.

7.7 Spam Mitigation for Multi-NFT Holders

Because every NFT can register an independent commitment, a wealthy user could emit multiple messages per epoch. We propose two mitigations:

(a) *Commitment aggregation.* Map all NFTs owned by the same EOA to a single leaf by hashing the tuple $(owner, nft\_set)$.
(b) *Per-EOA quota.* Limit each distinct EOA to one active commitment, enforced at the contract layer.

Both strategies require only minor contract modifications and leave the proof system unchanged.

7.8 Moderator-(Dis)involvement

The design is fully cryptographic: neither the circuits nor the on-chain logic assumes any privileged moderator role. Potential collusion is thus confined to external applications that might withhold proofs; such issues belong in the threat model, not in protocol logic.

8 Discussion and Future Work

While our protocol establishes a solid foundation for censorship-resistant, privacy-preserving group chats gated by NFT ownership, several aspects remain unresolved. Below, we discuss these limitations and propose future research directions to address them, alongside additional ideas to enhance the protocol's robustness and applicability.

1. **Time-Bound Ownership and Replay Safety**
   Our use of timestamps and epochs for ownership validation and replay prevention (Section 6.4) is functional but could be improved. We plan to explore integrating a monotonic epoch counter or block-number-based public inputs to strengthen replay resistance. This will require updates to the circuit logic and security arguments, which we will formalize in future work.
2. **RLN Design and Generalization**
   The Rate-Limited Accountability Mode (Section 5) adopts a degree-1 polynomial ($k = 1$), limiting users to one message per epoch before risking key exposure. While this choice simplifies the circuit and maintains efficiency, generalizing to $k > 1$ could enhance flexibility. Future work will evaluate performance trade-offs for higher-degree polynomials and justify an optimal k based on use-case requirements.

3. **Confidential-Chat Key Management**
   The confidential messaging mode (Section 5) lacks automated key rotation and forward secrecy. We propose integrating an HPKE envelope with a double-ratchet mechanism to ensure:
   – Keys rotate automatically on NFT transfer via an ERC-721 `Transfer` hook.
   – Forward secrecy prevents former owners from decrypting future messages.
   Formal security proofs for this enhancement will follow in future work.
4. **Trusted-Setup Reuse**
   Our protocol uses a single Groth16 ceremony across all modes and RLN degrees (Section 7), leveraging an unchanged circuit. Future work will investigate whether multiple ceremonies or parameter sets could improve flexibility or security.
5. **Multiple-NFT Spam Mitigation**
   Currently, a user with multiple NFTs can send multiple messages per epoch (Section 7). We propose:
   (a) **Commitment aggregation**: Mapping all NFTs per EOA to one commitment.
   (b) **Per-EOA quota**: Restricting each EOA to a single active commitment.
   Both options will be tested for usability and security impacts.

In conclusion, while Free-Delete offers a promising framework, addressing these gaps and exploring new directions will significantly enhance its security, scalability, and adoption potential.

9 Summary

This paper presents Free-Delete, a novel protocol for constructing privacy-preserving, censorship-resistant group chats gated by non-fungible token (NFT) ownership. By leveraging a sparse Merkle tree and Groth16 zk-SNARKs, Free-Delete enables users to prove NFT ownership while supporting multiple privacy tiers—Fully Anonymous, Linkable Anonymous, Publicly Identified, and Rate-Limited Accountability—within a single circuit. The protocol extends the Rarimo social forest framework [9] by focusing on chat applications, transforming public on-chain ownership rights into leaves of an anonymous social tree. Key innovations include stake-free spam deterrence via rate-limiting nullifiers, dynamic membership tied to NFT transfers, and a moderator-free design that ensures immutable message delivery. Preliminary benchmarks demonstrate practical performance, with proof generation in under 3 seconds and gas-efficient verification on Polygon.

Despite these advances, challenges remain, such as optimizing key management for confidential chats, mitigating spam from multi-NFT holders, and providing comprehensive scalability analyses, as outlined in Section 8. Future work will address these gaps and explore broader applications, such as supporting soul-bound tokens. Free-Delete offers a robust foundation for decentralized, privacy-focused communication, with significant potential to empower communities in activist, research, and DAO contexts.

References

1. J. Baylina and M. Bellès, "Sparse Merkle Trees," iden3 Technical Publication, 2019. `https://docs.iden3.io/publications/pdfs/Merkle-Tree.pdf`
2. J. Groth, "Short Pairing-Based Non-interactive Zero-Knowledge Arguments," in *ASIACRYPT 2010*, LNCS 6477. `https://www.iacr.org/archive/asiacrypt2010/6477323/6477323.pdf`
3. J. Baylina and M. Bellès, "EdDSA for Baby JubJub Elliptic Curve with MiMC-7 Hash," iden3 Whitepaper, 2019. `https://iden3-docs.readthedocs.io/en/latest/_downloads/a04267077fb3fdbf2b608e014706e004/Ed-DSA.pdf`

4. A. Shamir, "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 11, 1979. https://dl.acm.org/doi/10.1145/359168.359176
5. Semaphore Team, "Semaphore: A Zero-Knowledge Protocol," 2022. https://semaphore.appliedzkp.org/
6. Privacy and Scaling Explorations, "Rate Limiting Nullifier (RLN) Specification v1.0," 2022. https://rate-limiting-nullifier.github.io/rln-docs/
7. Blagoj, "Rate Limiting Nullifier: A spam-protection mechanism for anonymous environments", https://medium.com/privacy-scaling-explorations/rate-limiting-nullifier-a-spam-protection-mechanism-for-anonymous-environments-bbe4006a57d
8. Rarimo Team, "Rarimo Whitepaper v0.2," pre-print, 2023. https://docs.rarimo.com/files/Rarimo_whitepaper_v0.2.pdf
9. Rarimo Team, "Rarimo: A Privacy-First (zk) Social Protocol," Whitepaper v3, 2024. https://docs.rarimo.com/files/Rarimo_whitepaper_v3.pdf
10. H. Rajat, "Push Protocol Litepaper v0.9," pre-print, 2024. https://push.org/litepaper.pdf
11. S. Kulechov *et al.*, "Lens Protocol Whitepaper v1.1," Aave Labs, 2023. https://lens.red.r0h.in/paper.pdf
12. O. Thorén, S. Taheri-Boshrooyeh, and H. Cornelius, "Waku: A Family of Modular P2P Protocols for Secure and Censorship-Resistant Communication," in *Proc. IEEE ICDCSW 2022*, arXiv:2207.00038.

Authors

Oleksandr Kurbatov is a PhD Candidate in Karazin Kharkiv National University, Ukraine. His research focuses on public-key infrastructure, blockchain technologies, and anonymous decentralized voting systems. Currently working as the Lead Cryptography Researcher at Distributed Lab

Yaroslav Panasenko ris the Chief Technology Officer at Distributed Lab. He holds a B.Sc. in Software Engineering from Kharkiv Polytechnic Institute. His research interests span decentralized anonymous banking systems, blockchain infrastructure, biometric systems and AI safety and security.

Volodymyr Dubinin holds an M. Sc. in Computer Science and is Co-Founder of Distributed Lab. His research interests span Decentralized Systems, Artificial Intelligence, Blockchain Scalability, and Cryptographic Protocol Design.

Yevhen Hrubiian is a lecturer at the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" and a cryptography researcher with Distributed Lab. He holds an M.Sc. in Applied Mathematics from NTUU "Igor Sikorsky KPI." His scientific interests encompass elliptic-curve cryptography, zero-knowledge proofs, folding schemes and privacy-preserving messaging systems.