# Decentralized Identity and the Prevention of Public Key Substitution Attacks

*A Technical Survey of Key Event Infrastructure, Hardware Security, and Distributed Trust Models*

Waitman Gobble

Hyphero

A California Benefit Corporation (B20260123923)

Dedicated to decentralization, user sovereignty and open-source communications

## Abstract

Public Key Infrastructure (PKI) authentication, despite its cryptographic rigor, is subject to a class of attacks that exploit not the mathematics of key generation but the integrity of the systems that bind identities to those keys. This paper examines the problem of public key substitution, wherein a malicious or compromised database administrator may silently replace a legitimate user's public key with one under adversarial control. It surveys the primary engineering defenses against this threat, including Hash-based Message Authentication Codes (HMACs), digital signing services, Hardware Security Modules (HSMs), and append-only audit logs. The analysis then proceeds to examine more fundamental architectural remedies, notably the Key Event Receipt Infrastructure (KERI) protocol, Soroban smart contracts on the Stellar blockchain, and deterministic key derivation from BIP-39 seed phrases. Hardware instantiation of these systems through PIV-compliant JavaCards is also addressed, culminating in a proposed layered identity model relevant to decentralized social networking applications.

## 1. The Public Key Substitution Problem

In any system that uses asymmetric cryptography for authentication, the server or relying party must possess an accurate record of each user's public key. The security of the system depends on the assumption that this record faithfully represents the key the user originally enrolled. While public keys are not confidential in the way private keys are, their integrity is fundamental. An attacker who can substitute a user's public key in the authoritative store with a key they themselves control will, in effect, be able to impersonate that user to any service that consults that store.

This threat does not require the attacker to break any cryptographic primitive. It requires only the ability to write to the database or directory where public keys are stored. A database administrator, a compromised application server, or an attacker who has gained write access through a separate vulnerability can execute this substitution silently and, in many architectures, without leaving a recoverable trace. This is the core weakness of centralized identity systems: the binding of an identity to a public key is enforced by an administrative trust relationship rather than by the mathematics of the cryptographic system itself.

## 1.1 Database-Level Integrity Checks Using HMACs

One direct mitigation is to store, alongside each public key, a Hash-based Message Authentication Code (HMAC) computed over the key material and a server-side secret. At registration, the application computes HMAC-SHA256 over the user's public key combined with a secret known only to the application layer. This HMAC is stored in the database alongside the public key. At each subsequent authentication attempt, the application recomputes the HMAC and compares it to the stored value. An attacker who replaces the public key but does not know the server secret will produce a mismatched HMAC, and the authentication will fail.

This mechanism defends specifically against a rogue database administrator who lacks access to the application's secret material. It does not, however, protect against an attacker who has also compromised the application layer and thereby obtained the HMAC secret, nor does it provide a transparent audit trail of when changes occurred.

| **Defense Method** | **Primary Threat Addressed** | **Complexity** |
|---|---|---|
| HMACs | Database tampering and rogue database administrator | Low |
| Digital Signatures | Key substitution and data corruption | Medium |
| HSMs | Physical and high-level logical access | High |
| Audit Logs | Insider threats and post-incident forensics | Medium |

## 1.2 Digital Signatures and the Root of Trust

A more robust approach replaces raw key storage with a signed credential. An internal signing service, typically running on dedicated hardware isolated from the main application stack, issues a digital signature over the user's public key at the time of enrollment. The application then refuses to accept any public key that is not accompanied by a valid signature from this internal authority. Even if an attacker replaces the key material in the database, they cannot produce a valid signature over the substitute key without access to the signing service's private key.

This approach introduces the concept of a root of trust: a distinguished component whose integrity anchors the trustworthiness of all downstream operations. The strength of the system is now proportional to the security of that signing service rather than the security of the database.

### 1.3 Hardware Security Modules

For environments requiring the highest assurance, Hardware Security Modules (HSMs) provide tamper-resistant physical storage for cryptographic keys and the secrets used to validate them. HSMs enforce strict access controls at the hardware level, such that even a user with root access to the operating system of the web server typically cannot extract or modify key material stored within the HSM. Many enterprise HSMs additionally enforce dual-control policies, requiring authorization from two separate administrators before sensitive operations can be performed, which significantly raises the cost of any insider attack.

### 1.4 Immutable Audit Logs

Write-once, read-many (WORM) logs and cryptographic transparency logs provide a complementary defense. While they do not prevent a substitution in real time, they ensure that any substitution is recorded permanently. If each key change is appended to a log that cannot be altered or deleted, a post-incident investigation can reconstruct the exact timeline of events and identify the source of any unauthorized change. The deterrence value of such logs is also significant: an attacker who knows their actions will be permanently visible and attributable is far less likely to attempt a substitution at all.

### 1.5 Multi-Factor Authentication for Key Changes

Application-level controls can further reduce the risk of unauthorized key substitution by requiring out-of-band verification for any key update operation. A requirement that the user confirm a key change via a hardware token such as a YubiKey, a time-based one-time password (TOTP), or an email confirmation to a verified address ensures that access to the database alone is insufficient to effect a substitution. Even an attacker with direct database write access cannot complete a key change without the user's secondary factor.

## 2. Blockchain-Based Decentralized PKI

The defenses described in Section 1 are each valuable but each also relies, to some degree, on trusting a centralized component. The HMAC mechanism trusts the application's secret store; the signing service trusts its own isolation; the HSM trusts its physical security; and the audit log, in many implementations, trusts the integrity of the log storage itself. A more fundamental remedy is to relocate the source of truth from a privately administered system to a public, decentralized ledger on which the rules of key modification are enforced by consensus rather than by administrative policy.

### 2.1 Soroban Smart Contracts on the Stellar Network

The Stellar blockchain, with its Soroban smart contract platform, provides a practical environment for implementing a Decentralized PKI (dPKI). In a traditional web application, an administrator can execute a direct database update to replace a user's public key. In a Soroban contract, the logic governing key updates is compiled into the contract bytecode and enforced by every node in the Stellar network. The contract can be written to require that any key update transaction be signed by the holder of the current private key, making unauthorized substitution cryptographically impossible without compromising the user's private key directly.

Because every transaction on the Stellar ledger is permanently recorded and globally visible, key changes cannot be performed covertly. Any attempt to modify a user's key, whether by the application developer or by any other party, produces a permanent, publicly inspectable transaction record. Furthermore, because the Stellar network is maintained by independent validator nodes operating under a consensus protocol, no single party can alter the state of the ledger unilaterally.

## 2.2 Architectural Comparison

| **Component** | **Traditional PKI** | **Soroban-Based Decentralized PKI** |
|---|---|---|
| Storage | Central SQL database | Global Stellar ledger |
| Key update mechanism | Admin dashboard or direct SQL | Smart contract update_key function |
| Verification method | Database record lookup | Transaction signature verification |
| Primary weak point | Compromised database or rogue administrator | Private key loss (mitigated by social recovery) |

## 2.3 The Key Transparency Contract Pattern

A practical implementation for a decentralized application would involve a Key Transparency Contract deployed on Soroban. At account creation, the user registers their public key in the contract. The contract records the ledger close time of that registration. Any subsequent messages from that user are trusted only if they are signed by the key anchored in the contract. The following Rust pseudocode illustrates the core update logic:

```
pub fn update_key(env: Env, old_key: BytesN<32>, new_key: BytesN<32>) {
    let current_key: BytesN<32> = env.storage().instance()
        .get(&DataKey::PublicKey).unwrap();
    assert_eq!(old_key, current_key, "Old key does not match current record");
    env.current_contract_address().require_auth();
    env.events().publish((symbol_short!("key_upd"), old_key.clone()),
new_key.clone());
    env.storage().instance().set(&DataKey::PublicKey, &new_key);
}
```

This contract enforces three invariants simultaneously: the new key can only be set if the caller can prove knowledge of the current private key (via require_auth), the old key must match the on-chain record, and every successful update emits a permanent, indexed event. If an attacker modifies the public key in the application's off-chain cache database but does not produce a corresponding on-chain transaction, any peer that verifies against the Soroban ledger will detect the discrepancy and reject the authentication.

## 3. KERI: Key Event Receipt Infrastructure

While blockchain-based solutions provide strong guarantees, they impose transaction fees and introduce a dependency on the continued operation of a specific network. Key Event Receipt Infrastructure (KERI) is a protocol designed to provide equivalent assurance of key lineage without requiring any shared ledger. KERI was formalized as a version 1.0 standard by the Linux Foundation's Trust Over IP (ToIP) Foundation on 21 January 2026, and has been adopted as the cryptographic backbone of the Global Legal Entity Identifier Foundation's verifiable Legal Entity Identifier (vLEI) system.

### 3.1 The Key Event Log

In KERI, each identity is associated with a Key Event Log (KEL), an append-only sequence of signed records that captures the complete history of key changes for that identity. The KEL begins with an Inception Event and is extended by Rotation Events whenever the user changes their active key. Each event in the log contains the sequence number, a cryptographic reference to the previous event (forming a hash chain), the current public key, and a commitment to the next key in the form of a cryptographic digest. Because each event is signed by the key that was current at the time it was created, the entire log can be verified by any party without consulting a central authority.

The self-contained nature of the KEL is its most important property: the authority to verify an identity derives from the mathematical consistency of the log itself, not from the trustworthiness of any server or administrator. This property is referred to in the KERI literature as ambient verifiability.

### 3.2 Pre-Rotation and Its Security Properties

KERI's primary defense against key compromise is a mechanism called pre-rotation. When a user creates their inception event, they generate two keypairs: a current signing key and a next rotation key. The inception event publishes the current public key in full but publishes only a cryptographic digest of the next key. The actual next public key is not disclosed and need not be stored anywhere other than the user's own secure backup.

If an attacker later steals the current signing key, they cannot produce a valid rotation event because they do not possess the next private key and cannot determine it from its published digest. A rotation event is only accepted by verifiers if it reveals the full public key corresponding to the previously committed digest and is signed by the associated private key. This means that the security of the account is not determined solely by the strength of the current key but by a chain of committed future keys that were never exposed to the environment in which the current key operated.

The following JSON structure illustrates a simplified rotation event in KERI's encoding:

```
{
  "v": "KERI10JSON000122_",
  "t": "rot",
  "i": "DHoqdD_OjU...uyE",
  "s": "1",
  "p": "EHS-oD_sC0...HB1",
  "k": ["DQpEvB6ylw...7k"],
  "n": ["EaQSjEqCJX...ZoU"]
}
```

## 3.3 Social Recovery in KERI

KERI supports multi-signature rotation events, which enables a social recovery mechanism for users who have lost their active key. At inception, the user may commit to a next key that requires signatures from a threshold number of designated guardians before a rotation is accepted. For example, a user might commit to a next key structure that requires three of five named public keys to co-sign any rotation event. If the user loses their primary key material, they can generate a new key, present it to their guardians, collect the required signatures, and publish a valid rotation event without any administrator's involvement.

This mechanism entirely eliminates the administrative recovery pathway that is the conventional weakness of centralized systems. There is no help desk to socially engineer, no administrator who can be bribed or coerced, and no database reset procedure that can be exploited. The recovery authority rests with the user's social relationships, expressed in terms of cryptographic signatures.

## 3.4 Deterministic Key Derivation from Seed Phrases

A practical challenge in implementing KERI is the management of the next private key, which must be preserved securely but is not stored on any connected device. A standard solution is to derive both the current and next keypairs deterministically from a BIP-39 mnemonic seed phrase using a hierarchical key derivation function. The seed phrase, typically 12 to 24 words, is the root of a deterministic sequence: the key at index zero is used as the current signing key, the key at index one is the next rotation key, the key at index two is the rotation key after that, and so on.

After the inception event has been constructed, the next private key (index one) need not be retained on any device. If the current key is later compromised or lost, the user can reconstruct the next private key at any time by applying the same derivation function to their seed phrase. Because the derivation is deterministic, the result will always be the same key that was committed to in the inception event, enabling a valid rotation without any administrative intervention.

The primary security burden this model places on the user is the physical security of the seed phrase itself. The seed phrase should be stored on a durable medium, such as a metal card, in a secure physical location, and should never be stored electronically in plaintext. The next private key can be discarded from digital memory precisely because it is reproduced on demand from the seed, which reduces the attack surface of the user's connected devices to only the current signing key.

## 4. Hardware Instantiation: PIV-Compliant JavaCards

The practical deployment of a KERI-based identity system for everyday users requires hardware that can store cryptographic commitments securely and perform signing operations in a tamper-resistant environment. PIV (Personal Identity Verification) compliant smart cards, including the NXP JCOP 4 series running JavaCard 3.0.5, provide a suitable platform for this purpose.

### 4.1 Card Selection and Provisioning

For a community-oriented deployment in which auditability is a primary concern, the NXP J3R180 card is an appropriate choice. This card provides 180 kilobytes of EEPROM, supports elliptic curve cryptography up to 521 bits including the secp256k1 curve required for KERI, and presents a dual-interface design supporting both contact-based USB readers and contactless NFC operation. Cards must be obtained in the unfused configuration: fused cards have their transport keys locked by the manufacturer or distributor and cannot be re-programmed with custom applets.

Once obtained, cards should be provisioned with an open-source PIV applet such as PivApplet or the SeedKeeper applet, loaded using GlobalPlatformPro (gp). Verification of the card's initial state before provisioning is essential. The gp --info command reports the Card Production Life Cycle (CPLC) data, which identifies the chip version and fuse state. Any card that shows existing keys or a non-zero signature counter should be treated as potentially compromised and wiped using the PIV applet reset command before use.

### 4.2 Storing KERI Commitments in PIV Slots

The PIV standard defines a set of data objects accessible via tagged slot identifiers. For KERI use, the relevant slots are the authentication slot (0x9A), the data management slot (0x9D), and the printed information slot (0x5FC109). The current public key is best stored in the authentication slot, where it is accessible to standard PIV tooling. The next key digest, which is the cryptographic commitment to the future rotation key, is stored in the printed information slot, which requires PIN entry for both read and write access.

The following Rust pseudocode illustrates writing a next-key commitment to the card using the yubikey crate:

```
fn anchor_keri_commitment(
    yk: &mut YubiKey,
    next_key_hash: [u8; 32]
) -> Result<(), Box<dyn std::error::Error>> {
    let default_mgmt = MgmtKey::default();
    yk.authenticate(default_mgmt)?;
    yk.verify_pin(pin.as_bytes())?;
    let tag: u32 = 0x5FC109;
    yk.put_data(tag.try_into()?, &next_key_hash)?;
    Ok(())
}
```

By storing the digest rather than the next private key, the card acts as a hardware attestation of the commitment made in the inception event. Even if the card is stolen, the attacker learns only the hash of the next key, from which they cannot derive the next private key. The actual rotation capability remains with whoever holds the seed phrase.

### 4.3 The secp256k1 Succession Protocol

Because the JCOP 4 SecID profile natively accelerates secp256k1 operations through its hardware co-processor, KERI implementations on this platform should use secp256k1 rather than Ed25519, which may require software-emulated curve arithmetic on some card models and is consequently slower and less reliable. The k256 Rust crate provides the necessary primitives. The succession proof is generated as follows:

```
fn generate_keri_succession(seed: &[u8], current_index: u32) {
    let key_a = SigningKey::from_bytes(
        &derive_key(seed, current_index)).unwrap();
    let key_b = SigningKey::from_bytes(
        &derive_key(seed, current_index + 1)).unwrap();
    let next_pub_key = key_b.verifying_key()
        .to_encoded_point(false);
    let mut hasher = Sha256::new();
    hasher.update(next_pub_key.as_bytes());
    let commitment_digest = hasher.finalize();
}
```

The card verifies, at rotation time, that the revealed public key for Key B matches the stored commitment digest, and that the rotation event is signed by Key A. Both conditions must hold for the rotation to be accepted.

## 5. Cost-Free Alternatives and Complementary Technologies

While the Soroban-based approach provides the strongest possible guarantees of immutability and transparency, it requires users to hold a small amount of XLM to cover transaction fees, which creates friction in mainstream adoption. Several technologies provide comparable security properties for key lineage verification without incurring transaction costs.

## 5.1 KERI with Distributed Hash Tables

KERI logs are self-contained signed files that can be distributed through any reliable transport. A Kademlia-based Distributed Hash Table (DHT), such as the one used by the libp2p networking library, can serve as a decentralized storage and discovery layer for Key Event Logs. A user publishes their KEL to the DHT under a key derived from their KERI Autonomic Identifier (AID). Peers wishing to verify the user's identity query the DHT, retrieve the KEL, and verify the chain of signatures from the inception event forward. No central server is required, no transaction fees are incurred, and the adversarial requirement for a successful key substitution attack is the ability to subvert the DHT consensus, which requires controlling a substantial fraction of the participating nodes.

## 5.2 The PLC Transparency Log Model

Bluesky's AT Protocol employs a PLC (Placeholder) directory as a specialized, high-transparency append-only log for identity operations. Users hold an offline rotation key, derived from a seed phrase, which is the only key authorized to sign changes to the user's identity record in the directory. Even if the directory server is fully compromised, the attacker cannot modify any user's key without a valid signature from that user's rotation key. The directory is operated as a public service and publishes a continuous stream of all operations, allowing any party to detect unauthorized modifications.

This model is well-suited to federated social networks because it places the security burden on key management rather than on server integrity, while remaining compatible with existing web infrastructure and requiring no per-transaction fees from users.

## 5.3 Content-Addressed Storage via IPFS

IPFS (InterPlanetary File System) provides a content-addressed storage layer in which every object is identified by the cryptographic hash of its content. A user can store their KERI Inception Event on IPFS and distribute its Content Identifier (CID) as their identity anchor. Updates to the identity create new objects that reference the CID of the previous state and are signed by the key established in that previous state. Because the CID of any historical object is immutable, an attacker who modifies the identity data cannot produce the same CID. The divergence is immediately detectable by any verifier.

| **Method** | **Financial Cost** | **Rogue Admin Protection** | **Complexity** |
|---|---|---|---|
| Soroban / Stellar | Very low (requires XLM) | Absolute (ledger consensus) | High |
| KERI with DHT | None | High (verifiable lineage) | Very high |
| PLC transparency log | None | High (append-only log) | Medium |
| IPFS content addressing | None | Medium (content hashing) | Low |

## 6. Integration with Federated Social Networks

The problem of key substitution is not merely a theoretical concern for decentralized identity systems; it has direct practical relevance for federated social networking protocols such as ActivityPub. In ActivityPub, each actor is represented by a JSON-LD document containing, among other fields, a public key used to verify HTTP signatures on messages the actor sends. This document is served from the actor's home server. If that server is compromised, or if the server administrator is malicious, the public key in the actor document can be replaced, enabling the attacker to send messages that appear to be from the actor.

### 6.1 KERI Anchoring for ActivityPub Actors

A Key Transparency Contract on Soroban or a KERI KEL published to a DHT can serve as an authoritative external reference for an ActivityPub actor's public key. The actor document would include, in addition to the standard publicKey field, a reference to the actor's KERI AID or Soroban contract address. Receiving servers that wish to verify the authenticity of a message can resolve the AID or contract address to obtain the authoritative current key and compare it against the key in the actor document. A discrepancy between the server-hosted key and the ledger-anchored key is a definitive signal of substitution.

### 6.2 Nomadic Identity and the Hubzilla Model

Hubzilla's Zot protocol demonstrates a complementary approach to the same problem. In Hubzilla, a user's channel is a portable cryptographic object that can exist simultaneously on multiple servers as a set of synchronized clones. If one server's administrator substitutes the public key in that server's copy of the channel, the discrepancy is immediately apparent from the existence of the other clones, which continue to serve the legitimate key. Incoming contacts who are aware of all clone locations can detect the substitution by comparing the keys served by each clone.

This nomadic identity model reduces the rogue administrator problem to a clone-detection problem. It does not entirely eliminate the risk, since an attacker who can simultaneously compromise all clone servers could effect a consistent substitution, but it significantly raises the cost of a successful attack. The KERI KEL model can be combined with the nomadic identity model: the KEL serves as the ultimate authority on key lineage, while the clone network provides high availability for key resolution.

## 7. The Key Recovery Lifecycle

A complete identity system must address not only the prevention of unauthorized key substitution but also the authorized processes by which users legitimately change their keys over time. Three scenarios are of particular practical importance: proactive key rotation when replacing hardware, reactive recovery after hardware loss, and migration from one key type to another.

## 7.1 Proactive Rotation

When a user replaces their primary signing hardware, such as a YubiKey, the rotation follows the standard KERI succession protocol. The inception event committed to a digest of the next key. At rotation time, the user reveals the full next public key (proving it corresponds to the committed digest) and signs the rotation event with the old key. The result is a valid KERI rotation event that all verifiers accept, and the old key is permanently revoked from the log.

## 7.2 Reactive Recovery After Key Loss

If the primary signing hardware is lost or destroyed without opportunity to perform a proactive rotation, recovery depends on the pre-rotation commitment. In the seed phrase model, the user enters their seed phrase on a new device, re-derives the next keypair at the appropriate index, and constructs a rotation event proving that this key was the committed successor. In the social recovery model, the user's designated guardians co-sign a recovery event that endorses the new key as the legitimate successor. In both cases, the recovery is authorized by the mathematical structure of the KEL itself, not by any administrator's discretion.

## 7.3 Migration Between Key Types

Migration from a hardware-bound key, such as a PIV card, to a platform authenticator such as a device-based passkey is handled identically to a proactive rotation. The platform authenticator generates a new keypair in the device's secure enclave, and the user signs a rotation event with the old hardware key authorizing the new platform key as the current signing key. Subsequent interactions use the platform key for daily signing. For high-security operations, a tiered authority model is possible: the platform key may be sufficient for message signing, while operations such as key rotation or account recovery require the hardware key or a threshold of guardian signatures.

| Scenario | Practical Mechanism | Proof Required |
|---|---|---|
| Proactive hardware replacement | Standard KERI rotation event | Signature from the current hardware key |
| Lost key (seed phrase recovery) | Deterministic re-derivation of next key | Derived key matches committed digest |
| Lost key (social recovery) | Multi-signature guardian event | Threshold of guardian signatures |
| Platform key migration | Cross-type rotation event | Signature from outgoing key authorizing incoming key |

# 8. The Layered Identity Model

Drawing together the mechanisms described in the preceding sections, a coherent layered identity model a decentralized platform can be specified. The model is designed to achieve three properties simultaneously: zero dependence on administrative trust for key integrity, a practical user experience that does not require users to manage raw cryptographic material, and a graduated security posture that reserves the most burdensome procedures for infrequent high-stakes operations.

## 8.1 Architecture Overview

| Tier | Component | Purpose |
|---|---|---|
| Hot | Secure enclave or passkey | Daily signing for ActivityPub messages and matchmaking operations |
| Cold | PIN-protected PIV card (J3R180) | Hardware storage of seed phrase index and next key commitment |
| Social | Guardian threshold signatures | Emergency recovery when both hot and cold credentials are lost |

## 8.2 Enrollment Protocol

At account creation, the application generates a BIP-39 seed phrase and derives two keypairs: the current signing key at index zero and the next rotation key at index one. The current key is enrolled in the user's secure enclave or PIV card. The next key's digest is written to the PIN-protected printed information slot of the PIV card and committed to in the KERI inception event. The seed phrase is displayed to the user once, printed or otherwise externalized to offline storage, and then purged from application memory. The next private key is also purged, as it can be re-derived from the seed phrase at any future time.

The inception event is published both to the application database, which serves as a convenient cache for fast lookups, and to the user's chosen external anchoring system, which may be a Soroban contract, a DHT-hosted KEL, or a PLC directory. The database is explicitly treated as a cache, not as an authority: any peer that requires high assurance resolves the user's identity against the external anchor.

## 8.3 Verification Protocol

When a peer receives a signed message from a user, the verification procedure proceeds in two steps. First, the message signature is verified against the public key included in the message or the sender's cached actor document. Second, if the operation is high-stakes, such as a key change notification or an administrative action, the verifying peer resolves the sender's identity anchor and confirms that the key used for signing matches the key currently recorded in the authoritative log. If the two do not match, the message is rejected and the discrepancy is logged.

This two-tier verification allows routine high-volume operations, such as message signature verification in a social feed, to proceed efficiently using the local cache, while ensuring that operations with lasting identity consequences are always verified against the authoritative source.

## 9. Standardization Status and Ecosystem

KERI, together with its companion specifications CESR (Composable Event Streaming Representation) and ACDC (Authentic Chained Data Containers), was formally approved as a version 1.0 standard by the Trust Over IP Foundation Steering Committee on 21 January 2026. The KERI Suite Working Group, co-chaired by Samuel Smith and Philip Feairheller, maintains the specifications under the Linux Foundation umbrella. The CESR specification governs the encoding of KERI events, supporting both human-readable text mode and efficient binary mode. ACDC governs the issuance and verification of credentials anchored to KERI identifiers.

The most significant production deployment of KERI as of early 2026 is GLEIF's vLEI (verifiable Legal Entity Identifier) system, which uses KERI as the cryptographic infrastructure for identifying legal entities in cross-border financial contexts. GLEIF chose KERI specifically because of its ledger-agnostic design: a vLEI-anchored identity can be verified by institutions operating on different blockchain networks or no blockchain at all, with no dependency on any single intermediary.

For Rust implementations, the keriox and kerir libraries provide the relevant primitives. The keriox crate is currently being used in Linux Foundation Decentralized Trust Labs for high-performance edge deployments. The did:keri method is registered within the W3C Decentralized Identifier (DID) ecosystem, enabling KERI-based identifiers to interoperate with tooling built for W3C DIDs.

## 10. Conclusion

The binding of a user's identity to a public key is the most structurally fragile element of any PKI-based authentication system. Administrative trust, whether placed in a database administrator, an application server, or a certificate authority, represents a surface through which the mathematical guarantees of asymmetric cryptography can be circumvented without breaking any cryptographic primitive. The defenses available range from incremental mitigations such as HMACs and audit logs to fundamental architectural alternatives that remove the administrative binding entirely.

KERI, as a now-standardized protocol, offers the most principled solution to this problem by grounding identity in a self-certifying, append-only event log whose integrity depends on the consistency of signatures rather than the trustworthiness of any server. Combined with deterministic key derivation from BIP-39 seed phrases, PIV-compliant hardware for commitment storage, and social recovery mechanisms for the loss scenarios that hardware alone cannot address, KERI enables an identity system in which the answer to the question of whether a public key is legitimate is not 'the administrator says so' but rather 'the mathematics of the log proves it.'

For decentralized social networking applications operating on ActivityPub or similar protocols, the practical integration of these mechanisms requires bridging between the key transparency layer and the actor document model, and between the federated server infrastructure and the authoritative identity anchor. This is an area of active development, and the maturation of KERI tooling in Rust makes it an increasingly practical choice for new projects that take the integrity of user identity seriously.

## References

Trust Over IP Foundation. KERI Specification v1.0. Approved 21 January 2026. KERI Suite Working Group.

Trust Over IP Foundation. CESR Specification v1.0 (Composable Event Streaming Representation). January 2026.

Trust Over IP Foundation. ACDC Specification v1.0 (Authentic Chained Data Containers). January 2026.

Smith, Samuel M. Key Event Receipt Infrastructure (KERI). Internet-Draft draft-ssmith-keri. IETF.

Global Legal Entity Identifier Foundation (GLEIF). Verifiable LEI (vLEI) Ecosystem Governance Framework. 2024.

W3C. Decentralized Identifiers (DIDs) v1.0. W3C Recommendation, July 2022.

Hardman, Daniel, et al. did:keri DID Method Specification. Decentralized Identity Foundation.

Bitcoin Improvement Proposal 39 (BIP-39). Mnemonic Code for Generating Deterministic Keys.

NXP Semiconductors. JCOP 4 P71 Security Controller Product Data Sheet.

Yubico. PIV (Personal Identity Verification) for YubiKey. Technical Reference.

Frazee, Paul, and Jay Graber. The AT Protocol: A Federated Protocol for Large-Scale Distributed Social Applications. Bluesky, 2023.

Hess, Florian. GunDB Technical Reference and Distributed Application Framework. 2022.

libp2p Project. Kademlia DHT Specification. Protocol Labs.