

KeyPears: Simple Federated Encrypted Messaging System

Ryan X. Charles

ryan@ryanxcharles.com

April 2026

Abstract. KeyPears is a simple federated encrypted messaging system for communication and authentication. It keeps email's useful properties: human-readable `name@domain` addresses, DNS-based federation, and the ability for any domain to run its own server. It adds encrypted message bodies, automatic public-key lookup, proof of work for abuse resistance, and redirect-based third-party authentication. Private-key operations, encryption, signing, and proof-of-work mining happen client-side. Servers store encrypted message bodies and secret payloads, but do not possess the keys needed to decrypt already-stored ciphertext. Domains remain trusted authorities for their users' current public keys and hosted client code. This email-like trust boundary keeps the protocol small enough for broad implementation: if a user does not trust a hosted server in that role, the trust exit is to switch providers or host their own domain. The cryptography combines Ed25519 and X25519 with ML-DSA-65 (FIPS 204) and ML-KEM-768 (FIPS 203) for hybrid post-quantum defense-in-depth.

1. Introduction

Internet communication has relied on email for over four decades. Email got two things right: human-readable addresses (`name@domain`) and federation via DNS (any domain can run a mail server, and servers discover each other through MX records). These properties gave email universal reach without central control.

But email was designed for trusted networks. Two fundamental deficiencies, deeply embedded in the SMTP protocol, have resisted every attempt at correction.

No key exchange. Without cryptographic keys bound to addresses, end-to-end encryption requires users to manage keys manually. PGP attempted this: public-key cryptography layered onto email via key servers and a web of trust. Whitten and Tygar demonstrated the result in 1999 [1]: given 90 minutes with PGP 5.0, the majority of test participants could not successfully encrypt a message. The problem was not the cryptography but the user interface. Twenty-seven years later, encrypted email remains a niche practice.

No cost to send. Delivering an email costs the sender nothing, making spam economically rational. Back proposed Hashcash in 1997 [2]—a proof-of-work scheme that imposes a computational cost on each message. The idea was sound, but SMTP has no mechanism to negotiate proof of work, and backwards compatibility prevents making it mandatory.

Centralized alternatives solved these problems by abandoning federation. Signal [3] provides automatic key management and end-to-end encryption, but identity is bound to phone numbers and a single organization runs every server.

We propose a protocol that keeps what email got right—federated `name@domain` addressing, DNS-based server discovery, and simple deployment—while adding hybrid post-quantum key exchange for encrypted message bodies, composite signatures for authentication, and proof of work for spam mitigation. The design intentionally keeps domain servers as the authorities for their users' current public keys. This does not remove all trust from hosted servers; it makes the trust boundary portable. A user who does not trust a hosted server can switch service providers, or host the domain themselves.

Because elliptic-curve systems face long-term quantum risk, KeyPears uses hybrid classical + post-quantum cryptography. All asymmetric operations combine Curve25519 primitives (X25519, Ed25519) with NIST-standardized post-quantum algorithms (ML-KEM-768 [4], ML-DSA-65 [5]), matching the hybrid direction adopted by Signal [6], Chrome TLS, and the IETF OpenPGP PQC draft [7].

2. Design Principles

KeyPears is guided by six principles:

1. **Simple.** The protocol is small enough for many kinds of applications to embed. Simplicity is an adoption and security property: a protocol that many apps implement correctly is preferable to a more ambitious protocol that few implement at all.
2. **Federated.** Any domain can run a KeyPears server. Servers discover each other via DNS. No registration authority controls participation.
3. **End-to-end encrypted.** Message bodies and secret payloads are encrypted client-side; servers store only ciphertext for these. Metadata (addresses, vault labels) is intentionally plaintext for routing and search.
4. **Client-side proof of work.** Every account creation, login, and message requires proof of work computed by the client.
5. **DNS-based identity.** Addresses are `name@domain`. Identity is bound to DNS domain ownership, not to a phone number or a central registry. The server for a domain is trusted to publish honest current public keys for that domain's users; self-hosting is the trust exit.
6. **Hybrid post-quantum by default.** Classical (Ed25519, X25519) and post-quantum (ML-DSA-65, ML-KEM-768) algorithms are combined in every cryptographic operation. Both must be broken to compromise the system.

3. Identity and Addressing

A KeyPears address has the form `name@domain`—intentionally identical to an email address. The domain is a standard DNS domain. An organization with existing email addresses can use the same addresses for KeyPears.

Each user has signing keys and encryption keys. Signing keys prove who sent a message or approved an authentication request. Encryption keys protect message bodies and stored secrets. Each role uses one classical key pair and one post-quantum key pair:

- **Ed25519 key pair**: a 32-byte public key and 32-byte private key. Used as the classical signing key.
- **ML-DSA-65 signing key pair** (FIPS 204): a 1,952-byte verifying key and a 4,032-byte signing key. Used as the post-quantum signing key.
- **X25519 key pair**: a 32-byte public key and 32-byte private key. Used for classical Diffie-Hellman key exchange.
- **ML-KEM-768 encapsulation key pair** (FIPS 203): a 1,184-byte encapsulation key and a 2,400-byte decapsulation key. Used to encrypt a fresh shared secret to the recipient.

Users may rotate all four key pairs atomically, up to 100 sets per account. Old keys are retained so that messages encrypted under previous keys can still be decrypted. All private keys are encrypted client-side with AES-256-GCM under the user’s encryption key (Section 5) and stored on the server as ciphertext.

The currently hosting server publishes the active public key set for each address on its domain. This is an authoritative server response rather than a global transparency-backed identity proof.

4. Overview

A typical interaction proceeds as follows. Alice (alice@a.com) wants to send an encrypted message to Bob (bob@b.com). Neither Alice nor Bob has communicated before.

The flow uses three basic operations. Signing proves Alice sent the message. Encryption protects the message body. Proof of work pays the delivery cost Bob requires for incoming mail.

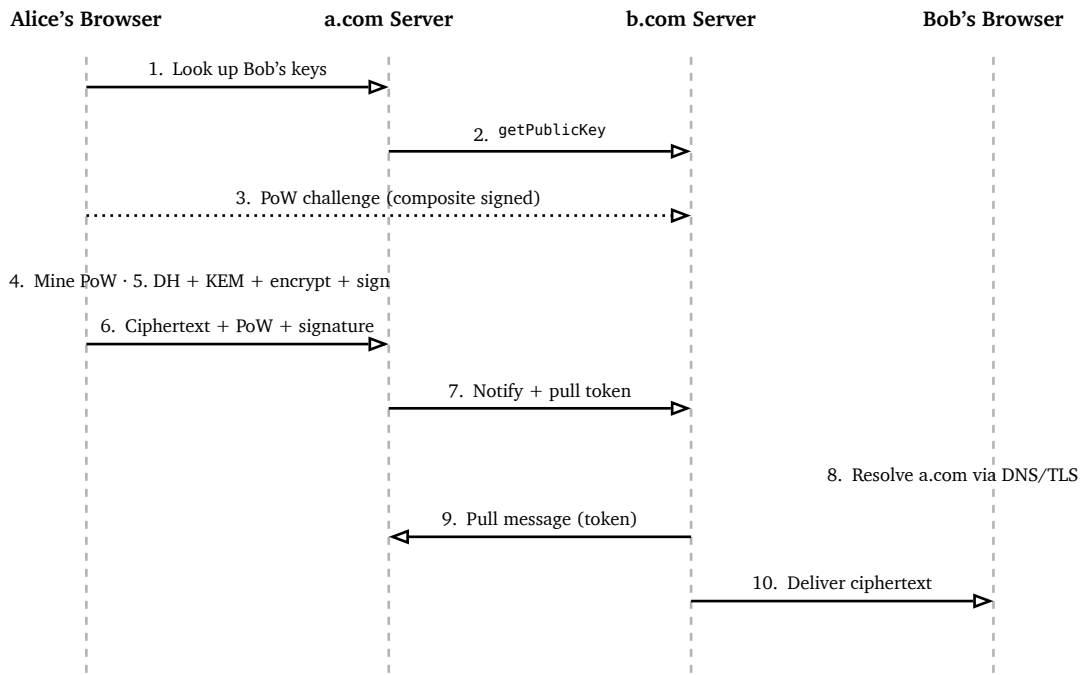


Figure 1: Message flow: Alice sends an encrypted message to Bob across two federated domains.

1. Alice's client asks her server for Bob's public keys. Her server fetches all four of Bob's public keys (Ed25519, X25519, ML-DSA-65, ML-KEM-768) from Bob's server via the federation API. Bob's server is the authority for the current keys of addresses hosted on Bob's domain.
2. Alice's client requests a proof-of-work challenge from Bob's server. The request is signed with Alice's composite Ed25519 + ML-DSA-65 key.
3. Alice's client mines the challenge on the GPU.
4. Alice computes an X25519 DH shared secret and encapsulates to Bob's ML-KEM-768 key. Both shared secrets are combined via HKDF-SHA-256 to derive an AES-256 key. She encrypts the message and a second copy to her own keys for sent-message history. She signs a canonical envelope with composite Ed25519 + ML-DSA-65.
5. Alice sends the ciphertexts, composite signature, and PoW solution to her server.
6. Alice's server stores her copy, creates a pull token, and notifies Bob's server.
7. Bob's server independently resolves Alice's domain via DNS and TLS. It pulls the ciphertext, verifies the composite signature, and stores the message.
8. Bob's client retrieves the ciphertext, re-derives the X25519 DH shared secret and decapsulates ML-KEM-768, and decrypts.

Under honest key discovery, no server possesses the plaintext or the keys needed to derive it. A domain server is still trusted to publish honest current public keys for its users; this trust boundary is discussed in the security analysis.

5. Key Derivation

Password-based key derivation uses a three-tier PBKDF2-HMAC-SHA-256 scheme (RFC 8018). The server-side tier alone performs 600,000 rounds, matching the OWASP Password Storage Cheat Sheet recommendation. The client also performs two deterministic 300,000-round tiers before sending the login key to the server. For a single target user, a password guess must pass through those client-side tiers and that user's server-side tier. Across many users, the deterministic client-side work can be reused per password candidate, while the 600,000-round server tier remains per-user because it uses a per-user salt.

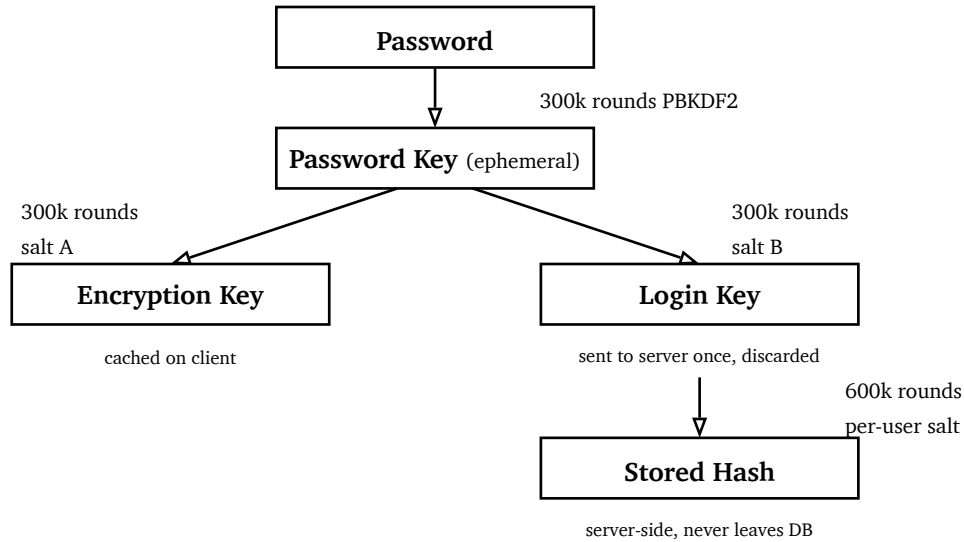


Figure 2: Three-tier key derivation. The password key is ephemeral. The encryption key and login key are siblings—compromising one does not reveal the other.

Tier 1: Password → Password Key. The password is stretched with 300,000 rounds of PBKDF2-HMAC-SHA-256. The result is an ephemeral 256-bit password key, used to derive the encryption key and login key, then discarded.

Tier 2a: Password Key → Encryption Key. A second 300,000-round PBKDF2 derivation with a distinct salt produces the encryption key. This key is cached on the client and used to encrypt and decrypt all four private keys (Ed25519, X25519, ML-DSA-65, ML-KEM-768). It is never sent to the server.

Tier 2b: Password Key → Login Key. A parallel derivation produces the login key. This key is sent to the server exactly once, then discarded. The server hashes it with an additional 600,000 rounds using a per-user salt before storage. This 600,000-round per-user server tier is the conservative password-storage baseline; the client-side tiers provide additional deterministic stretching.

Vault key. A separate key for encrypting stored secrets is derived as $K_{\text{vault}} = \text{HMAC-SHA-256}(K_{\text{encryption}}, \text{"vault-key-v2"})$. Each vault entry is independently encrypted with AES-256-GCM under K_{vault} .

6. Encryption

KeyPears uses AES-256-GCM (NIST SP 800-38D) for all symmetric encryption. Two encryption modes are used.

Hybrid message encryption. When Alice sends a message to Bob, she computes an X25519 Diffie-Hellman shared secret using her private key and Bob’s public key, and encapsulates a fresh shared secret to Bob’s ML-KEM-768 encapsulation key. The AES-256 key is derived from both shared secrets via HKDF-SHA-256 (RFC 5869):

$$K_{\text{AES}} = \text{HKDF-SHA-256}(\text{salt} = 0^{32}, \text{IKM} = S_{\text{X25519}} \parallel S_{\text{ML-KEM}}, \text{info})$$

where `info = "webbuf:aesgcm-x25519dh-mlkem v1"`. An attacker must break both X25519 ECDH and ML-KEM-768 to recover the message key. The message payload is encrypted with AES-256-GCM using K_{AES} ,

with the sender and recipient addresses bound as Additional Authenticated Data (AAD). The sender encrypts a second copy to their own keys for sent-message history. The KEM ciphertext and AES ciphertext are stored together as a combined blob per message.

Composite message signing. The sender signs a canonical length-prefixed envelope with composite Ed25519 + ML-DSA-65 (3,374 bytes: 1 version byte + 64-byte Ed25519 signature + 3,309-byte ML-DSA-65 signature). Both halves must verify independently. The envelope covers sender address, recipient address, all public keys, and both ciphertexts.

Vault encryption. The vault stores secrets encrypted under the vault key derived from the encryption key (Section 5). The server stores ciphertext alongside user-provided plaintext labels to enable server-side search.

7. Federation

Any domain can participate in the KeyPears federation by serving a configuration file at `/.well-known/keypears.json`:

```
{ "apiDomain": "keypears.acme.com" }
```

This file declares the domain’s API endpoint. An optional `admin` field names a KeyPears user authorized to manage accounts for the domain. Three deployment patterns are supported:

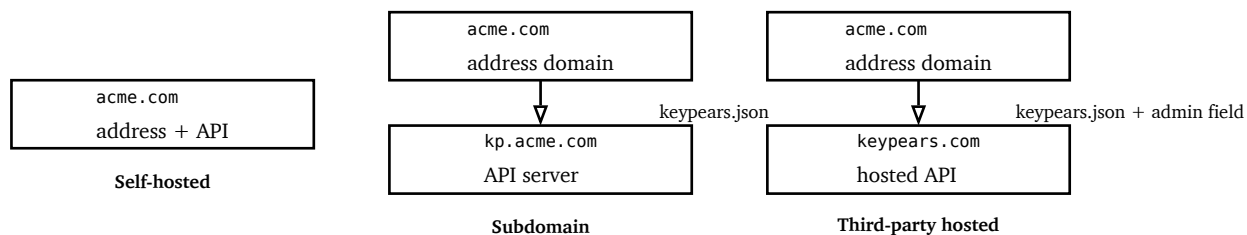


Figure 3: Three federation patterns. In each case, the address domain may differ from the API server.

Pull-model message delivery. Cross-domain messages use a pull model rather than server-to-server push. The sender’s server stores the ciphertext and issues a pull token. It notifies the recipient’s server, which independently resolves the sender’s domain via DNS and TLS. The recipient pulls the ciphertext, verifies the composite signature, and stores the message.

The pull model verifies which domain is speaking; it does not remove trust from that domain’s server. Each server remains the authority for the current public keys of the users it hosts. This is the same simplicity tradeoff that made email deployable, but with encrypted stored content and a self-hosting exit for users who want to control their own key authority.

8. Third-Party Authentication

KeyPears provides a redirect-based authentication protocol that allows any third-party application to verify a user’s identity without API keys, client registration, or pre-existing relationships.

1. The user types their domain on the third-party application.
2. The application fetches `keypears.json` to discover the API domain.

3. The application redirects the user's browser to the KeyPears server's /sign page with a structured signing request.
4. The user reviews a consent screen and approves. The KeyPears server generates a nonce, timestamp, and expiry, and the user signs a canonical JSON payload with their composite Ed25519 + ML-DSA-65 key.
5. The signed response is submitted back via HTTP POST (POST is required because composite signatures are 3,374 bytes, exceeding URL length limits).
6. The application verifies the composite signature by fetching the user's public keys via federation.

9. Proof of Work

Back proposed Hashcash [2] in 1997 as a proof-of-work scheme to make email spam expensive. KeyPears makes proof of work a first-class protocol requirement, building on Hashcash with four adaptations.

Interactive challenges. The recipient's server issues a challenge signed with HMAC-SHA-256, including a 15-minute expiry. Challenges are stateless until a valid solution is submitted.

GPU mining. All proof of work is computed client-side using the pow5-64b algorithm via WebGPU. Servers never mine.

Configurable difficulty. Server operators set difficulty for account creation and login. Individual users set difficulty for incoming messages.

Authenticated challenges. Challenge requests require the sender to sign with their composite Ed25519 + ML-DSA-65 key. The recipient's server verifies both the Ed25519 and ML-DSA-65 components via federation. Both sender and recipient addresses are bound into the challenge. This prevents social-graph probing.

10. Security Analysis

10.1. Server Compromise

The server stores encrypted message bodies and secret payloads alongside hashed credentials (login key hashed with 600,000 additional rounds of PBKDF2-HMAC-SHA-256 using a per-user salt, after deterministic client-side stretching). An attacker who captures the database cannot decrypt message content or secret payloads. Metadata (addresses, vault labels, channel counterparties) is stored in plaintext.

This protection applies to database compromise and passive server compromise. An active server remains trusted to publish honest current public keys for the addresses it hosts and to serve honest client code. If that server lies about future public keys, it can cause future messages to be encrypted to attacker keys. KeyPears does not attempt to prevent that class of hosted-server man-in-the-middle attack. Instead, it reduces the blast radius compared with email: a compromised email server can read stored mail, while a compromised KeyPears server cannot decrypt already-stored application ciphertext unless it also obtains client-side keys, passwords, or active client/session access.

This is a deliberate design choice. Adding global key transparency, contact pinning, or a no-trust hosted-server layer would make the protocol harder to embed and implement. The protocol's trust exit is self-hosting the domain.

10.2. Quantum Resistance

KeyPears uses a hybrid construction combining classical (Ed25519, X25519) and post-quantum (ML-DSA-65, ML-KEM-768) algorithms in every cryptographic operation. An attacker must break both to compromise any operation. This provides defense-in-depth matching the approach adopted by Signal [6], Chrome TLS, and the IETF OpenPGP PQC draft [7]: if a structural flaw is discovered in lattice-based cryptography, the classical algorithms still protect; if a cryptographically relevant quantum computer arrives, the post-quantum algorithms still protect. Grover’s algorithm provides only a quadratic speedup against symmetric primitives, reducing their effective security to 128 bits—still well within safe margins.

Babbush et al. [8] demonstrated in April 2026 that breaking 256-bit elliptic-curve discrete logarithms requires only 1,200 logical qubits—executable in approximately 9 minutes on a superconducting architecture with fewer than half a million physical qubits. KeyPears therefore does not rely on elliptic curves alone.

10.3. Spam and Sybil Attacks

Every account creation, login, and message requires proof of work, and the difficulty is tunable. The cost of an attack scales linearly with the number of targets and with the difficulty level.

10.4. Domain Spoofing

The pull model prevents domain spoofing without additional signing infrastructure. The recipient’s server independently resolves the sender’s domain via DNS and TLS. A malicious server cannot forge another domain’s identity because TLS guarantees the response came from the real domain.

Domain spoofing protection should not be confused with key transparency. Once a domain has been resolved, the server for that domain is still trusted as the current-key authority for hosted addresses.

10.5. Client Storage Theft

A localStorage-only theft exposes the cached encryption key, but it does not derive the login key, recover the user’s password, or create a server session. If the attacker also obtains encrypted private-key blobs, the cached encryption key can decrypt all four private keys (Ed25519, X25519, ML-DSA, ML-KEM). Active origin compromise is stronger: malicious script or malware running as the KeyPears origin can combine session access, server functions, the cached encryption key, and client-side crypto helpers to sign messages or third-party auth assertions as the user until the session is revoked, keys are rotated, or the compromised client is cleaned.

10.6. Forward Secrecy

KeyPears provides encrypted stored messages, authenticated messages, hybrid post-quantum defense-in-depth, and key rotation. It does not provide message-level forward secrecy or post-compromise security in the Signal sense. This is a deliberate design choice.

TLS 1.3 gives KeyPears forward secrecy for transport sessions: later compromise of a TLS server key does not decrypt previously recorded network traffic. Stored message ciphertext is different. Messages are intentionally stored for later retrieval, and users need retained key material to read their inbox across sessions

and devices. If a user’s long-term X25519 private key and ML-KEM decapsulation key are later extracted, old messages encrypted to those keys are decryptable.

A Signal-style ratchet would reduce that exposure by deriving and deleting a fresh key for every message, but it would add prekey bundles, chain-key state, skipped-message-key handling, multi-device synchronization, and recovery semantics. KeyPears accepts the simpler tradeoff: durable retrieval, federation simplicity, and independent implementability over Signal-style forward secrecy and post-compromise security. Key rotation limits exposure for future messages; message deletion can reduce retained ciphertext, subject to backups and copied data.

10.7. Limitations

KeyPears does not protect against compromised endpoints, weak passwords, or DNS-level attacks. A compromised active client can read plaintext and sign as the user while the session and keys remain usable. SLH-DSA (FIPS 205), a hash-based signature scheme, exists as a fallback against a structural break in lattice cryptography but is not currently used due to its substantially larger signatures (8–50 KB). The Rust PQC libraries used by this implementation (RustCrypto `ml-kem` and `ml-dsa`) have not received an independent third-party audit as of this writing.

11. Related Work

	PGP	Signal	Matrix	KeyPears
Identity	Email addr.	Phone #	@user:domain	name@domain
Federation	Key servers	None	Homeservers	DNS + pull
E2E encryption	Manual	Automatic	Automatic	Automatic
Spam mitigation	None	Phone reg.	Rate limits	Proof of work
Key management	Manual	Automatic	Automatic	Automatic
Post-quantum	No	Hybrid KEM	No	Hybrid KEM+sig
3rd-party auth	No	No	No	Yes
Open source	Yes	Yes	Yes	Yes

Table 1: Comparison of encrypted communication systems.

PGP (1991) provides strong cryptography but requires manual key management. **Signal** (2013) solved usability with automatic key management, but is centralized. Signal’s PQXDH protocol [6] uses hybrid X25519 + ML-KEM for key exchange, but identity signatures remain classical Ed25519. **Matrix** (2014) is federated and encrypted, but uses a Matrix-specific address format and a substantially more complex architecture. **Keybase** (2014) combined social-proof identity with encryption, but was acquired by Zoom in 2020. KeyPears is, to our knowledge, the first federated messaging system with full hybrid post-quantum cryptography for both key exchange and signatures.

12. Future Work

Several extensions are planned. **Group messaging** with multi-party key agreement would extend the protocol beyond pairwise communication. A **native mobile client** with hardware-backed key storage would improve security for the encryption key. Future extensions should preserve the core design goal: a simple federated protocol that many applications can embed correctly.

13. Conclusion

Email demonstrated that federated, human-readable addressing can achieve universal reach. But email's design assumed trusted networks, and four decades of effort have failed to retrofit encryption and spam resistance. Centralized alternatives solved these problems by abandoning federation. KeyPears takes a different path: a simple protocol that preserves `name@domain` addressing and DNS-based federation while making hybrid post-quantum key exchange, composite signing, and proof of work mandatory from the start.

The result is a system where encryption is the only mode, spam is computationally expensive, identity authentication requires no passwords or API keys, and domains remain portable. KeyPears does not pretend hosted servers require no trust; it makes that trust explicit and moveable. If a user does not trust a hosted server to publish honest keys, they can host their own domain. That simplicity is what allows the protocol to be embedded broadly.

References

- [1] A. Whitten and J. D. Tygar, "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0," in *Proceedings of the 8th USENIX Security Symposium*, Aug. 1999.
- [2] A. Back, "Hashcash - A Denial of Service Counter-Measure," Aug. 01, 2002. [Online]. Available: <http://www.hashcash.org/papers/hashcash.pdf>
- [3] M. Marlinspike, "Reflections: The ecosystem is moving." [Online]. Available: <https://signal.org/blog/the-ecosystem-is-moving/>
- [4] National Institute of Standards and Technology, "Module-Lattice-Based Key-Encapsulation Mechanism Standard," Aug. 13, 2024, FIPS203.
- [5] National Institute of Standards and Technology, "Module-Lattice-Based Digital Signature Standard," Aug. 13, 2024, FIPS204.
- [6] Signal, "PQXDH: Post-Quantum Extended Diffie-Hellman." [Online]. Available: <https://signal.org/blog/pqxdh/>
- [7] F. Strenzke and A. Wussler, "Post-Quantum Cryptography in OpenPGP" *Internet Engineering Task Force*, Art. no. draft-ietf-openpgp-pqc-07, Feb. 2025.
- [8] R. Babbush *et al.*, "Securing Elliptic Curve Cryptocurrencies against Quantum Vulnerabilities: Resource Estimates and Mitigations," Apr. 17, 2026. [Online]. Available: <https://arxiv.org/abs/2603.28846>