

KeyPears: Decentralized Diffie-Hellman Key Exchange

An Email-Like Architecture for Secret Management

Identellica LLC
2025-10-26

Abstract

KeyPears is a decentralized secret management system based on an email-like architecture for Diffie-Hellman key exchange. Users can exchange secrets securely between any two email-style addresses (e.g., `alice@example.com` ↔ `bob@example2.com`) without relying on a centralized service. The system supports three primary use cases: password management, cryptocurrency wallet management, and secure messaging. KeyPears operates as a local-first native application with optional synchronization through a permissionless marketplace of self-hosted or commercial node providers.

Introduction

Traditional secret management systems rely on centralized services that become single points of failure and trust. KeyPears reimagines secret management using a federated, email-like architecture where users maintain sovereignty over their data while benefiting from cross-device synchronization and peer-to-peer secret sharing.

The core innovation is applying the email model—where anyone can run a mail server and communicate with users on any other server—to Diffie-Hellman key exchange. This creates a decentralized network for secure secret sharing without centralized gatekeepers.

Architecture

Client-Side Application

KeyPears operates primarily as a native, client-side application available on all major platforms:

- Desktop: Windows, macOS, Linux
- Mobile: Android, iOS

All cryptographic operations occur client-side. The application maintains local vaults—encrypted containers for secrets—that function independently without any server connection.

Email-Style Addressing

Each vault in KeyPears is identified by an email-style address:

`username@domain.com`

This addressing scheme provides:

- **Familiar UX:** Users understand email addresses
- **Decentralization:** Multiple independent domains can coexist
- **Discoverability:** Public keys can be discovered via domain servers

Optional Node Synchronization

Users can optionally connect their vaults to a KeyPears node (server) to enable:

1. **Cross-device synchronization:** Keep secrets in sync across multiple devices
2. **Public key discovery:** Make your vault's public key discoverable to others
3. **Secret exchange:** Receive secrets from other KeyPears users

Nodes are **optional**. The KeyPears client works fully offline without any server connection.

Diffie-Hellman Key Exchange

When a vault is synchronized to a node, its public key becomes discoverable at an API endpoint:

`https://domain.com/api/vaults/username/public-key`

This enables Diffie-Hellman key exchange:

1. Alice (`alice@alicecompany.com`) wants to share a secret with Bob (`bob@bobscompany.com`)
2. Alice's client fetches Bob's public key from `bobscompany.com`
3. Alice's client computes a shared secret using Diffie-Hellman
4. Alice encrypts the secret with the shared key and uploads to Bob's node
5. Bob's client downloads the encrypted secret and decrypts it using his private key

This process is entirely end-to-end encrypted. Neither Alice's node nor Bob's node can read the shared secret.

Decentralization

KeyPears nodes are:

- **Open source:** Anyone can review the code
- **Self-hostable:** Run your own node with full control
- **Federated:** Nodes communicate across organizational boundaries
- **Permissionless:** No central authority controls the network

This creates a marketplace of node providers:

- Self-hosted for maximum control
- Commercial providers for convenience

- Employer-hosted for organizational use
- Community-hosted for open-source projects

Use Cases

KeyPears supports three primary categories of secrets at launch:

Password Management

Store and sync passwords, API keys, environment variables, and other text-based secrets across devices. Share passwords with team members via Diffie-Hellman exchange.

Example: A developer stores AWS credentials locally, syncs them across laptop and desktop, and shares specific API keys with teammates at different companies.

Cryptocurrency Wallet

Manage private keys for cryptocurrency wallets with full self-custody. Sync keys across devices while maintaining end-to-end encryption.

Example: A user stores Bitcoin and Ethereum private keys in KeyPears, syncs between phone and hardware wallet, and shares a wallet backup with a trusted family member.

Secure Messaging

Exchange encrypted messages with other KeyPears users across organizational boundaries.

Example: Alice at Company A sends an encrypted message containing sensitive project details to Bob at Company B, without either company's IT department having access to the content.

Security Model

Three-Tier Key Derivation

KeyPears uses a three-tier key derivation system:

1. **Password Key:** Derived from user's master password using Blake3
2. **Encryption Key:** Derived from password key, encrypts vault data
3. **Login Key:** Derived from password key, authenticates to server

This separation ensures the server receives only a login credential, never the encryption key.

Cryptographic Primitives

- **Hashing/KDF:** Blake3 (fast, secure, modern)
- **Encryption:** ACB3 (AES-256-CBC + Blake3-MAC)
- **Key Exchange:** Diffie-Hellman (X25519)
- **Signatures:** Ed25519 (for message authentication)

Threat Model

KeyPears protects against:

- **Compromised nodes:** Servers cannot read vault contents
- **Network surveillance:** All data transmitted is encrypted
- **Device theft:** Vaults require master password to decrypt
- **Malicious peers:** Public key infrastructure prevents impersonation

KeyPears does **not** protect against:

- **Compromised client devices:** Malware with keylogger access
- **Weak master passwords:** Users must choose strong passwords
- **Social engineering:** Users must verify recipient identities

Implementation

KeyPears is implemented using:

- **Rust:** Core cryptography library and node server
- **TypeScript:** Client applications and web interfaces
- **SQLite:** Local vault storage with append-only logs
- **PostgreSQL:** Server-side synchronization state
- **Axum:** REST API framework for nodes
- **Tauri:** Cross-platform native app framework

All code is licensed under Apache 2.0 and available at:

<https://github.com/keypears/keypears>

Future Work

Planned enhancements include:

- **Group sharing:** Multi-party secret sharing beyond pairs
- **Revocation:** Key revocation and rotation mechanisms
- **Federated identity:** Integration with existing identity providers
- **Hardware security:** Support for hardware security modules
- **Auditing:** Cryptographic audit logs for compliance
- **Mobile-first UX:** Optimized mobile experience with biometric auth

Conclusion

KeyPears demonstrates that secret management need not rely on centralized services. By applying email's federated architecture to Diffie-Hellman key exchange, we create a decentralized network where users maintain control over their secrets while benefiting from cross-device synchronization and secure peer-to-peer sharing.

The system's local-first design ensures functionality without servers, while optional node synchronization enables advanced features for users who choose to

participate in the federated network. This approach combines the best aspects of centralized convenience with decentralized security and control.

For more information, visit <https://keypears.com>