**Yes, it is feasible to build an encrypted, P2P, client-side storage mail system for desktop, but with important practical limitations and trade-offs—especially for true asynchronous "email-like" delivery (messages when recipient is offline) and the strict "100% client-side storage" requirement.** Purely local-only storage without any distributed/temporary network involvement makes reliable async delivery nearly impossible without usability compromises (e.g., real-time only or sender-side retries with presence polling). Most viable approaches rely on **distributed P2P storage** (e.g., IPFS/DHT among participating clients) for temporary/encrypted message holding, which aligns with "client-side" in spirit (no central third-party servers, data encrypted and hosted by peers) but not literally 100% per-user local only.

## Key Existing/Related Projects

Several projects demonstrate parts of this concept (all encrypted by default, P2P-focused, desktop-capable or adaptable):

- **BitMessage** (2012–ongoing, but low activity; last stable ~2018): Closest to a P2P encrypted "email" alternative. Decentralized P2P protocol inspired by Bitcoin. Messages use asymmetric encryption (recipient's public key), addressed via BM-addresses (derived from pubkeys). Messages flood/broadcast the network + Proof-of-Work (anti-spam); nodes store them temporarily (~2 days or configurable TTL) for polling by recipients. No central servers; trustless. Fully client-side key management and local storage of inbox/sent. Pros: Strong anonymity/privacy, resistant to eavesdropping. Cons: Broadcast doesn't scale well (bandwidth/PoW overhead), low adoption, not true 1:1 addressing efficiency. Desktop client available (Python/C++).
- **Ricochet Refresh** (fork/refresh of original Ricochet): Pure P2P instant messaging using Tor hidden services (.onion addresses as identities). Direct peer-to-peer connections (no intermediate servers), E2EE, metadata-free (IP hidden). Storage is local/client-side only. Pros: Excellent privacy, no servers. Cons: Real-time/chat-focused (both parties need to be online; no built-in offline queuing/storage/delivery). Not async email-like. Desktop app.

- **Eppie** (in development/public testing, active): Open protocol for encrypted P2P email + decentralized identity. Aims for providerless/serverless design with full user ownership (BIP39 seed phrases for local accounts). E2EE via elliptic-curve crypto (PGP support WIP). Storage: IPFS (distributed P2P). Transport: SBBS (Secure Bulletin Board System) by default; current testnet uses some cloud nodes (not fully decentralized yet). Bridges to traditional IMAP/SMTP email (e.g., Gmail, Proton, Outlook) and crypto networks (Bitcoin/Ethereum addresses as mailboxes). Desktop GUI (C#/Uno, Windows now; cross-platform planned), local backups. Data exchange P2P/encrypted; distributed storage makes blocking/reading impossible for third parties. Closest ongoing match to your spec, with interoperability.
- **Peergos**: P2P encrypted global filesystem + secure messenger + encrypted email client/bridge. End-to-end client-side encryption (TweetNaCl; private keys never leave client; post-quantum for unshared data). Storage: Decentralized encrypted chunks (Merkle-CHAMP structure, random labels, no cross-links visible), IPFS integration; users run their own instance (self-host or S3-compatible). Direct P2P sharing, secret links for files/messages. No central servers; fine-grained access control. Supports email-like private communication/sharing. Desktop-capable via apps.
- **Quiet**: Encrypted P2P team chat (Slack alternative) with no servers. Syncs messages directly between devices over Tor (hides IP/metadata). Storage: Entirely local on users' devices. Async via channels/direct messages, timed deletion, notifications. Good for small groups; leverages Tor for connectivity. Not full email, but shows serverless P2P async viability for closed networks.

Other mentions: Tox (P2P E2EE IM, no servers), Delta Chat (E2EE over existing email infrastructure, client-side), Session (mixnet with some storage nodes), Peergos-style IPFS-based systems. No fully mature, widely adopted pure async P2P email exists due to the offline/storage problem.

## Technical Feasibility & Challenges

**Encryption**: Straightforward and recommended (client-side keypair generation, publish pubkey as part of address; encrypt with recipient's pubkey; optional forward secrecy via double-ratchet like Signal). Libraries: libsodium, OpenSSL, or age/Sequoia for desktop.

**P2P Connectivity & Discovery**:

- Addressing: Pubkey-derived IDs, DHT (Kademlia in IPFS/libp2p), Tor/I2P hidden services, or blockchain names.
- NAT/firewalls: Hole punching (STUN/ICE), libp2p circuit relays, Tor (built-in relays/hidden services—no public IP exposure).
- Libraries: libp2p (Go/JS/Rust; modular for DHT, transports, pub/sub), IPFS (for content-addressed storage/retrieval), Tor integration (arti or embedded), WebRTC (desktop via Tauri/Electron).

**100% Client-Side Storage & Async Delivery** (the core hurdle):

- Inbox/sent/outbox: Local DB (SQLite, encrypted with SQLCipher or similar).
- Delivery: If both online → direct P2P push. If offline → impossible without storage.
    - Options: (1) Sender queues + retries (needs recipient presence/online status via DHT/polling—inefficient, drains battery/bandwidth). (2) Temporary distributed storage (encrypt message, upload to IPFS/DHT with TTL/expiry; recipient polls/fetches by content hash or user ID). This is "client-side" in that peers (other users' devices) provide the storage temporarily, encrypted so they can't read it—no central server.
    - BitMessage-style flooding + short-term node storage. IPFS with auto-pinning/expiry logic in Eppie/Peergos/Quiet style.
- Strict no-network-storage: Limits to real-time chat or small friend-of-friend (FoF) networks where trusted peers relay/store briefly. Unusable for general email.

**Other Challenges**:

- **Spam/Abuse**: PoW (BitMessage), rate limiting, reputation systems, allowlists, or small-world/FoF graphs.
- **Scalability/Reliability**: Global broadcast fails at scale; use pub/sub topics, hierarchical DHT, or community/interest-based overlays. Messages can delay/lose if peers go offline before propagation.
- **Metadata/Privacy**: Tor/IPFS/libp2p help (hide IP, content-addressed); avoid central indexes.
- **Bandwidth/Performance**: Higher than server-based (replication, retries).
- **Adoption/Network Effect**: Hardest part—P2P needs users online to work well; bridges (like Eppie to SMTP) help onboarding.
- **Security**: Audits critical (BitMessage had RCE vuln); key backup (seeds), revocation, forward secrecy.

**Desktop Implementation Path**:

- **Stack suggestions**: Rust/Go for core (performance, libp2p/IPFS crates), Tauri/Electron for cross-platform UI, SQLite for local storage, embedded Tor (arti), E2EE libs. Or Qt/C++.
- Start small: Friend-only network (manual pubkey exchange) for simplicity/security, add DHT later. Implement queuing + IPFS temp store.
- Open-source examples: Fork/extend BitMessage, contribute to Eppie, or use libp2p/IPFS demos.

**Pros**: Censorship resistance, no provider access to data/keys, true data ownership, E2EE default, potential anonymity (Tor).
**Cons**: Complexity, usability (discovery/offline), performance, limited adoption today.

In summary, **build it if you're prepared for the trade-offs**—focus on distributed encrypted temp storage (IPFS/libp2p) + local DB + Tor/libp2p for connectivity + E2EE. Pure 100% local-only async is not practical for a general mail system. Look at Eppie/Peergos/BitMessage codebases for starting points. This space is active in privacy/decentralization communities; full decentralization is promising but maturing slowly. If you share more details (e.g., real-time vs async priority, anonymity needs, target scale), I can refine further.