Keeping contact across a change of address

Abstract

A domain can expire while the relationship behind it continues. FindMeStill gives each contact a stable identifier and lets the contact sign a sequence of public address records. A recipient who saved the identifier and a verified record can check a later address even after the original host disappears. Distribution can happen through an independent backup, a fresh introduction, or a web page found by searching for the identifier.

The design makes two different failures recoverable. Someone who loses a domain can announce a move if they retain their signing key and history. Someone who loses the browser that held their address book can restore a previously exported snapshot with its recovery code. If both people move, each recovers the other in exactly the same way. Each direction needs some surviving copy or channel.

What a recipient saves

Suppose a customer meets a studio owner and scans the contact link on the owner’s screen. The link contains an HTTPS address and a case-sensitive ID in its fragment. The client checks the signature, derives the ID from the signer’s public key, and saves the exact signed record under that ID. The in-person handoff connects the ID to the studio the customer recognizes. The cryptography can establish continuity only from that starting point.

Each person can host on ordinary HTTPS. The protocol information site publishes the specification and software. It has no role in issuing contact IDs, approving moves, or learning who saved whom. Two people need never use the same application.

The signed card

A contact generates an Ed25519 root key and keeps its private half offline. The ID starts with `fms1-` followed by the unpadded base64url SHA-256 thumbprint of the public key's canonical JWK. The contact signs the ASCII header and payload segments joined by a dot. Adding the signature segment gives the exact compact JWS record that a recipient saves. Every revision names the same root key and ID.

Signed fields	What the verifier learns
Profile, contact ID, public key	Which format and public identity to verify.
Revision, previous record digest	Where the record belongs in this history. The first revision is 1 and has a null predecessor.
Issued at, expires at	When the record was issued and when it must stop being used as a current address.
Primary, up to two backups	Which HTTPS locators may serve the public document.

How a verifier accepts a move

A host returns an HTTP 200 JSON array from the first signed record through its latest. The verifier checks the expected ID, signature, revision increment, timestamps and digest link at each step. The previous digest covers every byte of the previous compact JWS. Re-encoding its JSON can change those bytes. A newer history must contain the exact saved record.

Compare all valid histories with each other and the saved pin, including the immediate predecessor digest committed by the pin. If the client observes two different records for the same revision, it reports the conflict and preserves the pin. Otherwise it considers the highest observed revision including the pin. If that revision has expired, it cannot serve as a current address. A lower, still-live card cannot replace it.

A signer can issue a successor after expiry. An expired predecessor still supplies evidence about history. A client also distinguishes "this address is on file" from "a host returned this revision on this check." It persists an accepted update atomically under the same ID. A missing server supplies no evidence that the owner moved.

The v0.1 document limit is 64 records and 262,144 bytes. Renewals consume revisions too. A contact approaching the limit needs a fresh trusted introduction before the final card expires unless a successor profile has been agreed. Exact serialization, clock bounds and URL grammar appear in the Discovery v0.1 specification.

Recovery when a domain disappears

Consider two contacts who saved one another before a move. The names below use reserved example domains. Neither the lookup nor verification depends on those names being globally unique across the life of the relationship.

1. **Before the move.** The studio publishes its latest signed history at its primary and an independently hosted backup listed in the card. The customer saves the verified pin. The studio keeps its private signing key and a copy of the complete history away from the primary host.
2. **After losing the old domain.** The studio signs the next revision, with the new primary and the digest of its previous record, and publishes the full chain on the backup and the new primary. The customer's client can recover through the backup already present in the saved card.
3. **If every saved host is gone.** The studio publishes a human-readable page at `workshop.example/fms` containing its full ID as visible text and a link to a separate signed JSON document. The customer searches for the quoted saved ID using their preferred search engine, or obtains the new link from a mutual contact. The client verifies the discovered chain against the pre-existing pin before updating the address.
4. **If the customer also moves.** They issue and publish their own successor. The studio independently verifies that move using its own saved customer pin. Availability in one direction says nothing about availability in the other.

When the address book itself is lost

The reference client can export a snapshot of exact ID and signed-record pairs. Each export generates a fresh random 256-bit recovery key and encrypts the snapshot with AES-256-GCM. Keep the file and separately displayed recovery code somewhere you can reach after losing the device or app origin. On a trusted replacement client, provide both to authenticate the file and verify every signature before import. The importer preserves any existing local pins and reports differences.

The snapshot contains public cards, but their grouping can identify a private network of relationships. It carries neither the issuer's private key nor a private message history. A person who loses their root cannot use the book to sign a move. Someone who loses both their pin and their backup needs a fresh trusted introduction to regain that assurance.

Where trust and availability end

Situation	What happens
The old domain changes hands	The new owner can suppress updates or serve a stale copy. A new address for the saved ID requires a valid signed history.
A search result copies a name and ID	The copy may point to a genuine chain or to unrelated bytes. The verifier checks the chain against the recipient's saved pin. A first-time visitor still needs a trusted introduction.
The signer publishes two histories	A client that sees conflicting evidence keeps the pin. A client that sees only one compatible branch can miss the split. The protocol has no global witness.
Every distribution channel disappears	The contact remains in the book. Discovering a new address must wait until its signed history reaches the recipient.
The root is stolen or lost	A thief can sign plausible moves. A lost root prevents further signatures. v0.1 recovery in either case requires a new identity and trusted reintroduction.

Publishing a searchable page is voluntary. It publicly connects one identity to the addresses it uses. A search provider receives the ID in the query and can return stale results, omit the page, or retain earlier locations. The private contact list never needs to be published. A backup file and its recovery code together reveal that list.

Ed25519 aims at about 128 bits of classical security. Against ordinary computers, recovery-code theft, root theft, misleading introductions and compromised clients are more practical concerns than brute-forcing a random key. A sufficiently capable fault-tolerant quantum computer would break the signature assumption. AES-256 contact-book encryption does not protect discovery signatures. A future migration needs a versioned algorithm and identity binding, authenticated while the existing key remains trustworthy, and independent testing for downgrade behavior. v0.1 specifies no such migration. See RFC 8032 §8.5 and the NIST ML-DSA standard.

Private communication and the protected player

Public discovery establishes continuity for an address. An application that delivers messages must also authenticate the conversation, enforce local permissions and decide what to do with received bytes. The following are experimental profiles and implementation results. A client can implement discovery alone.

A move delivered inside a conversation

In the address-update draft, a sender can send a signed move directly to a saved contact, or separately to selected contacts. Each message binds the sender, recipient, unique message ID and full signed chain to their existing conversation. An established contact accepts this type by default and can change the setting for that local identity. Applying a valid move preserves the existing thread. Replaying identical bytes returns the original disposition. Reusing the message ID for different bytes produces a conflict. The application's "applied" result and the relay's storage receipt answer different questions.

Receiver classification and scoped settings have been exercised with shared synthetic vectors. The media pilot below demonstrated end-to-end delivery of text and WAV. That demonstration did not establish end-to-end deployment of typed address updates.

Intermittent delivery and playback

The paired media pilot sends an encrypted envelope to a relay. A client can resume a download in 256 KiB ciphertext ranges. It checks the receipt digest and authenticates the complete envelope before releasing plaintext. Each app can use its own same-origin gateway while the relay sees only scoped requests. The relay can still observe message sizes, timing and mailbox use. The shared pairing secret provides no forward secrecy.

The recipient explicitly opens a message. Text is rendered as text. For supported media, trusted player code screens a bounded buffer and passes accepted bytes to a browser decoder in a sandboxed frame. Both clients can ship the player code and load it from a dedicated, cookie-free player site. Only the selected bytes cross into the frame. WAV, MP4 and WebM are the pilot's allowed containers, with an 8 MiB content limit. A decoder vulnerability is still possible. Authentication establishes the integrity of the encrypted content. Decoder isolation addresses a separate risk during playback.

A separately paired listening-agent pilot can decode audio in an isolated, network-disabled process. A transcript is input data. It supplies no authority to call tools, open links or act on instructions in a recording. Forwarding audio to an external service requires its own authorization. The media and resume profile documents these boundaries.

How to check an implementation

Start with two separately hosted copies of a synthetic signed history and a client with a fixed test clock. Verify the first record from the introduction. Stop the original primary, publish the signed successor at the known backup and new primary, and refresh. The success condition is a changed locator, an unchanged contact ID, and an exact pin equal to the signed successor. Repeat with an older reply and with conflicting signed histories. Record both the user-visible outcome and the bytes left in storage.

The published reference includes an independent Python issuer, a Python backup host, a Node HTTPS host and a Web Crypto verifier. Its local conformance exercise stops the original host during a move. A separate browser exercise restores two encrypted address books offline on a fresh app origin. One contact recovers by backup and the other checks a public HTML lead against a saved pin. These exercises use synthetic contacts and a controlled clock where required. They establish interoperability for those cases. Search engine indexing, delivery to a physical phone, and an independent security audit require separate evidence.

Reproduce the discovery exercise

Check out revision `fe6461f258e64289bd6824b4501a0b9305769bf2` from the public repository. Use Node.js 24, Python 3 and OpenSSL 3. Run `npm ci`, `npm test` and `npm run build`. The fixture carries a public test key and its own clock. Use a freshly generated offline key for a real identity.

Specifications, vectors and further reading

- Reference release source and commands and documented interoperability exercises.
- Signed predecessor and fork vectors and the encrypted contact-book format.
- Publishing a searchable contact page.
- JSON Web Signature, JWK Thumbprint and fully specified JOSE algorithms.

FindMeStill is an open protocol. The reference project is available under the Apache License 2.0.