E2EE Key Storage Evolution

& Passkey PRFs

Sean Byrne

Cork|Sec 156 - June 2026

ABOUT ME

Who I am and why I care about this

Security engineer.

Dropbox, Patreon, Truework, & Vannevar Labs

Currently AuthDuty & EndEngine

Previously founded an End to End Encryption (E2EE) messaging startup.

That's where this interest started.

E2EE is technically solved. Operationally it's painful.

Usability, engineering complexity, speed of development all push against deploying it well.

I track anything that makes good E2EE more usable.

THE PROBLEM

Security vs Usability

E2EE Systems

"Save this recovery phrase
or your data is gone forever."

"Choose a PIN to protect your backup."

"Where do you want to store this key?"

"Write this down. Do not screenshot it."

"If you lose this, nobody can help you."

Most people do not have a password manager.

TELEGRAM

Enter phone number.

You're chatting.

That's it.

Not E2EE by default. Server reads your messages. **Doesn't matter. Users chose less friction.**

Three generations of E2EE key management

GEN 1

Password

GEN 2

PIN + hardware

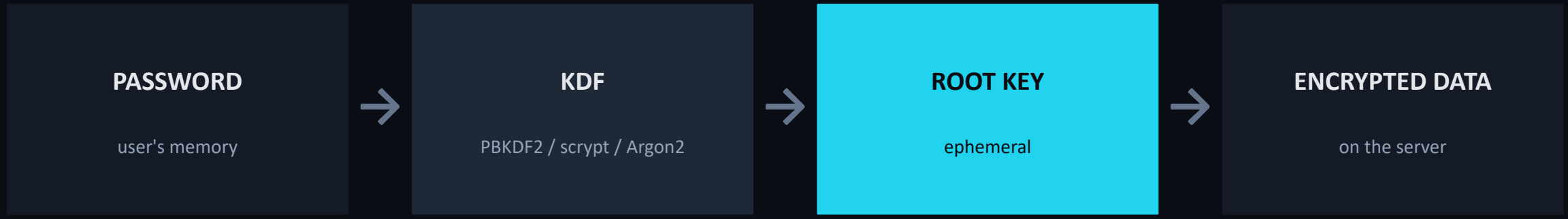
GEN 3

Trusted devices

GENERATION 1 / PASSWORD-DERIVED

The first answer to "where does the root key live":

Derive it directly from the user's password.



Server never sees the plaintext root key. Only the encrypted data.

… a human picked the secret

The hardware protects the key. The key is unlocked by something a human chose.

`123456`	`password`	`admin`
used 4.5M+ times	*still top 5*	*still top 10*

No enclave saves you if the thing that unlocks it is "123456".

Just generate & store the keys in devices' secure hardware.

Problem solved…

Thank you for coming.

…right?

Key storage is key recovery

You lose your device.
How do you get your keys back?

The security key model

The official advice:

Buy two hardware security keys.

Register both with every service you use.

Keep one backup in a fireproof safe.

Or a safety deposit box.

Then you sign up for a new service.

Keep it locked away securely.

Also have it with you whenever you need it.

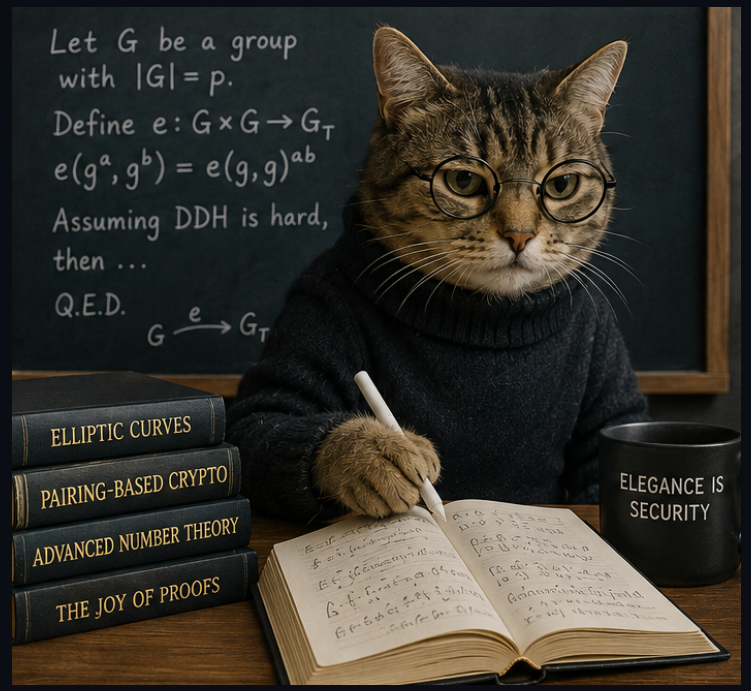
For every new service. Forever.

The hard part of cryptography is the keys.

Specifically, how you get them back when you lose your phone, buy a new phone, etc.

Cryptography vs cryptographic engineering

Cryptographer

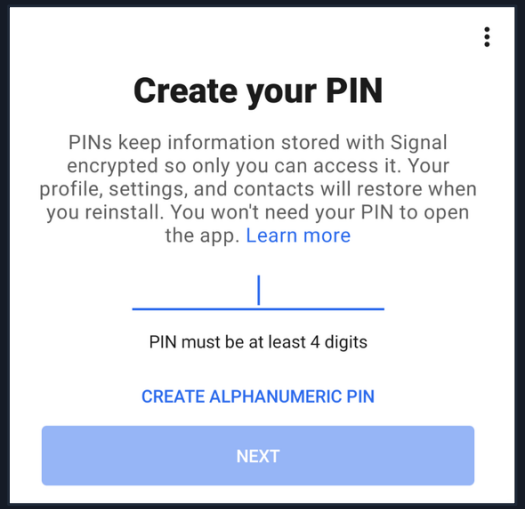


Proves security

Cryptographic Engineer

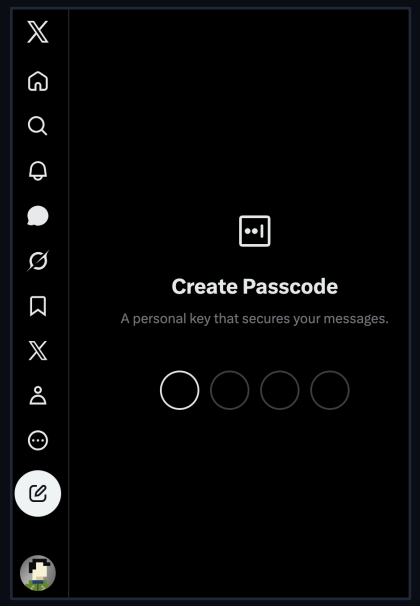
Balances security, engineering, and usability

What recovery looks like today…



PIN or Password

Signal



4-digit PIN

X (Twitter) encrypted DMs

Recovery key card

Apple ADP, Proton

Where does the root key live?

USER'S HEAD

as a password

DEVICE

key materializes momentarily

SERVER

only encrypted blobs

When you lose all your devices, where does access come back from?

Your password, that you remembered.

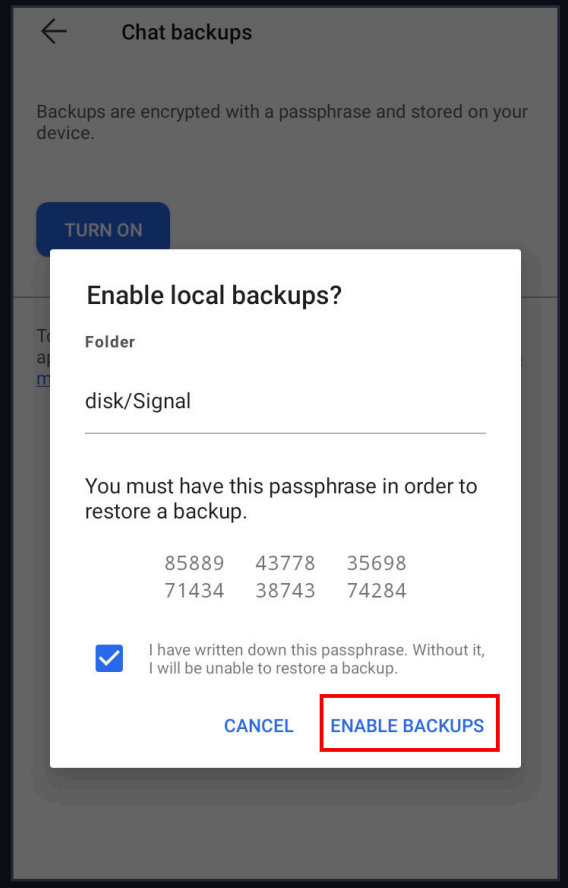
Signal's early design problem

Device-bound keys for messaging work well.

But what about backup?

What if the user loses their phone? Or doesn't manually back up?

Signal needed recovery.



The PIN problem

The natural answer: let the user pick a PIN to protect a remotely stored backup key for remote backups.

1234	0000	1111	4321	0852

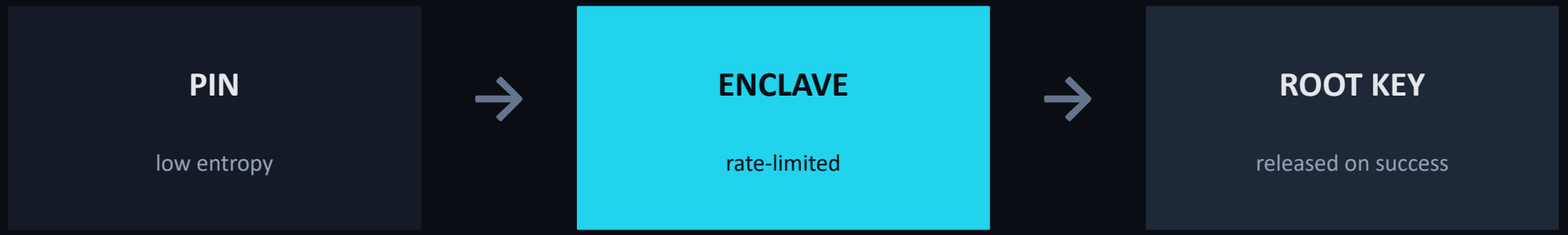
A weak PIN protecting a backup key on a server is **trivially brute-forceable**.

> ***So how do you use a low-entropy secret safely?***

PIN + hardware + rate limiting

Signal's answer (SVR): the PIN-derived key lives inside a secure enclave.

The enclave enforces a hard guess limit. After N wrong attempts, the secret is destroyed.



The factor we usually skip: phone number control + PIN + hardware.

SIM swap is a real attack vector. Signal's Registration Lock requires the PIN at re-registration to defend against it.

What you're trusting: *the enclave implementation, the carrier (for the phone number), the operator's provisioning, the rate-limiting logic.*

Other Gen 2 Solutions

WhatsApp

HSM-backed vault

Same shape as Signal. For years, backups to Google Drive / iCloud broke E2EE entirely. Eventually fixed.

Juicebox

distributed-trust variant

Pushes the pattern further: Shamir secret sharing + OPRF across multiple operators. Co-designed by Moxie Marlinspike.

X / Twitter

Juicebox, but...

Uses Juicebox, but runs all the realms themselves. The Juicebox authors publicly said this defeats the point.

Where does the root key live?

USER'S HEAD

a PIN — much smaller

DEVICE

messaging keys, identity keys

SERVER + HARDWARE

PIN-gated enclave

When you lose all your devices, where does access come back from?

Your phone number, your PIN, and a hardware enclave willing to help.

Apple's Advanced Data Protection

No user-memorized secret in the critical path at all.

Recovery happens through the device trust graph,
not through a remembered PIN.

A structural break from Generation 2.

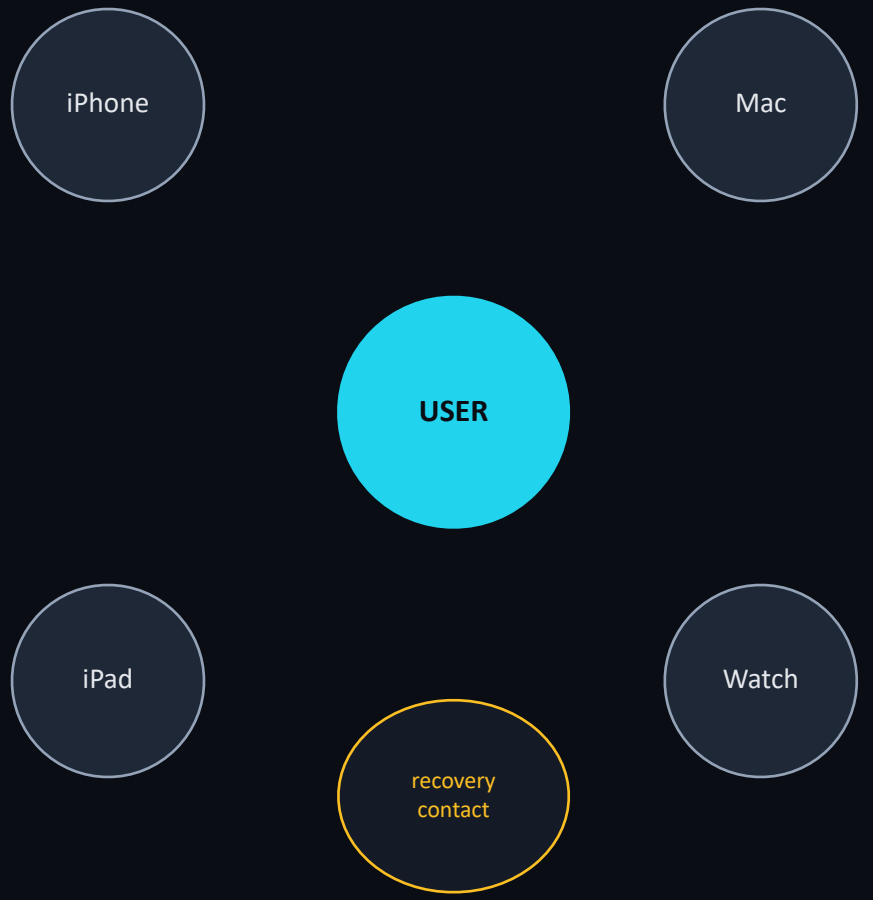
The device trust graph

Keys sync across a set of devices the user has explicitly trusted.

Recovery options:

- Another trusted device
- A recovery contact
- A printed recovery key

Trust is in the device set, not user memory.



Where does the root key live?

USER'S HEAD

nothing — zero in steady state

DEVICES (TRUSTED SET)

root key spread here

SERVER

encrypted blobs only

When you lose all your devices, where does access come back from?

Another device, a person, or a piece of paper. Nothing you have to remember.

THE TREND

Three generations, away from user memory

GEN 1

Password

(user memory = everything)

GEN 2

PIN + hardware

(user memory matters less)

GEN 3

Trusted devices

(user memory matters not at all)

Worth noticing: Moxie Marlinspike walked this entire arc personally.

Signal SVR (Gen 2). Then Juicebox co-author (Gen 2.5). And we're about to see Gen 4.

Passkeys: what changed

OLD FIDO2

Credentials were device-bound.

Lose the device, lose the credential.

No sync. No recovery. That's why adoption stalled.

PASSKEYS

Credentials sync through provider infrastructure.

iCloud Keychain, Google Password Manager, 1Password.

New device → credentials are already there.

Why this matters for us: **recovery is not a separate ceremony. It's built in.**

Any key material derived from a passkey inherits the same sync and recovery properties.

Callback: password reuse is **structurally impossible**. Each credential is bound to one relying party.

Same passkey, both jobs

LOGIN

authentication credential

ENCRYPTION

root key

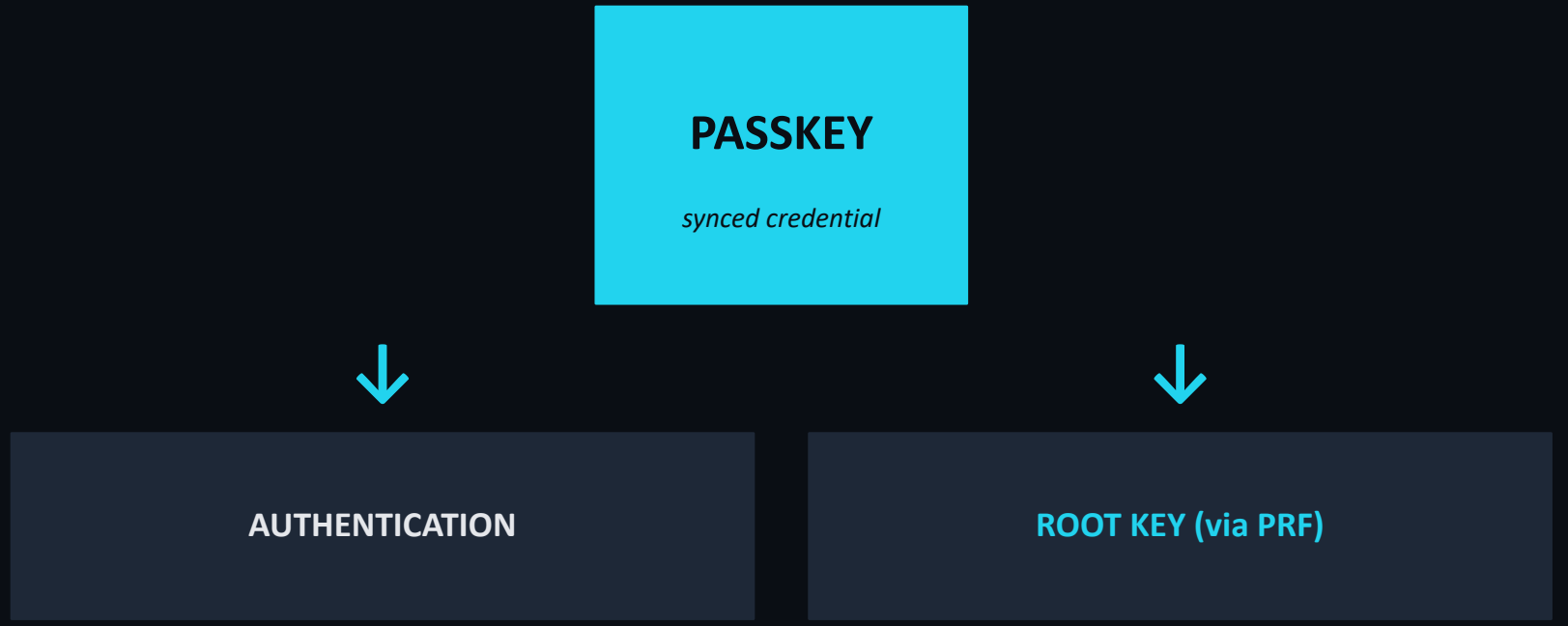
ONE PASSKEY

does both

One thing to lose, not three. Recovery inherits from passkey sync. Hardware-backed by default. Same UX as login.

> *"Cryptography turns data security problems into key management problems."*
> — Moxie Marlinspike, Confer (Dec 2025)

Where does the root key live?



Remember WhatsApp from earlier? Their encrypted backups now **default to passkey**, not password.

Same product. Two generations. In production right now.

The PRF extension

A passkey can now produce cryptographic keys, not just authentication signatures.

The WebAuthn PRF extension. Application supplies a salt. The authenticator returns a deterministic high-entropy value, derived from the credential.

Backed by HMAC under the hood. Originally the hmac-secret CTAP extension, designed for disk encryption.

When did this become deployable?

The primitive is old. Cross-platform availability is recent and uneven.

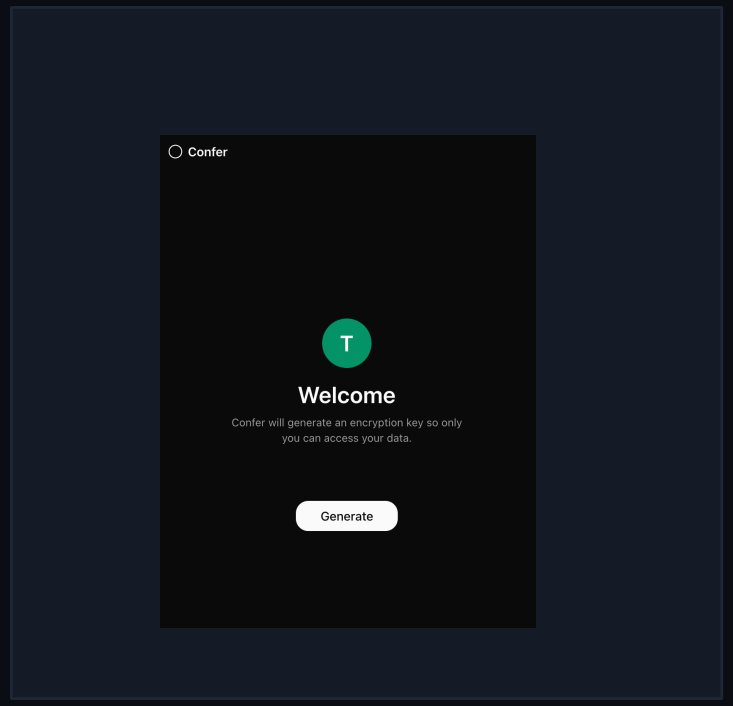
2018	**CTAP2 hmac-secret**	The underlying primitive. Hardware keys (YubiKey). Built for disk unlock.
2023	**Chrome / Chromium**	First browser. Security keys and hybrid (phone-as-authenticator).
2024	**Android**	Google Password Manager passkeys. Now the most robust support.
Sep 2024	**Apple: iOS 18, macOS 15, Safari 18**	Via iCloud Keychain. Puts PRF on a billion-plus devices. The inflection point.
2024-25	**1Password, Dashlane, Bitwarden**	Password managers derive vault keys from PRF.
Feb 2026	**Windows Hello**	Matters for enterprise Windows fleets.

Why Confer (Dec 2025) and WhatsApp (Oct 2025) ship now: the platform plumbing only landed at scale in late 2024.

Confer.to

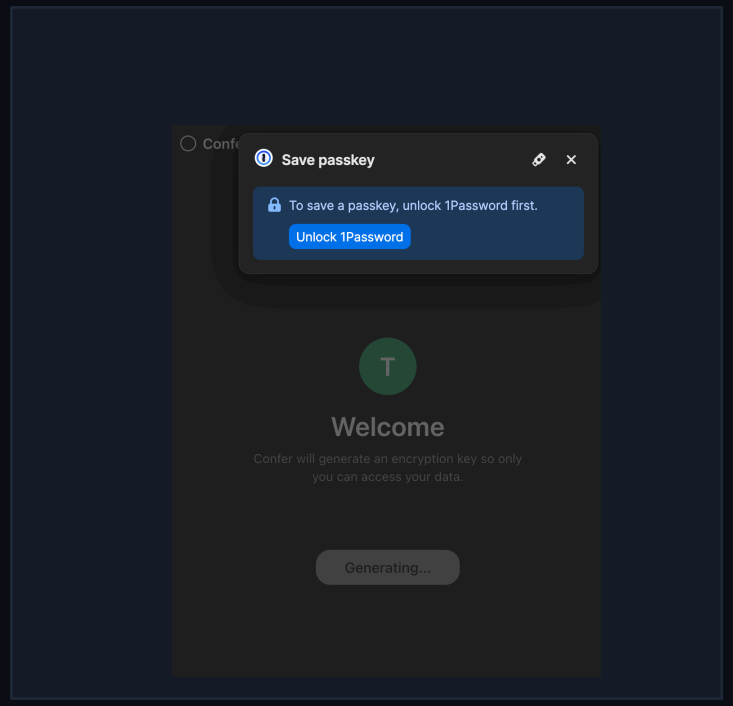
An E2EE AI Chatbot that uses Passkey PRF's for Encryption keys.

What the user actually sees

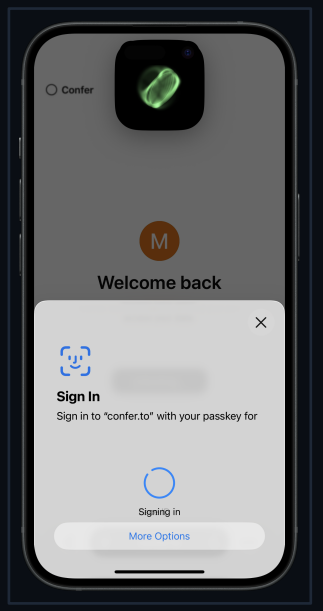


Sign up. The passkey is created on your device.

PRF-capable credential registered. No password.



PRF derives the key fresh each session. Server never sees it.



Another synced device.

Key recovery inherits from passkey sync.

Concrete construction (Confer)

Confer is Moxie's new E2EE AI chat product. The cleanest production example.

```
// 1. Request a PRF output from the user's passkey
const assertion = await navigator.credentials.get({
  publicKey: {
    extensions: { prf: { eval: { first: salt } } },
    // ...
  }
});

const prfOutput = assertion.getClientExtensionResults().prf.results.first;

// 2. Derive the app's root key via HKDF
const rootKey = await hkdf(prfOutput, { info: 'confer-root-key-v1' });

// 3. Envelope-encrypt arbitrary data (incl. asymmetric private keys)
const wrappedDEK = await wrap(dek, rootKey);
```

Envelope encryption *(PRF-derived KEK wraps per-data DEKs) handles any practical case, including asymmetric.*

WhatsApp

The largest E2EE deployment on earth. 3 billion users. Watch one product move generations.

BEFORE 2021 — no E2EE

Backups in plaintext

Chat backups to Google Drive / iCloud were not encrypted. The E2EE messages were safe; the backup was a hole.

→

2021 — Gen 2

Password + HSM vault

E2EE backups launch. A 64-digit key, or a password protecting it in an HSM Backup Key Vault. OPAQUE PAKE, rate-limited, key destroyed after too many guesses.

→

2025 — Gen 4?

Passkey-backed

Backups now use a passkey. Fingerprint, face, or device lock instead of a password. Mechanism (PRF, or passkey-auth to the existing HSM vault) not publicly confirmed.

It skipped Gen 3 entirely. Gen 2 to passkey, same product, four years. The trend, in production, at three-billion-user scale.

What got fixed?

- ✓ User Experience & No scary text “You will loose everything unless you guard this with your life!!!””
- ✓ No per service PINs
- ✓ No backup key escrow ceremony
- ✓ No recovery phrase to write down
- ✓ One core solution for the entire tech industry
- ✓ One credential lifecycle instead of two

The bigger win:

First time E2EE has a recovery story competitive with non-E2EE.

Telegram beats Signal not on security but on "log in with your Google account." Passkey PRFs close the gap.

Enterprise E2EE?

- Every iPhone, Android, Mac, Windows endpoint
- Secure enclave or TPM
- WebAuthn: a standard interface
- Biometric UX, already solved
- Enterprise
- E2EE Slack, Dropbox, Teams, etc.

Already in your employees' pockets.

Every employee now carries hardware-backed key material by default.

The trust relocates: from servers to client devices that already exist.

Passkey PRFs are
not a cryptographic breakthrough.

They are an engineering simplification that collapses two long-separated key management problems into one.

A generational shift in cryptographic engineering.

Maybe the first time E2EE has a recovery story usable enough to compete on convenience.

How we got here

The work that brought passkey PRFs from spec to production E2EE.

Adam Langley + WebAuthn WG	wrote the PRF extension spec
Matthew Miller	Jan 2023 — introduced PRF-for-encryption to web developers
Kevin Lewi (WhatsApp)	RWC 2023 — documented the OPAQUE + HSM backup design (Gen 2)
1Password (Rene Leveille, Max Crone)	Jul 2024 — first major production ship; open-sourced their library
Filippo Valsorda	Jul 2025 — file encryption with cryptographer-grade design (age / typage)
Moxie Marlinspike	Dec 2025 — Confer; the clearest public framing of why this matters
WhatsApp	largest-scale deployment, via passkey-backed encrypted backups
Eric Chiang	BSidesSeattle talk covering the HSM side of this story

All publicly available. Read the primary sources.

Thank you.

Questions?

Sean Byrne

Conic.AL