State of Frostr

Intro / What is Frostr

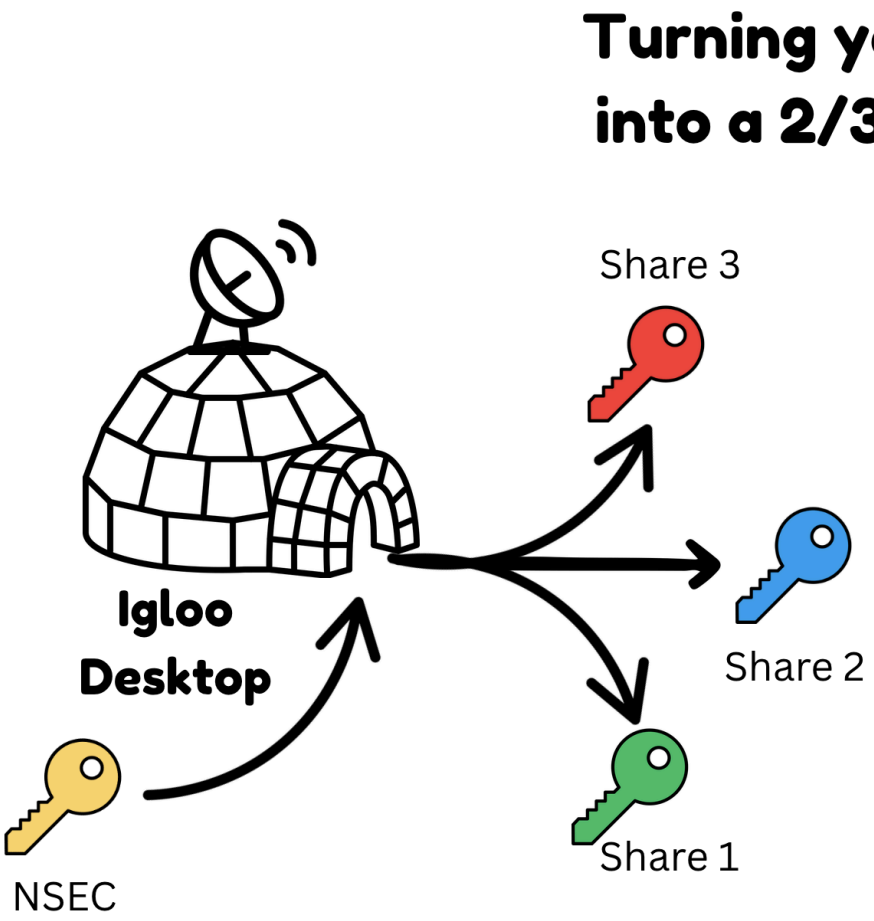
Hello Nostriches and obscure cryptography geeks, welcome to Frostr! I am Austin (writing this) one of two devs working on this project. Frostr is an open source protocol and suite of apps that is attempting to bring simple k-of-n threshold signing to Nostr. What does this mean? It means we want every Nostr user to be able to leverage a FLEXIBLE multi-key setup for their private key.

Split your NSEC into a Frostr Keyset

Frostr allows you to take your existing NSEC (or generate a new one) and split it into a total number of sub keys (called “shares”) and a required threshold for those keys. Meaning your one NSEC could be powered by 3 keys but you only need ⅔ to successfully sign a note (or recover your original NSEC). When you split up your NSEC with Frostr you will end up with a “keyset” which is your set of “shares” (the individual private keys) and then a “group” (basically the public key & info about the keyset). Shares will be prefixed with “bfshare1” and the group with “bfgroup1” for example.

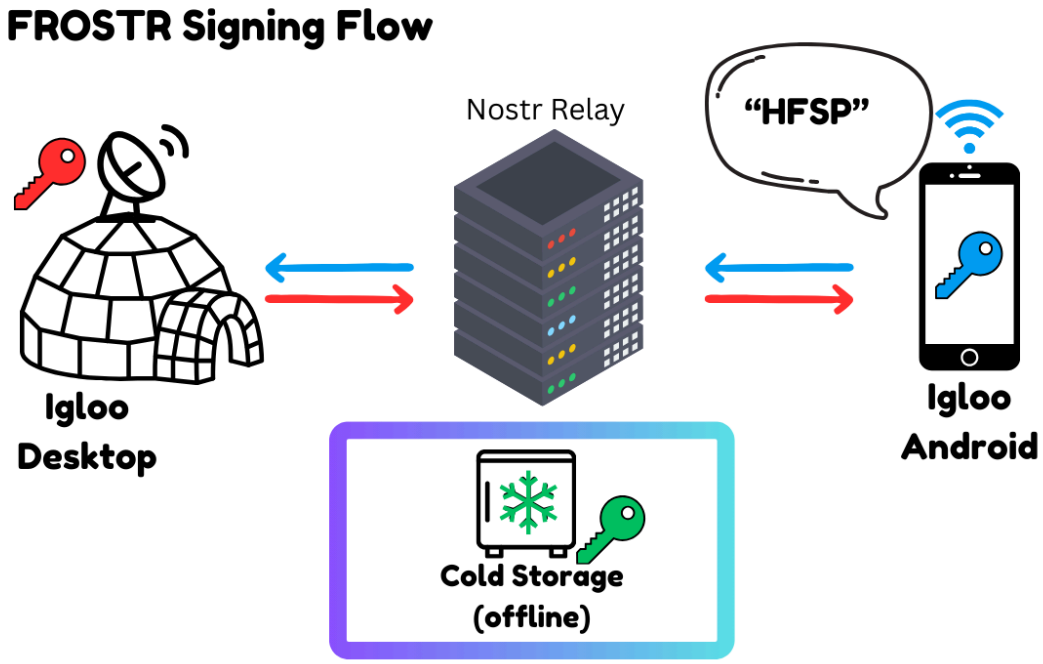
GROUP:
bfgroup1qtm33…

SHARES:
bfshare1qq3ty2…
bfshare12x3y2e…
bfshare1pp2li2…



Signing notes with Frostr

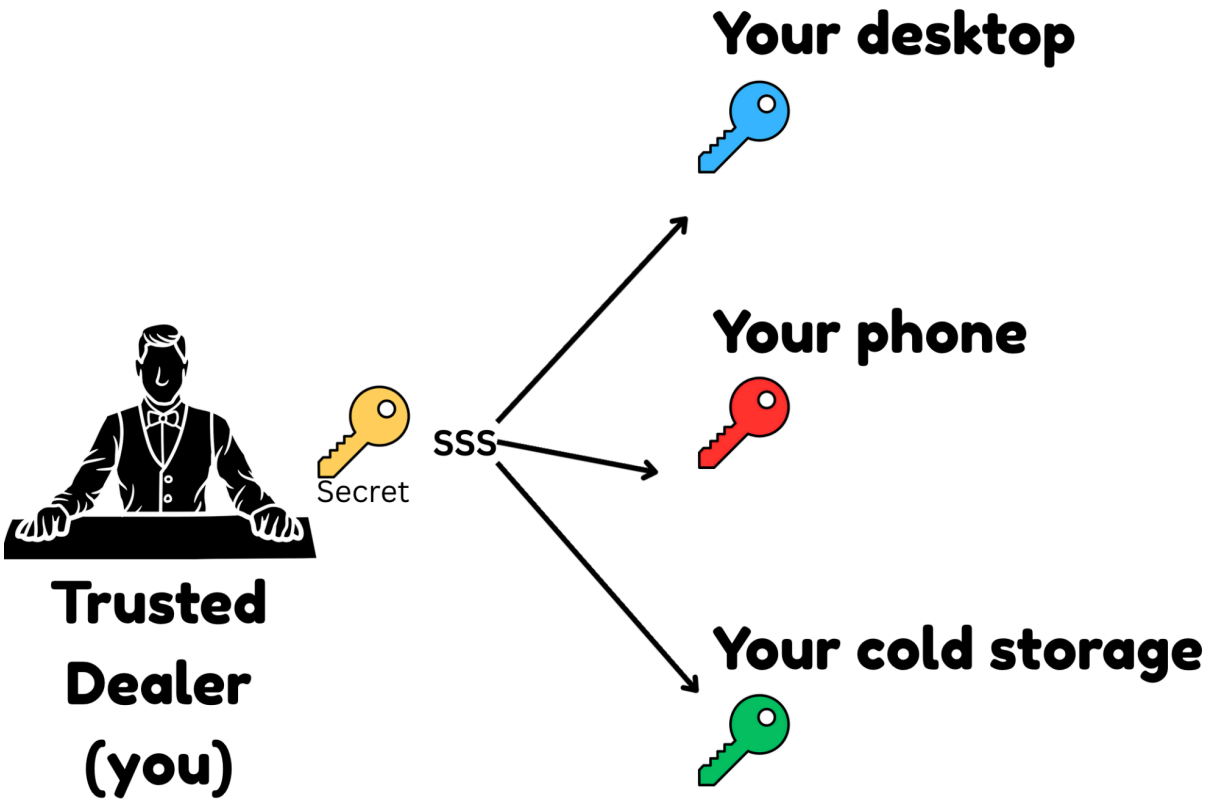
With your own keyset you can begin to utilize the Frostr ecosystem of Apps to run signer for your threshold of shares. If you have a ⅔ Frostr keyset and you want to sign notes you will need to run two different Frostr signers with two different shares from that keyset. With both of your signers running you can now log into a Nostr client and start posting notes! When you go to sign a note (or encrypt / decrypt for DM's) your shares will talk to each other over Nostr relays and collaboratively create a valid threshold signature in an encrypted note. A good way of thinking about it is that your shares are actually their OWN Nostr identities that talk to each other and DM in order to create valid signatures. Pretty cool, right!?



Trusted Dealer vs Distributed Key Generation for FROST

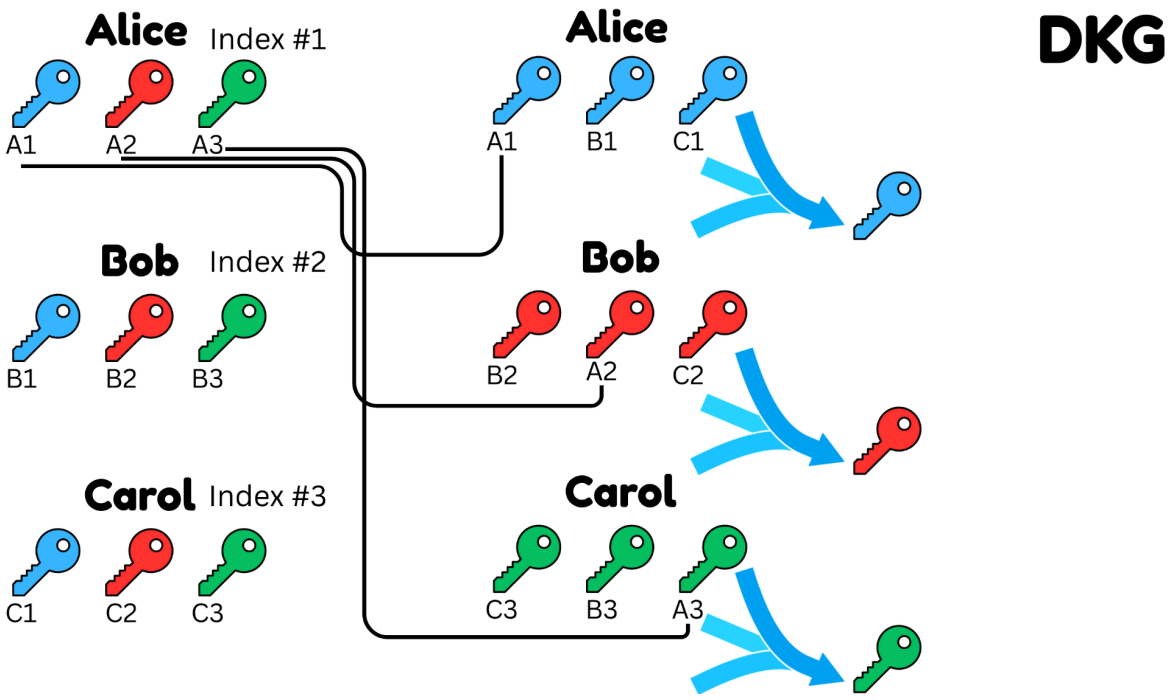
Frostr uses a “trusted dealer” scheme for FROST instead of DKG (distributed key generation). Trusted dealer means that the underlying secret (NSEC) is known or generated in one place

upfront. DKG on the other hand is a flow for generating the secret in a distributed manner.



Trusted dealer works great for Frostr at this time because:

1. Frostr is focused on building for the individual first and foremost; you CAN trust yourself to be your own trusted dealer
2. Trusted dealer allows you to bring an existing NSEC instead of needing to generate a new one to leverage FROST signing
3. Greatly reduces protocol surface and complexity
4. DKG can always be added later as an optional setup step (for nostr keypairs controlled by multiple people)



Recovery

One of the most important — and most misunderstood — features of Frostr is **recovery**.

With a Frostr keyset (for example, a ⅔ setup), the same threshold that allows you to produce valid signatures can also be used to **fully reconstruct the original NSEC**. In other words: Any threshold of shares within a given keyset can recover the underlying NSEC.

This is intentional, and powerful - It means you can:

- Split a single NSEC into many different keysets
- Use different total share counts and thresholds over time
- Rotate, retire, or replace devices
- And still always recover *the same* original Nostr identity — **as long as you control the threshold of shares for that keyset**

This gives Frostr both flexibility and longevity: your identity is not tied to any single device, app, or signer. At first glance, this might sound dangerous. If a threshold of shares can reconstruct the full NSEC, what happens if a share is lost or stolen?

This is where Frostr makes a **very explicit design choice**:

- Signing is distributed and interactive.
- Recovery is manual, local, and non-interactive.

When signing:

- Shares can communicate over Nostr relays
- A single running signer may receive a request to help produce a threshold signature
- No share ever reconstructs the full private key

When recovering:

- There is no network protocol
- No relay messages
- No remote coordination
- No automatic flow

Recovery only happens when you manually gather the threshold of shares and enter them locally into a Frostr app, which then reconstructs and displays the NSEC. If one of your shares gets stolen it is possible for the attacker to get a note signed but since recovery is local only they will not be able to steal your NSEC and you will be able to rotate (or orphan them) out of the keyset as long as you still control the threshold of shares.

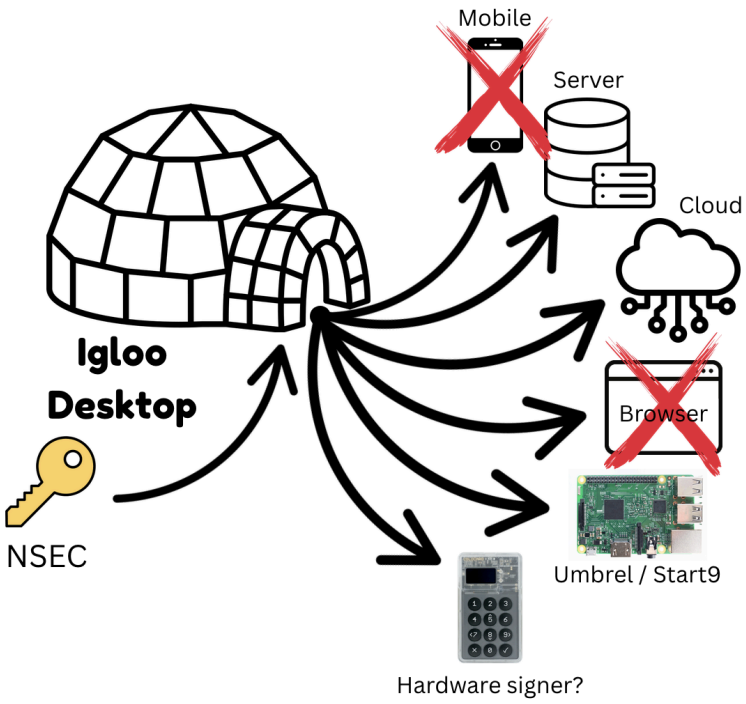
A crucial property of Frostr is that shares are not globally meaningful - they only work:

- With their original keyset
- With their original group parameters
- With their original threshold configuration

When you destroy a keyset and recreate a new one:

- Old shares cannot be mixed with new shares
- Old shares cannot recover the NSEC
- Old shares cannot participate in signing

This is why losing a device or even multiple devices does not imply catastrophic loss, as long as the threshold remains intact.

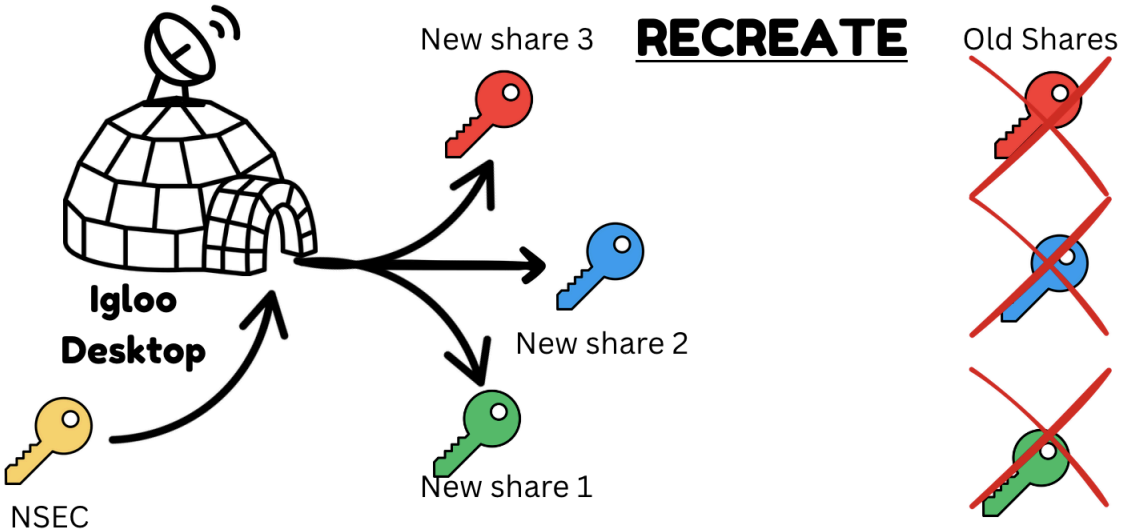


Lost a Device or a Key?

So recovery on Frostr today is done **manually** through **destruction** and **orphaning**

THAT'S OK!

Just DESTROY and RECREATE



Shares are only valid WITHIN their given keyset

Origin of Frostr

Frostr came from the big brain of @cmdruid (on github) aka @btctechsupport (on X) @cmd (on Nostr) Topher (in person); you get it, man with many nyms. After seeing the first official RFC specification for FROST cmd built his own FROST library to the spec and soon after saw an opportunity to apply this paradigm to Nostr signing. CMD realized that Nostr keys within the FROST paradigm could:

- Be split into any number of total keys with an arbitrary signing threshold
- Produce standard, valid Schnorr signatures
- Perform ECDH operations for encryption/decryption in a distributed way
- Be recombined to recover the original NSEC
- Be re-split again and again into new keysets with different thresholds
- All **without ever needing to create a new Nostr keypair**

This was an exciting insight! As soon as CMD told me about this possibility I pressed him to work on it and expressed my interest to get involved.

> Unbeknownst to us at the time, **Nick Farrow of Frostsnap** had *already* built a FROST-over-Nostr example — and had even used the same name: **Frostr**.
> (😅 Sorry Nick, we already bought the domain.)
>
> For the record, the real OG “Frostr” credit belongs to Nick. His original work lives here: https://github.com/nickfarrow/frostr

TABCONF 2024: From Idea to Demo

After some early conversations, CMD and I decided to use the **TABCONF 2024 hackathon** as our forcing function to turn this idea into something real.

Our goal was intentionally minimal:

- A simple **desktop app** that could ingest (or generate) a Nostr NSEC
- Split that NSEC into **FROST shares**
- A lightweight **NIP-07 browser extension** that could run one of those shares as a signer
- And a full end-to-end demo showing a note being signed and published using a **2-of-3 threshold**

We wanted the most barebones possible proof that the whole thing actually worked:

- Split an NSEC
- Run signers in different environments
- Produce a valid Nostr signature with 2-of-3 shares

No new signing standards. No protocol changes. No magic. Just threshold signing, end to end.

That hackathon is also where the first versions of **Igloo Desktop** and **Frost2x** were born — and yes, we ended up winning. 🏆

The last year

Bitcoin++ Austin 2025

In early 2025, we gave our **first public presentation on Frostr** at **Bitcoin++ Mempool Edition in Austin, TX**.

This talk was the first time we walked a live audience through:

- The motivation behind Frostr
- How FROST maps onto Nostr signing
- Why threshold signing solves real-world key management problems
- And how Frostr fits cleanly into existing Nostr signing standards without protocol changes

It was an important moment for us — not just to explain *what* Frostr is, but *why* it exists, and to validate that the mental model resonated with a room full of Bitcoiners.

Bitcoin Vegas 2025

At **The Bitcoin 2025 conference in Las Vegas**, we took Frostr out of slide decks and into the real world with our **first live, hands-on workshop** — and ended up making a little bit of Nostr history in the process.

This workshop wasn't just a walkthrough. We live demoed Frostr end to end, in front of a room full of developers, using real devices, real signers, and real Nostr relays. Attendees watched as we split a Nostr key, distributed shares across many environments, and coordinated signers live on stage.

Midway through the demo, things escalated.

We began onboarding more and more signers from the audience ALL on phones, all participating in a single Frostr keyset. What started as a small threshold quickly grew into the **largest threshold signature ever produced on Nostr**.

At peak, we had **a dozen signers online**, coordinating partial signatures in real time, all contributing to a single valid Schnorr signature that looked completely normal to the client and relays observing it.

No protocol changes.
No special relays.
No visible difference on the wire.

Just a standard Nostr signature backed by the largest multi-signer quorum Nostr has ever seen!

Doing this live forced us to confront real constraints:

- UX friction when onboarding signers
- Network latency and relay behavior
- Partial signer availability
- Threshold dynamics under live conditions

It was chaotic, stressful, and incredibly validating — and it directly shaped many of the improvements that followed in Frostr's tooling, reliability, and mental models.

📽️ The full edited workshop is available here:
https://www.youtube.com/watch?v=J1WG_InBsHg

OpenSats

In June 2025 OpenSats announced their support of Frostr in the twelfth wave of Nostr grants! https://opensats.org/blog/twelfth-wave-of-Nostr-grants

This was a huge turning point and milestone for us as we were now able to dedicate much more time and focus to the project and create a full roadmap that we believed could deliver on the dream of an easy multisig Nostr setup for any user.

- Say more here maybe??

The state of Frostr - what you can do today

Today, Frostr is no longer a prototype or single demo. It is a working ecosystem made up of **multiple production-ready apps and shared libraries**, covering the full lifecycle of threshold key usage on Nostr.

Keyset creation & management

You can **create, inspect, and manage Frostr keysets** (split an NSEC into shares, define thresholds, export/import data) using:

- **Igloo Desktop**
- **Igloo CLI**

These tools act as your primary “key workshop” for setting up and maintaining Frostr keysets.

Running signers (holding and operating a single share)

You can run an **individual Frostr share as an active signer** across many environments:

- **Igloo Web**
- **Frost2x** (browser extension)
- **Igloo CLI**
- **Igloo Desktop**
- **Igloo Server**
- **Igloo Android**

Each signer holds exactly **one share** and participates in signing only when requested. No signer ever reconstructs the full private key during normal operation.

Initiating signing requests (by Nostr signing standard)

Frostr intentionally integrates with existing Nostr signing specs instead of inventing new ones. Today, signing requests can be initiated via:

- **NIP-07**
  - Frost2x
- **NIP-46**
  - Igloo Server
- **NIP-55**
  - Igloo Android

From the client’s perspective, these flows look identical to standard Nostr signing—Frostr remains invisible on the wire.

Shared libraries powering the ecosystem

Frostr’s apps are built on a small set of reusable libraries:

Protocol libraries

- **bifrost** – core FROST cryptography and share coordination
- **nostr-connect** – Nostr-based communication primitives for signers

Application library

- **igloo-core** – shared logic used across Igloo apps (keysets, storage, flows, UX helpers)

These libraries make it possible to build new Frostr signers and tools without re-implementing the protocol.

What this means in practice

- You can **split a real Nostr key today**
- Run signers on **multiple devices and platforms**
- Sign notes, DMs, and events using **standard Nostr clients**
- Recover your original NSEC when needed
- Do all of this **without changing relays, clients, or protocols**

Frostr already works end-to-end—what's evolving now is **UX, coordination, and polish**, not core capability.

Learnings - integrating Frostr with Nostr signing specs

One of the main goals of Frostr from the beginning was for us to be able to make ALL of this work without needing to break / change any of the existing signing specs (therefore requiring the whole Nostr ecosystem to migrate, do extra work, and make changes). If Frostr can work with each existing signing standard, it not only saves users / devs time and effort, but also enables maximum interoperability and privacy. By strictly adhering to the signing standards a client already expects, neither the client nor the relay can distinguish a Frostr user from someone signing with a single key, even though the signature was produced using a distributed key setup. This means Frostr can be adopted unilaterally and incrementally, without coordination or permission, while remaining indistinguishable on the wire from standard single key signing TLDR; security upgrades with zero protocol surface area!

So there are clearly a lot of advantages to integrating with existing signing specs vs making our own, but this has not been easy or straightforward for us to build in practice.

NIP-07:

NIP-07 was the first and easiest Nostr signing spec to integrate Frostr with. Our initial hackathon win at TAB CONF 2024 was showcasing creating the threshold signature for signing a Nostr note through the NIP-07 interface using our Frost2x browser extension (fork of Nos2x). NIP-07 definitely represents the most simple and unintrusive signing interface for Nostr, making it a great starting point for doing anything novel with Nostr private keys / signatures.

This is NOT to say we haven't had issues and bugs using Frost2x across various clients. The experience is still not perfect and there is A LOT we can do to improve it. Very likely we will write a new Frostr NIP-07 extension from scratch (since currently Frost2x is just a modified fork of Nos2x) when we revisit this interface soon.

NIP-46:

NIP-46 has been the most difficult Nostr signing spec for Frostr to integrate with so far. While it offers strong security properties, it also introduces significant complexity and UX challenges when used for interactive or threshold signing.

Advantages of NIP-46

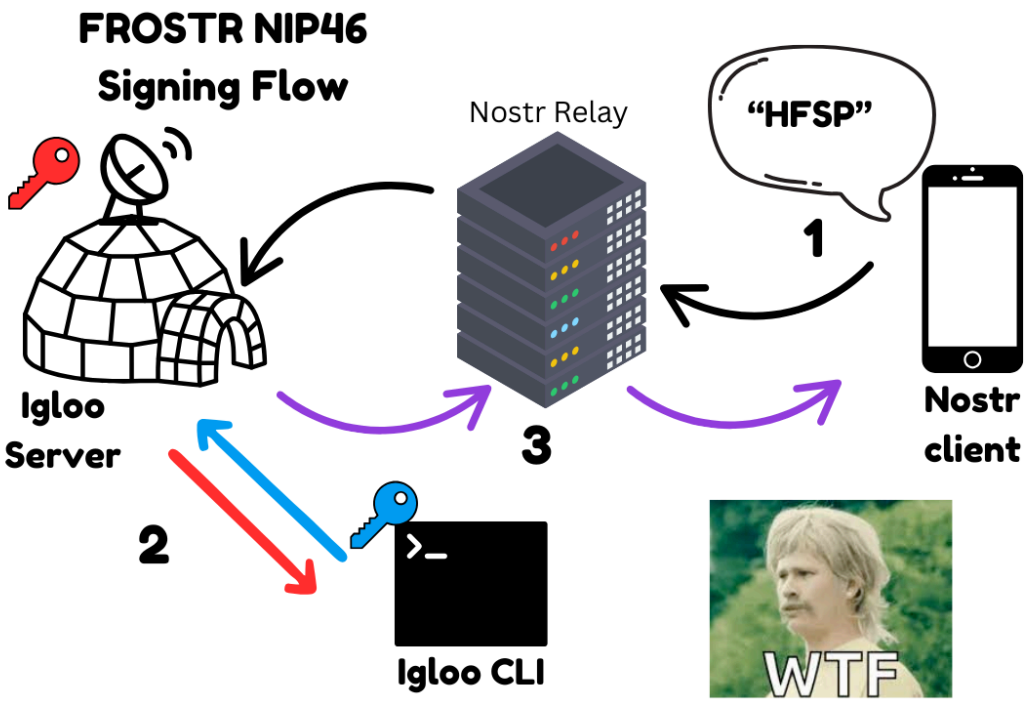
- **Strong transport-level privacy and security**
  Uses ephemeral transport keys so the signer's long-term key is never exposed to the client.
- **Encrypted request/response flow**
  All signing requests and responses are encrypted using **NIP-04** or **NIP-44**, protecting message contents from relays and observers.
- **Well-scoped remote signer model**
  Clean separation between client and signer, making it suitable for remote or headless signing environments.

Disadvantages of NIP-46

- **High message volume for a single signature**
  A single signing operation requires multiple request/response messages, which compounds quickly for threshold signing where multiple signers must coordinate.
- **Inconsistent real-world implementations**
  Clients vary significantly in how closely they follow the spec, leading to edge cases, assumptions, and incompatibilities that are not visible in the written standard.
- **Session lifecycle is fragile**
  Sessions can only be created, used, and destroyed—there is **no concept of session re-authentication or recovery**.
- **Poor session continuity UX**
  If a client loses its session (refresh, crash, device change), it must create a new one. Neither the client nor the signer has a reliable way to recognize that this is "the same client" returning.
- **Difficult to reason about for multi-signer setups**
  Session state, authorization, and timeouts become harder to coordinate when multiple Frostr signers must participate in a single signature flow.

Overall NIP-46 provides excellent security primitives, but its **session model and message complexity make it challenging for interactive, multi-party signing systems** like Frostr. Much of the integration difficulty comes not from cryptography, but from **state management, session fragility, and inconsistent client behavior**.

Below is I diagram I made for the NIP-46 signing flow that is done with Igloo Server today:



NIP-55:

Stuff

Learnings - integrating Frostr signers in different environments

Say some things right here and then…

Web:

Stuff

Server:

Server was pretty hard
Huge attack surface
Headless and DB mode (run igloo-server as a lightweight headless server surface OR in db mode run it with a full sqlite database and a full frontend)
Nip46

Umbrel packaging
Possibly start9 packaging soon

Desktop:

Mention desktop app and cli
Not too bad
Electron a bit finicky
The only frostr app (beside igloo-cli) that let's you create a keyset
Shares can be saved and encrypted to local files with pbkdf2

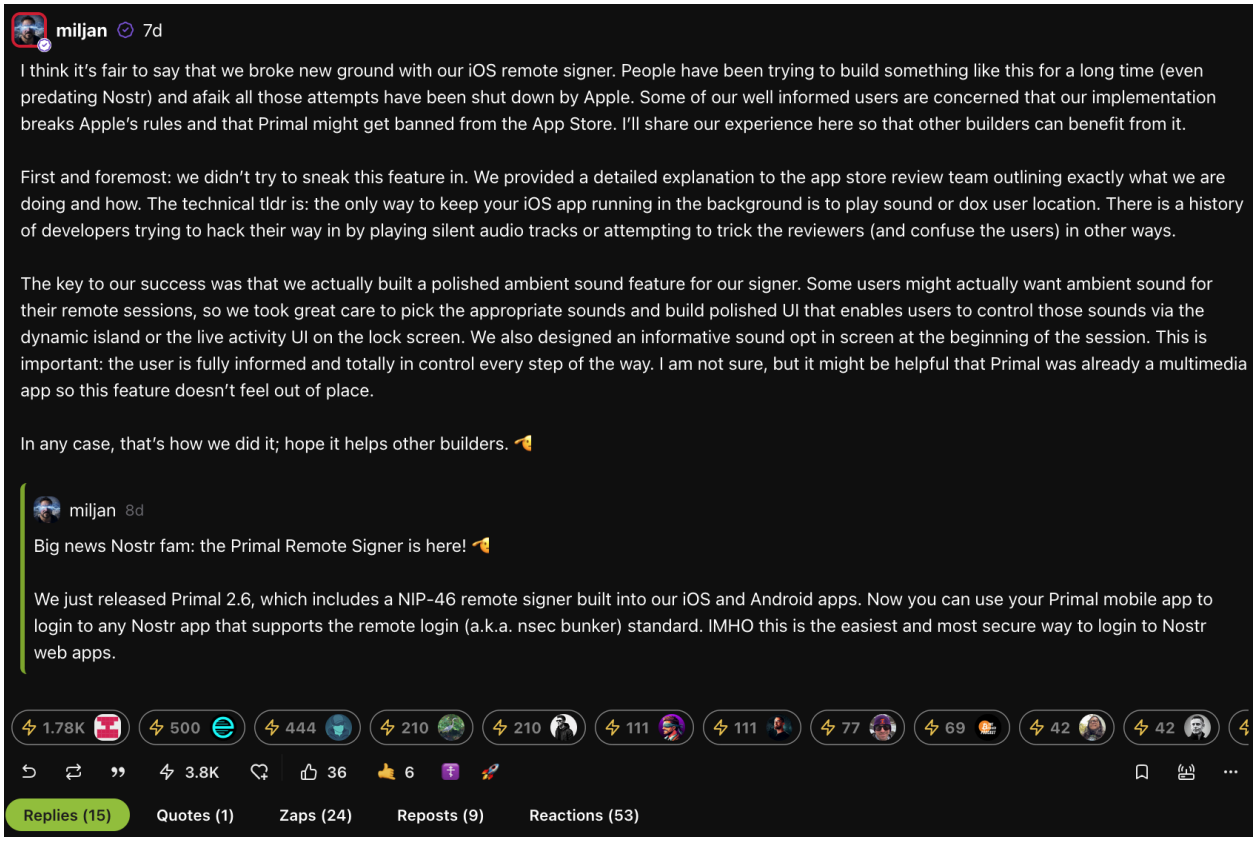
IOS:

iOS has consistently been the most challenging environment to design and ship a Frostr signer for; not just due to the technical difficulty of keeping a signer reliable, performant, and intermittently online on a mobile device, but also because all of this must be done without compromising user privacy. On top of these constraints, there is an additional, non technical question that must be considered from the start: whether Apple will allow the resulting design to be distributed at all under App Store policies.

Up until last week, the most promising approach we had was a prototype design that relied on Apple Push Notifications to wake an iOS-based Frostr signer when a signing request arrived. The intent was to use push notifications strictly as a wake-up signal, not as a transport for signing data.

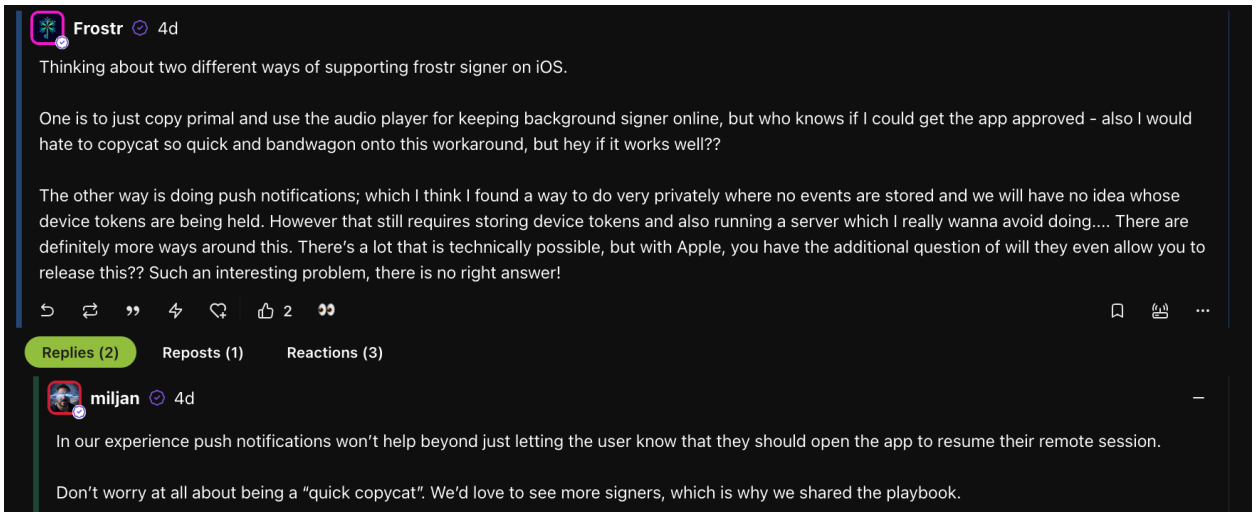
In this design, the push relay would store only two pieces of information: the signer's NIP-46 transport public key, used as a pseudonymous identifier, and the device's APNs device token. No user accounts, no Nostr pubkeys, no signing metadata, no message contents, and no additional correlatable fields would be stored. The relay would not know who the user is, what they are signing, or when signatures occur. All actual signing requests and responses would continue to flow end-to-end encrypted over Nostr, preserving Frostr's privacy and security guarantees.

HOWEVER everything changed when I saw this meme last week (and Miljan from Primal's announcement that this meme was replying to)

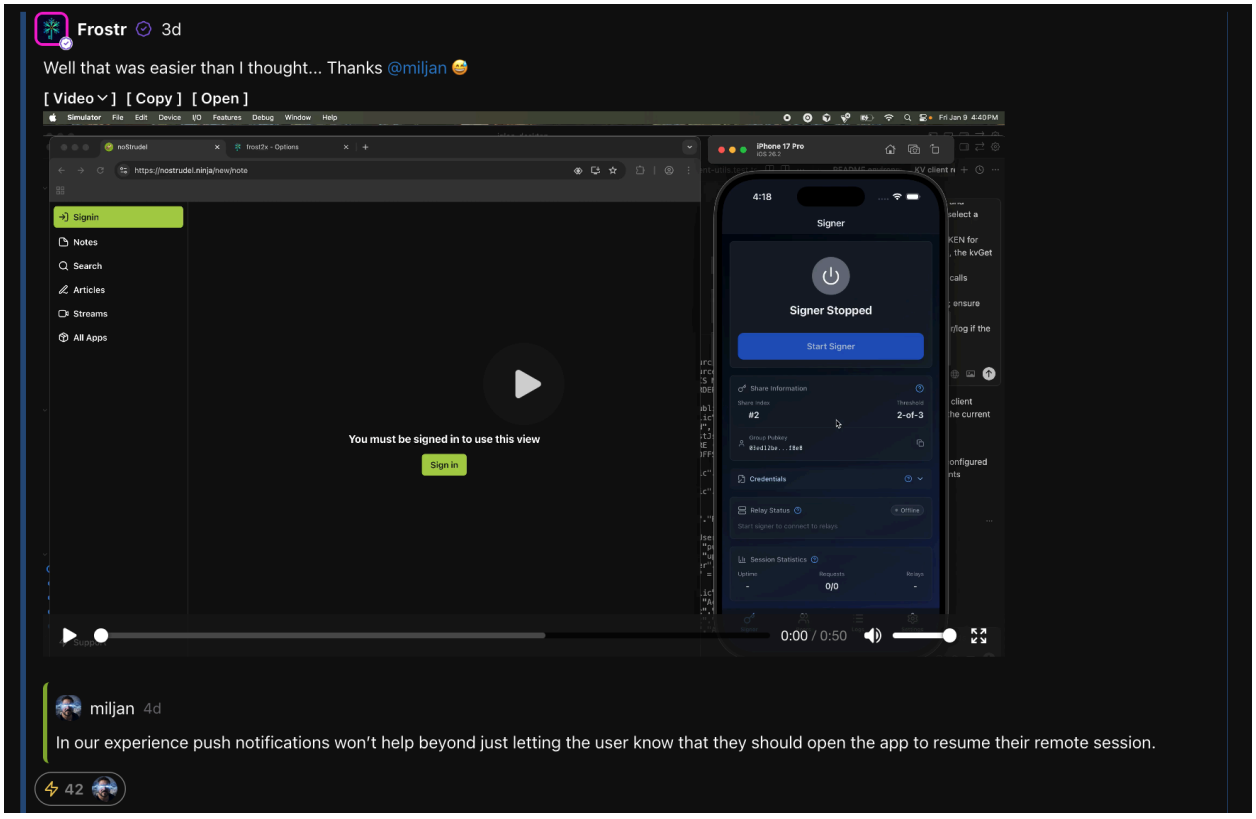


This was obviously an extremely timely development for my problem! Immediately my thoughts were “I could do this” but then it was “what if by doing this I bandwagon onto this hack and scare Apple away from approving this kind of background setup?”

So I posted this:



And the very next day…



Here is the demo video of this prototype:
https://plebdevs-bucket.nyc3.cdn.digitaloceanspaces.com/videos/frostr/igloo-ios-bg-signing-prototype.mp4

So, what does this mean? Maybe IOS in 2 weeks ™?

Android

Stuff

Next steps:

Our roadmap moving forward:

New Clients:

- Frostr IOS signer (if apple let's us release lol)
- Igloo Server Start9 packaging (will work just like igloo-server for umbrel)

Client Updates:

- Igloo Android usability improvements
- Improved & standardized onboarding for existing clients

New Libraries:

- frostr-rs NEW frostr core protocol library written in rust (built to replace bifrost and likely igloo-core)
- nostr-p2p (rust rewrite - will be bundled into frostr-rs but will be able to be used on it's own as well)