ONE NPUB — FAQ & Technical Reference

ONE NPUB — FAQ & Technical Reference

Version 2.1 Stable | March 2026 | Thomas+Agent21

> 📖 **Cross-reference:** Each answer references the relevant demo step. See **ONE_NPUB_DEMO_STEPS_GUIDE.pdf** for full UI + CLI instructions per step.

Introduction: What Problem Does This Solve?

The Nostr Identity Problem

Every Nostr user has a keypair: a **public key (npub)** that is their identity, and a **private key (nsec)** that signs everything they post. Your npub is your username, your reputation, your followers, your zaps — everything.

The nsec is a single 32-byte secret. It lives in a file, on a device, maybe copy-pasted into a browser extension. **If that device is stolen, hacked, or lost — your identity is gone.** Forever. No password reset. No customer support. Game over.

The ONE NPUB Solution

ONE NPUB splits the private key across multiple devices using **FROST threshold cryptography**. No single device holds the full secret. To sign an event, a minimum number of devices (the "threshold") must cooperate.

- Your npub stays the same — permanently
- Lose a device? Evict it and reshare. Old share becomes mathematically useless
- Someone steals your laptop? They have 1 share out of 7. They need 5. Worthless
- Want to rotate? New shares, same identity, old copies invalidated

The output is a **standard BIP-340 Schnorr signature**. No client, relay, or verifier can tell that threshold signing happened.

What We Built

This is **working software**, not a paper:

- **Real Docker containers** — each agent is a separate isolated process *(→ Step 2)*
- **Real FROST cryptography** — RFC 9591, audited curve library *(→ Steps 3–5)*
- **Real Nostr relay** — strfry (production C++ relay) *(→ Step 0)*
- **Real signatures** — every one verified with `schnorr.verify()` *(→*

Steps 3–18)
- **23-step interactive demo** — genesis to recovery *(→ Steps 0–22)*
- **Automated E2E test** — Playwright, 0 errors *(→ Step 22)*

How It Works (30-Second Version)

```
1. Coordinator creates a FROST key (1/1)              → Step 1
2. Start 4 agents → reshare to 5-of-7                 → Step 2
3. Coordinator asks 2 agents for partial signatures    → Step 3
4. Combines with its own 3 shares → threshold met     → Step 3
5. Output: standard Schnorr sig → Nostr relay          → Steps 3-5
6. npub never changes, even across reshares            → Step 14
```

Communication Architecture

```
Client → HTTP → Coordinator → WebSocket (NIP-01) → strfry Relay → WebSocket → Agents
```

Every FROST message is a signed NIP-01 event. The relay is pure transport. Each participant has its own transport keypair (separate from the group npub).

NIPs Used

NIP	Standard	Usage	Demo Steps
NIP-01	Basic Protocol	Event format, signatures, relay communication	All signing steps (3–5, 7–8, 10, 12, 15–18, 20)
NIP-03	OpenTimestamps	Anchor event existence in Bitcoin blockchain	Step 22 (verification)
NIP-19	bech32 Encoding	Human-readable `npub1...` addresses	Step 1 (genesis npub display)
NIP-85	Labeled Events (draft)	Epoch metadata and policy tags	Steps 3–5 (event tags)

Part 1: Easy Questions (Thomas Explains)

What is ONE NPUB in one sentence?

One Nostr identity, protected by multiple devices that must cooperate to sign — no single device can be hacked to steal your key.

> 📖 *See the full lifecycle: Steps 1–22*

Why not just keep my nsec in a password manager?

A password manager protects storage, not usage. The moment you paste your nsec into a client to sign, it's in memory. ONE NPUB means the full key is **never** in memory on any single device.

> 📖 *Step 1: Genesis creates the key. Step 2: Reshare splits it so no device holds the full key.*

Can I still use Primal, Damus, Amethyst?

Yes. Any Nostr client works. The signature output is identical to a normal nsec signature. Clients don't know (or care) that threshold signing happened.

> 📖 *Steps 3–5: Sign Kind 1, 7, 0 — all produce standard BIP-340 Schnorr signatures.*

How many devices do I need?

Minimum: 1 coordinator + 3 agents = 4 devices. An "agent" can be a phone, laptop, Raspberry Pi, or cloud VM. You can scale to 8 agents.

> 📖 *Step 2: Deploy 4 agents (5/7 model). Step 11: Add 5th agent (6/9 model).*

What happens if I lose one device?

Nothing breaks. The system keeps signing with remaining agents. Then: evict the lost device, reshare — old share becomes cryptographically useless.

> 📖 *Step 7: Agent D goes offline — signing still works. Step 13: Agent B evicted — old share dead.*

What if the coordinator goes down?

No signing happens. That's by design — the coordinator is your trust anchor. Keep it online and protected.

> 📖 *The coordinator is always participant #1 in every signing operation (Steps 3–18).*

Can I import my existing nsec?

Yes. Mathematically, an nsec is a degree-0 polynomial (a constant). Import it as 1/1 FROST, then reshare to your agents. Your npub stays the same.

> 📖 *Step 1: Genesis creates a 1/1 key — equivalent to an nsec import.*

What's the mnemonic backup?

Your emergency recovery phrase — 12 or 24 words. If everything is compromised, reconstruct from this. **Write it on paper. Store offline.**

> 📖 *Step 19: Emergency lockdown generates the mnemonic. Step 20: Recovery signs with it.*

Does this need a blockchain or tokens?

No. No blockchain, no tokens, no fees. It runs on standard Nostr relays with your own coordinator and agents.

> 📖 *Step 0: Prerequisites check — only Docker, Node.js, and a relay.*

What hardware do I need?

Each agent: ~50MB RAM Docker container. The full stack runs on a Raspberry Pi 4.

> 📖 *Step 2: Four agents start as lightweight Docker containers.*

How fast is signing?

~1-2 seconds per signature. Main bottleneck is relay round-trips, not cryptography.

> 📖 *Step 10: Stress test — 10 signatures in ~13 seconds, avg 1.3s each.*

Part 2: Hard Questions (Agent 21 Explains)

“The coordinator holds 3 of 7 shares — isn’t that a single point of failure?”

No. The threshold is 5. The coordinator has 3. 3 < 5. It cannot sign alone.

The math (k=2 model with 4 agents):

```
Coordinator shares:  a + 1 - k = 4 + 1 - 2 = 3
Agent shares:        a = 4 (one each)
Total shares:        n = 2a - 1 = 7
Threshold:           t = a + 1 = 5
```

An attacker needs the coordinator **plus** 2 agents. Compromising only the coordinator gives 3 shares — 2 short.

Why k=2 not k=1? With k=1, coordinator holds 4 shares. Attacker steals coordinator + bribes 1 agent = 5 shares = can sign. With k=2, they need coordinator + 2 agents.

> 📖 *Step 2: Attack table shows all compromise scenarios.*

“But the coordinator can reconstruct the key during signing, right?”

Yes. By design. The coordinator IS your device. The threat model protects against external attackers, not against yourself.

Scenario	Shares	Can sign?
1 agent stolen	1/7	✗ (need 5)
Coordinator stolen	3/7	✗ (need 5)
Coordinator + 1 agent	4/7	✗ (need 5)
Coordinator + 2 agents	5/7	✓ compromised

> 📖 *Step 2: Threshold model explained. Step 13: Eviction after compromise.*

“Prove the signatures are real.”

Every signature is BIP-340 Schnorr over secp256k1:

```
event_id = SHA-256([0, pubkey, created_at, kind, tags, content])
// NIP-01
sig = BIP-340_sign(event_id, group_secret)
// FROST output
verify(sig, event_id, npub) → true
// standard Schnorr
```

Our demo publishes real NIP-01 events to a real strfry relay. Connect any Nostr client to `ws://localhost:7777`. The events are there.

> 📖 *Steps 3–5, 8, 10, 12, 15–18, 20: Every signature shows verified=true with event ID and sig hex.*

“How does FROST actually work? Show me the math.”

Key Generation (DKG): *(→ Step 1)* 1. Random polynomial `f(x)` of degree `t-1`, where `f(0) = secret key s` 2. Each participant `i` gets `f(i)` — their secret share 3. Group public key: `P = s · G` 4. Nobody knows `s`

Signing — 2 rounds: *(→ Step 3)*

Round 1: Each participant generates nonce pair `(d_i, e_i)`, publishes commitments `(D_i = d_i·G, E_i = e_i·G)`

Round 2:

```
z_i = d_i + (e_i · ρ_i) + λ_i · s_i · c
```

- `s_i` = share, `c` = BIP-340 challenge, `λ_i` = Lagrange coefficient, `ρ_i` = binding factor

Aggregation: `z = Σ z_i`, `R = Σ(D_i + ρ_i·E_i)` → `(R, z)` = valid Schnorr signature

> 📖 *Step 1: DKG creates the key. Step 3: Full 2-round signing with crypto annotations.*

“What is Lagrange interpolation?”

Reconstructs a polynomial from a subset of points. Core math behind “t-of-n”:

```
f(0) = Σ f(i) · λ_i     where λ_i = Π_{j≠i} (0 - j) / (i - j)
```

Critical: Lagrange must use **actual participating indices**, not the full set. Using transport IDs instead of FROST share indices was our #1 bug across 22 test runs.

> 📖 *Step 7: Lagrange recomputed when Agent D is offline. Step 14: Recomputed with new indices after rotation.*

“Can old shares + new shares be combined after reshare?”

No. Reshare creates a **completely new polynomial** with the same `f(0)`:

```
f_old(x) = s + a₁x + ... + a₄x⁴    (old random coefficients)
f_new(x) = s + b₁x + ... + b₄x⁴    (fresh random coefficients)
```

Same constant term `s` = same npub. But `f_old(3)` tells you nothing about `f_new(3)`. Old keys don't open new locks.

> 📖 *Step 13: Agent B evicted — old share f_old(3) useless. Step 14: Proactive rotation — all old shares dead.*

"Is the relay a security risk?"

Transport only. Every FROST message is a signed NIP-01 event.

Attack	Possible?	Why
Forge messages	✖	Signature verification fails
Extract secret key	✖	Only commitments transit, not shares
Drop/delay messages	⚠ Yes	Denial of service only
Replay old messages	⚠ Yes	Epoch + nonce checks reject replays

> 📖 *Step 0: strfry relay is a prerequisite. All steps communicate through it.*

"How does BIP-85 backup recovery work?"

```
secret s → BIP-39 encode → 12/24 word mnemonic → BIP-39 decode →
secret s → s·G = P (npub)
```

Same secret → same public key → same npub. Always.

Demo note: Emergency lockdown creates a new DKG (new npub). Production would decode the mnemonic → import original secret → same npub.

> 📖 *Step 19: Mnemonic generated during lockdown. Step 20: Recovery signs in 1/1 mode.*

"How does OpenTimestamps (OTS) work?"

```
event_id (SHA-256) → OTS calendar → Merkle tree → Bitcoin OP_RETURN
→ proof chain
```

Proves your event existed **before** a Bitcoin block. Independently verifiable.

> 📖 *Step 22: Final verification shows epoch chain with OTS-tagged events.*

"Who initiates signing?"

The coordinator. Always. Agents are passive.

```
Client → POST /action/sign → Coordinator → Relay → Agents → Relay →
Coordinator → Signature
```

Caller	Method
Demo UI	Browser button → `fetch('/action/sign')`
CLI Demo	`bash cli-demo/step-03-sign-kind1.sh`
Manual	`curl -X POST http://localhost:3333/action/sign`
Production	Any HTTP client / app / webhook

> 📖 *Every signing step (3–5, 7–8, 10, 12, 15–18, 20) follows this pattern.*

"FROST vs Bitcoin multisig?"

	Bitcoin Multisig	ONE NPUB (FROST)
Output	Multiple signatures (m-of-n visible)	Single Schnorr signature
Privacy	Reveals threshold structure	Threshold completely hidden
Rotation	New address, new identity	Same pubkey, reshare only

> 📖 *Step 14: Reshare keeps same npub. Bitcoin multisig would need a new address.*

"FROST vs GG20 / CGGMP?"

Property	FROST (RFC 9591)	GG20 / CGGMP
Signing rounds	2	4-8
Output	Native Schnorr (BIP-340)	ECDSA
Nostr compatible	✅ Direct	❌ Needs adapter
Standardized	RFC 9591 (IETF)	Academic papers

FROST is the only rational choice for Nostr.

> 📖 *Step 10: Stress test shows 2-round signing latency (~1.3s avg).*

Honest Limitations

Security Gaps (Demo vs Production)

Gap	Severity	Demo Step
No API authentication	🟠 High	All signing steps — anyone with network access can call the API
No TLS between containers	🟡 Medium	Steps 3–18 — FROST messages plaintext in Docker network
Mnemonic = full key	🟠 High	Step 19 — whoever has the mnemonic has everything
No rate limiting	🟡 Medium	Step 10 — stress test shows no throttling
Shares not encrypted at rest	🟡 Medium	Step 2 — shares stored as JSON in Docker volumes
No external audit	🟡 Medium	All steps — `@noble/curves` audited, orchestration not

Architectural Limitations

Limitation	Impact	Visible in
Max ~8 agents	$O(n^2)$ reshare communication	Step 11 (5 agents)
Coordinator always required	No signing without it	All signing steps
All agents online for reshare	Signing ≠ reshare requirements	Step 7 vs Step 2
~1-2s per signature	Relay round-trip latency	Step 10 (stress test)
Lockdown creates new npub (demo)	Production uses BIP-85 import	Step 20

Theoretical Attack Surface

Attack	Feasibility	Demo Reference
Coordinator + 2 agents	Full compromise	Step 2 (attack table)
Mnemonic stolen	Full compromise	Step 19 (backup shown)
Quantum computing	secp256k1 broken	Affects all Nostr/Bitcoin
Rogue coordinator	Can reconstruct key	By design (your device)

Next Steps (Roadmap)

Short-Term (v2.2)

- ☐ API authentication (mutual TLS / API keys)
- ☐ Encrypted-at-rest shares
- ☐ BIP-85 recovery path (same npub after lockdown)
- ☐ Rate limiting
- ☐ Real OTS calendar integration

Medium-Term (v3.0)

- ☐ Mobile agent (iOS/Android app)
- ☐ Hardware agent (ESP32 air-gapped share holder)
- ☐ Human approval workflow for `requires_cosign` *(→ Step 5)*
- ☐ Multi-relay redundancy
- ☐ External security audit

Long-Term (v4.0+)

- ☐ TEE/Enclave (Intel SGX / ARM TrustZone)
- ☐ Coordinatorless mode (fully distributed MPC)
- ☐ Cross-protocol (Bitcoin Taproot threshold signing)
- ☐ Standardized NIP for threshold key state migration

Demo Quick Reference

Step	Title	What It Proves
0	Welcome	System prerequisites
1	Genesis	1/1 FROST key creation

2	Deploy A–D	Threshold reshare 5/7
3	Kind 1	Autonomous signing
4	Kind 7	Fresh nonces per sig
5	Kind 0	Policy: requires_cosign
6	Kind 4	Policy: forbidden
7	D offline	Threshold resilience
8	Kind 9735	Zap receipt signing
9	D returns	Policy determinism
10	Stress	10/10 burst throughput
11	Add E	Scale to 6/9
12	E signs	New agent integration
13	Evict B	Compromise response
14	Rotate	Proactive key hygiene
15–18	Sign ×4	Post-rotation continuity
19	Lockdown	Emergency revocation
20	Recovery	1/1 signing
21	New fleet	Rebuild 5/7
22	Verify	Epoch chain audit

> 📖 **Full details:** ONE_NPUB_DEMO_STEPS_GUIDE.pdf

Colophon

Built by **Thomas+Agent21** — a human-AI collaboration.

- **FROST:** RFC 9591, `@noble/curves/secp256k1` (audited by Trail of Bits)
- **Transport:** NIP-01 WebSocket via strfry relay
- **Stack:** Docker Compose, Node.js 22, TypeScript, pnpm monorepo
- **Testing:** 22 Playwright runs → 0 errors across 23 demo steps
- **License:** MIT

“One npub. Many devices. No single point of failure.”