

# OpenPGP Wallet: FROST threshold signing for shielded ZEC in the Ironwood pool

Khadim

October 2026 – version 0.2 (before independent audit)

**Abstract.** A non-custodial wallet that signs on a single device must, for a moment, hold the full spending key in memory, where malware can read it. We describe how OpenPGP Wallet will remove that single point of failure for shielded ZEC. The spend authorizing key of a new account in the Ironwood pool is split with FROST 2-of-3 between the user’s computer, the user’s phone and an encrypted offline backup, and is never reconstructed, not even at signing time. The computer builds the transaction as a PCZT and signs its share; the phone decodes and checks it independently, then co-signs over an end-to-end encrypted channel carried by QR codes, with no server and no third party. The design follows ZIP 312 and ZIP 374 and builds on the Zcash Foundation’s re-randomized FROST crates. Threshold signing already exists for technical users; our aim is to make it an ordinary feature of a consumer wallet, and to publish the signing messages and the share backup format as an open specification, so that FROST wallets for shielded ZEC can work together. Nothing described here is implemented yet, and nothing has been reviewed by an independent auditor.

## 1. The problem: one device, one full key

Today, OpenPGP Wallet signs on a single device, and that device briefly holds the full key. The wallet’s master entropy is encrypted with OpenPGP, unlocked by a password or by Windows Hello (a secret sealed in the TPM), and backed up as three Shamir shares. These measures protect the key at rest. They do not protect it while it is being used.

To sign, the wallet decrypts the key in memory. It is kept out of swap (`mlock`, `VirtualLock`) and wiped right after use, but for that short window it sits in normal RAM. Malware that can read process memory, or log the password as it is typed, can take it. The hardware chips cannot help: TPM 2.0, Apple’s Secure Enclave and Android Keystore do not sign with RedPallas, the signature scheme of the Orchard protocol.

A second problem follows from the first. The confirmation screen is drawn by the same machine that signs; if that machine is compromised, it can show one transaction and sign another. As long as one device ever holds the full spending key, one compromised device is enough to lose the funds.

## 2. Goals and non-goals

The goal is that spending shielded funds requires two independent devices to agree, with no third party involved. We aim for seven properties:

- 1) the spend authorizing key is never reconstructed on any device, at setup or at signing time;
- 2) one compromised device, computer or phone, is not enough to spend;
- 3) the phone shows what it signs from its own decoding of the transaction, never from a description sent by the computer;
- 4) no server: devices talk only through an end-to-end encrypted channel carried by QR codes, and every share belongs to the user;
- 5) losing one device does not lose the funds;
- 6) daily use stays simple: scan a QR code, check the amount and recipient on the phone, approve with biometrics;
- 7) the signing messages and the backup format are an open specification, so a co-signer or a backup made by one wallet works with another.

We do not claim privacy against a compromised computer, which keeps the viewing keys; protection when two devices are compromised at the same time; or hardware-wallet isolation, since a phone is not a secure element. Sapling, transparent and remaining Orchard funds must be moved to the new account to benefit from it.

The first networks targeted are shielded ZEC, Solana, NEAR, Stellar and Bitcoin (Taproot addresses): all use Schnorr-type signatures, the kind FROST produces. This work covers shielded ZEC; the same

architecture extends to the other networks, to Cardano and to Solana tokens (such as the Solana versions of USDC, USDT and EURC), and more generally to any chain whose signatures match a standard FROST ciphersuite, such as Ed25519 or BIP-340 Schnorr; those extensions will be funded separately. Ethereum and compatible networks, Tron, XRP and Dogecoin sign differently (ECDSA, as the wallet’s accounts on those networks do): they need another, heavier family of protocols, considered later. Monero, transparent ZEC and non-Taproot Bitcoin cannot use FROST either.

### 3. Background

**Orchard and Ironwood.** On 29 May 2026 a soundness vulnerability was disclosed in the Orchard circuit. NU6.3 activated on 28 July 2026 at block 3,428,143: it sealed the Orchard pool and opened Ironwood, a second value pool of the same Orchard protocol, with its own note commitment tree and quantum-recoverable notes (ZIP 2005) [6][11]. Orchard funds can only leave, through a turnstile. Everything below uses the Orchard protocol’s keys and signatures, applied to the Ironwood pool.

**Orchard keys.** An Orchard spending key yields three components: the spend authorizing key  $ask$ , the nullifier deriving key  $nk$  and the commit randomness  $rivk$ . The full viewing key is  $(ak, nk, rivk)$ , where  $ak$  is the public key matching  $ask$ . Seeing funds requires  $nk$  and  $rivk$ ; spending also requires a signature from  $ask$ . Each spend carries a RedPallas signature over the transaction’s sighash (ZIP 244), made under a randomized key

$$rk = ak + [\alpha] G, \tag{1}$$

where  $\alpha$  is a fresh randomizer per spend [7]. This keeps spends unlinkable on chain.

**FROST.** FROST [1][2] is a two-round threshold Schnorr signature protocol. In round 1, each signer commits to fresh nonces; in round 2, each signer produces a signature share over a signing package that contains the message and all commitments. Shares aggregate into one ordinary signature, and the group key is never reconstructed.

**Re-randomized FROST and ZIP 312.** Orchard signatures are made under  $rk$ , not  $ak$ . Re-randomized FROST [3] lets signers produce a signature valid under the group key shifted by a randomizer; the Zcash Foundation implements it in `frost-rerandomized`, with a RedPallas ciphersuite in `reddsa` [8]. ZIP 312 (Draft) [4] specifies its use for Sapling and Orchard: only  $ask$  is split, and  $ak$  becomes the FROST group public key. Key generation is out of scope; the ZIP notes that an  $ask$  created by DKG cannot be recovered from a seed phrase. Its threat model trusts the coordinator with privacy but not with spending: a rogue coordinator can break unlinkability, but cannot sign without the threshold of signers.

**PCZT.** ZIP 374 (Draft) [5] defines the partially created Zcash transaction format, with roles such as Constructor, Prover, Signer and Combiner. Its version 2 adds an Ironwood bundle and v6 transactions, in which signatures cover neither the anchors nor the proofs, so a transaction can be signed before it is proved.

**Existing work.** FROST for Zcash is already used: the Zcash Foundation publishes demo tools such as `zcash-sign`, the Zkool wallet supports shielded multisignature accounts with DKG and FROST [10], several community projects build shared-custody layers, and in October 2026 NoirWallet published a design for a similar 2-of-3 setup [13]. Our contribution is not the protocol but the product: a consumer wallet where the second signer is the user’s own phone, paired by QR code, with no server and an independent transaction check; and an open specification, so that these wallets can work together (section 10).

### 4. Architecture

Three participants hold one FROST share each (figure 1). The computer and the phone sign every day; the backup only replaces a lost device.

The FROST account is a new account in the Ironwood pool, with its own unified address. Its  $ask$  is created by DKG and is not derived from the wallet’s master entropy: if it were, anyone who stole that entropy would bypass FROST entirely. Users move their ZEC to the new address once, from inside the wallet; funds still in the sealed Orchard pool reach it through the turnstile.

The desktop app runs on Windows today. This work brings the coordinator role to macOS and Linux as well, so that every desktop user can pair a phone, and adds signer apps for Android, then iOS.

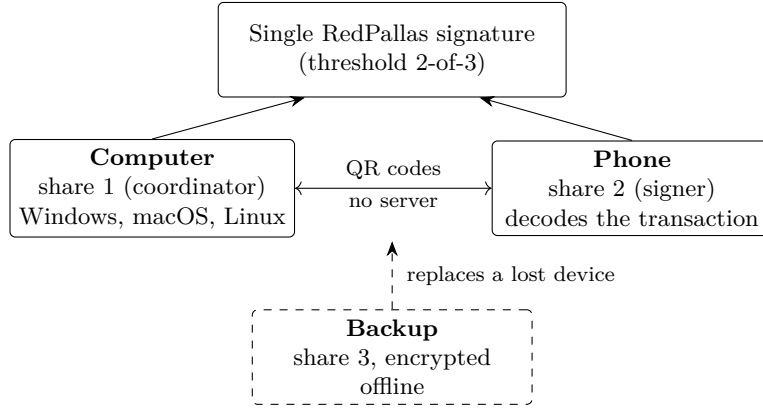


FIG. 1: The daily signers are the user’s two devices. The backup share only replaces a lost device.

| Participant | Holds                     | Role                                                                               | Protection at rest                                                                                                                    |
|-------------|---------------------------|------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Computer    | Share 1, full viewing key | Coordinator: builds the PCZT, aggregates the signature, creates proofs, broadcasts | OpenPGP vault; Windows Hello / TPM on Windows, Keychain / Secure Enclave on macOS, Secret Service (and TPM 2.0 when present) on Linux |
| Phone       | Share 2, full viewing key | Signer: decodes and checks the transaction, co-signs                               | Wrapped by a key in the Secure Enclave or Android Keystore, unlocked by biometrics                                                    |
| Backup      | Share 3                   | Recovery only; never signs daily                                                   | Encrypted under its own OpenPGP identity and passphrase, offline on removable media                                                   |

TABLE 1: Key material and protection per participant.

## 5. Key generation and pairing

Shares are created by distributed key generation (DKG), so no device ever holds the full ask, not even during setup.

- 1) **Pairing.** The computer shows a QR code with an ephemeral public key; the phone scans it and answers with its own. Both derive a shared channel key and display a short code that the user compares. Every later message, at setup and at signing, travels over this end-to-end encrypted and authenticated channel, carried by QR codes.
- 2) **DKG.** The devices run the two-round DKG of **frost-core** over the paired channel, with threshold 2 and three participants. Round 1 packages are public; round 2 packages are encrypted to each recipient.
- 3) **Backup participant.** The backup is not a device, so its DKG role runs on the phone. As soon as share 3 is computed, it is encrypted to the backup OpenPGP identity, exported to removable media through the computer, and wiped from the phone.
- 4) **Viewing components.** The computer generates the viewing components ( $nk$ ,  $rivk$ ) and sends them to the phone over the channel, so both devices hold the full viewing key. No standard yet defines this step for an Ironwood account where nobody holds a spending key; section 12 explains how we handle it.
- 5) **Address check.** Each device derives the unified address independently and displays it; the user confirms they match before any funds are sent.

Step 3 has a known weakness: for a short time at setup, the phone holds two shares, which is the threshold. We bound this window and offer a way to remove it (section 12, decision 2). The whole setup takes a few QR scans and happens once per account.

## 6. Signing flow

Signing takes three QR codes between the phone and the computer, with no server in between (figure 2). The phone commits first; the computer builds the transaction around randomizers derived from both devices' commitments and signs its share; the phone checks the transaction and the computer's share, then co-signs; the computer finishes.

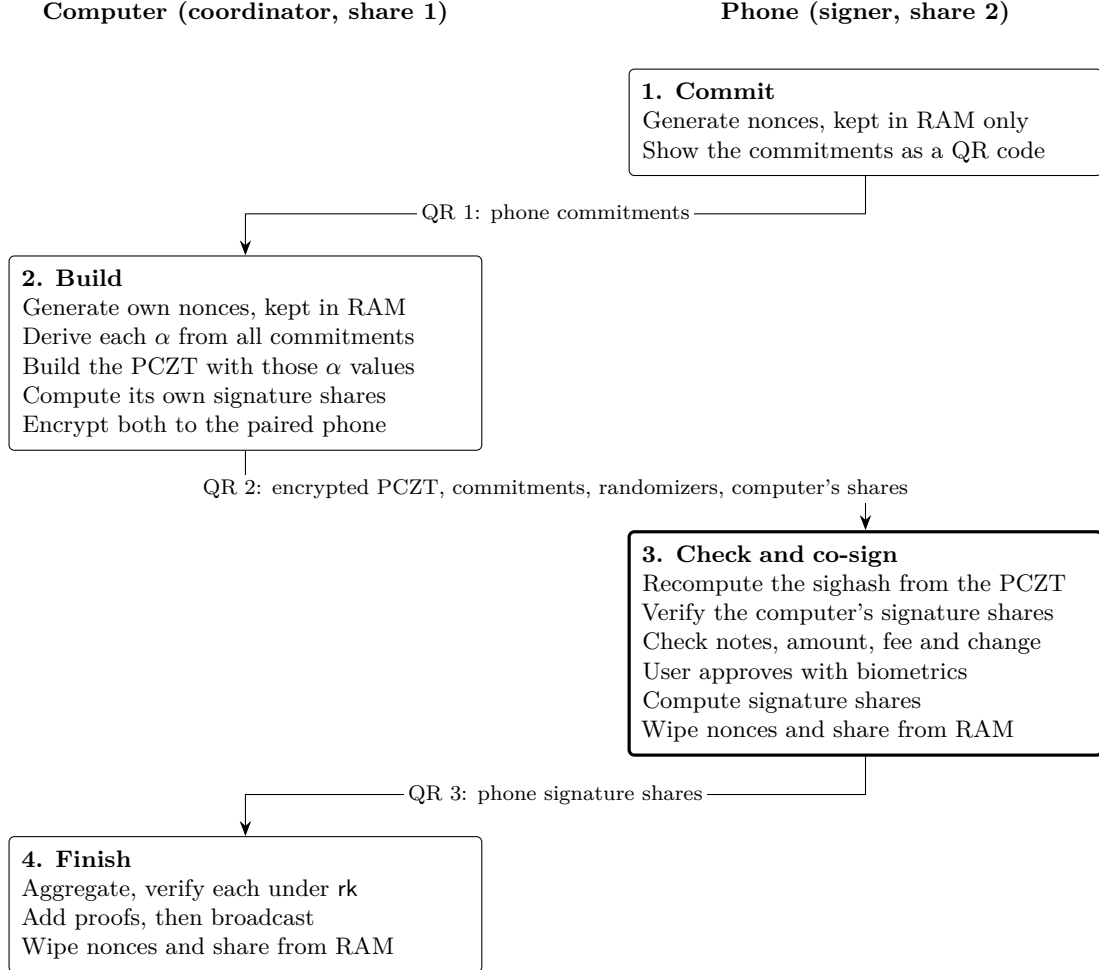


FIG. 2: Signing a transaction: two devices, three QR codes. Nonces never leave RAM.

**Why the phone commits first.** Each spend's  $rk$  is part of the sighash, so  $\alpha$  must be fixed before the transaction is built. ZIP 312 derives the randomizer from the signers' commitments, so the commitments must exist first; this is the order proposed in `zcash/orchard#571` [9].

**Library support.** `frost-rerandomized` 3.0 derives the randomizer from a seed and the commitments, but the Orchard builder does not yet accept a caller-supplied  $\alpha$  [9]. Until it does, we follow `zcash-sign`: the builder picks  $\alpha$  and signers use the external-randomizer path, which the crate's maintainer has confirmed will remain available [8].

**Confidential channel.** ZIP 312 requires the randomizer to travel over a confidential and authenticated channel, because it links  $rk$  to  $ak$ . QR payloads are therefore encrypted with the key set up at pairing, even though they never leave the room.

**Nonces.** Nonces are single-use and live only in RAM, so the phone app stays open across the three codes; an aborted session discards them on both devices. Each spend needs its own signature, and all spends of one transaction travel in the same QR codes, which are animated multi-part codes when a PCZT exceeds one code's capacity. In v6 transactions, signatures cover neither the anchor nor the proofs, so the computer can prove before or after signing.

## 7. What the phone verifies before signing

The phone signs only what it has decoded and checked itself; it never trusts a summary from the computer. Because it holds the full viewing key, it can check every part of the transaction that matters to the user:

- 1) **Sighash.** It recomputes the ZIP 244 sighash from the PCZT and signs that value only.
- 2) **Computer's share.** It verifies the computer's signature shares against the computer's public verifying share, so "your computer has already signed its share" is checked, not assumed.
- 3) **Spends.** For each spent note, it checks that the nullifier matches the note data and the full viewing key, so the computer cannot spend notes the user did not choose.
- 4) **Outputs.** For each output, it checks that the note commitment matches the recipient, value and randomness carried in the PCZT, and that the displayed address contains that recipient, as ZIP 374 requires of signers.
- 5) **Values.** It checks values against their value commitments and recomputes the fee from the value balance, flagging anything above the ZIP 317 fee.
- 6) **Change.** It recognizes outputs to the user's own address as change, and shows only external recipients as payments.
- 7) **Dummies.** It rejects any PCZT that still carries dummy spending keys, as ZIP 374 requires.

The screen then shows each recipient's full address, the amount, the fee and the memo, built from these checks. ZIP 374 lets a constructor redact values for signers that do not need them; our phone always needs them, so it refuses any PCZT where a field it relies on is missing. The first version supports plain transfers from the Ironwood pool to Ironwood, unified or transparent recipients, and only spends notes already mined.

## 8. Storing the shares

Each share is encrypted at rest with a key the device's security chip controls, but it is in normal memory while it signs. Secure Enclave and Android Keystore cannot hold a RedPallas key or run FROST; they can hold a P-256 key that never leaves the chip. We use such a key, gated by biometrics, to wrap the share: the encrypted share is stored on the device, and the chip must approve each unwrap.

- **Phone.** The share is unwrapped only after a fresh biometric check, used for one signing session, then wiped. Its memory is locked against swap where the platform allows.
- **Computer.** The share sits in the existing OpenPGP vault, unlocked by password, or by Windows Hello on Windows and Touch ID on macOS; Linux has no standard biometric unlock, so it uses the password. Memory is locked and wiped after use, as today.
- **Backup.** Share 3 is encrypted under its own OpenPGP identity and passphrase and stored offline. It never touches a device except during recovery.

This protects a stolen or powered-off device. It does not protect a running, compromised device during a signing session; that is what the second device is for.

## 9. Recovery and device replacement

If one device is lost, the backup share and the surviving device sign together to move the funds to a fresh FROST account: the surviving device and the replacement run a new DKG, which also creates a new backup share; the surviving device, with the old backup share as second signer, signs one transaction sweeping all funds to the new address; the old backup share is then retired. We sweep instead of repairing the lost share: **frost-core** offers share repair and refresh, but repairing alone would leave the lost device's share valid, so whoever finds that device would still hold one of the two shares needed to spend. If two of the three shares are lost, the funds cannot be recovered, as with the wallet's existing Shamir backup.

## 10. Interoperability: an open specification

Several wallets now build FROST for shielded ZEC. Zkool ships it, NoirWallet has published a design close to ours [13], and each one defines its own pairing, signing messages and share backups. A co-signer from

one wallet then cannot co-sign for another, and a backup restores only in the app that made it; Zkool’s author has noted that no standard exists yet for FROST account backups. Fragmentation also has a security cost: when every signer of an account comes from the same publisher, one compromised release pipeline or store account can push a malicious update to two signers at once, which reaches the threshold. Users who can pair devices from different vendors close that gap.

We therefore publish the parts that wallets can share as an open, versioned specification with test vectors, implemented in a standalone Rust crate that has no dependency on the rest of OpenPGP Wallet:

- **QR signing protocol:** pairing, message encryption and framing, animated multi-part codes, and the three messages of a signing session (section 6);
- **Signer checks:** the verifications of section 7;
- **Share backup format:** the share, the group’s public data, the viewing components, encryption and versioning, with two profiles: an encrypted file with no recovery phrase, as in section 8, and a mnemonic profile for wallets that prefer phrases.

The draft is discussed in public with the other wallet teams and the Zcash Foundation’s FROST team, and offered as a draft ZIP if they see value in it.

## 11. Threat model

An attacker needs two of the three shares to spend; one compromised device lets them watch the funds, not take them (table 2).

| Attacker controls                             | Spend? | Sees funds?       | Why                                                     |
|-----------------------------------------------|--------|-------------------|---------------------------------------------------------|
| Running computer (malware)                    | No     | Yes               | One share; the phone shows what it actually signs       |
| Running phone (malware)                       | No     | Yes               | One share; it never builds transactions                 |
| Stolen device, off or locked                  | No     | Depends on unlock | Share wrapped by the security chip or the OpenPGP vault |
| Backup media                                  | No     | No                | One share, encrypted under its own passphrase           |
| Computer and phone at the same time           | Yes    | Yes               | Two shares reach the threshold                          |
| One device plus the backup and its passphrase | Yes    | Yes               | Two shares reach the threshold                          |
| Phone, at the moment of setup                 | Yes    | Yes               | It briefly holds the backup share (section 5)           |

TABLE 2: What an attacker can do depending on what they control.

Two human factors remain. A user who approves on the phone without reading the screen gives up most of the protection; a user who keeps the backup passphrase next to the backup media turns it into an unprotected share. This design is not hardware-wallet isolation, it does not hide the user’s history from a compromised computer, and it has not been audited.

## 12. Research findings and design decisions

Every design question is settled: table 3 records what the specifications and issue trackers establish, and three decisions close the remaining choices.

- 1) **Viewing key: real tests now, public mainnet with the standard.** Until `zcash/zips#895` and ZIP 2005 define the derivation, test accounts use the one `zcash-sign` uses today, which gives up quantum recoverability. We test on testnet and on mainnet with small amounts, our own and our beta testers’, and sweep those funds to a standard account once it exists. The public mainnet release creates accounts only with the standard derivation, so no user keeps funds in an account without quantum recoverability. If the standard lands late, testing continues and the public release follows the standard.
- 2) **Backup share: one bounded window, or a spare device.** By default, the phone runs the backup role during setup, encrypts share 3 at once and wipes it; the window lasts one setup and the shares exist only in RAM. Users who want no window at all run the backup role on a spare phone or tablet instead.

| Question                                | What we found                                                                                                                                                                | Effect on the design                                                                              |
|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| Which pool?                             | Orchard was sealed by NU6.3 on 28 July 2026; new shielded funds go to Ironwood                                                                                               | The FROST account lives in the Ironwood pool                                                      |
| How is the randomizer generated?        | ZIP 312 derives it from fresh randomness and the signers’ commitments; rk is in the sighash, so this must happen before the transaction is built [9]                         | The phone commits first: three QR codes instead of two                                            |
| Is the randomizer API stable?           | The seed-based API cannot yet take a transaction’s $\alpha$ ; the maintainer will keep an external-randomizer path [8]                                                       | We use that path, as <b>zcash-sign</b> does, and track the API                                    |
| How are nk and rivk created?            | No standard yet; <b>zcash-sign</b> uses a fork-only constructor that gives up Ironwood’s quantum recoverability; the proper path waits on <b>zcash/zips#895</b> and ZIP 2005 | Real tests use small amounts; the public mainnet release waits for the standard path (decision 1) |
| Does a PCZT carry what the phone needs? | Yes: ZIP 374 carries recipient, value and randomness per output, and defines the signer’s checks; values may be redacted                                                     | The phone follows those checks and refuses redacted PCZTs                                         |
| Is this new?                            | Zkool and Zcash Foundation tools already do FROST shielded multisig [10]; NoirWallet published a similar design [13]                                                         | The work is a consumer product and an open specification, not a new protocol                      |

TABLE 3: Research findings and their effect on the design.

- 3) **Nonces: never stored.** Pre-generated commitments would save one scan, but stored nonces could be replayed after a backup restore and leak a share. One extra scan costs less than that risk.

### 13. Implementation plan and status

OpenPGP Wallet v2.13.2 runs on Windows, with 387 core tests and 27 application tests, continuous integration and no known vulnerability reported by **cargo audit**. Its code is public under the MIT license [14]. Since v2.13.2 its Zcash module counts and spends shielded funds in the Ironwood pool; shielded spending has not yet been tested with real funds. Nothing in this document is implemented yet. The work is planned in six phases; phases 2 and 3 are the subject of a grant request to Zcash Community Grants, and the later phases will be funded separately:

- 1) **Groundwork** (October 2026, nearly done). Code published under MIT, Zcash module on v6 transactions and the Ironwood pool. Remaining: a real shielded spend on the single-key account.
- 2) **FROST core and draft specification** (six weeks after funding). DKG, re-randomized signing, PCZT coordinator role, QR transport and a desktop test co-signer, checked against the crates’ test vectors, with 2-of-3 spends on testnet and on mainnet with small amounts; specification draft v0.1 published for review.
- 3) **Crate, backups and recovery** (eleven weeks after funding). The standalone crate, the backup format with its two profiles, the recovery sweep, specification v1.0 after review, and an audit scope.
- 4) **Android co-signer**. Pairing, transaction decoding and checks, Android Keystore wrapping and the approval screen, built on the same crate.
- 5) **iOS co-signer, macOS and Linux coordinator**. Secure Enclave wrapping; share storage in the macOS Keychain and in the Linux Secret Service, with TPM 2.0 when present; QR scanning by webcam; notarized and signed builds.
- 6) **Independent audit and public release**. Security review of the FROST, protocol and co-signer code by an outside firm, fixes, then the public mainnet release once the viewing-key standard is final.

A Windows VBS enclave for the computer’s share is an optional module after these phases. Each phase will be reported publicly on the Zcash forum [15].

## 14. Conclusion

We have described how a consumer wallet can keep shielded ZEC behind two independent devices: a FROST 2-of-3 key that is never reconstructed, a phone that checks every transaction itself, QR codes instead of a server, and an open specification so that FROST wallets for shielded ZEC can work together. The design builds on existing Zcash standards rather than inventing new cryptography, states its limits openly, and settles each open question with an explicit decision. The code and its verification are meant to be exposed to public review, and to an independent audit, before any use with real funds.

## References

- [1] D. Connolly, C. Komlo, I. Goldberg, C. Wood, “The Flexible Round-Optimized Schnorr Threshold (FROST) Protocol for Two-Round Schnorr Signatures”, RFC 9591, IETF, 2024.
- [2] C. Komlo, I. Goldberg, “FROST: Flexible Round-Optimized Schnorr Threshold Signatures”, SAC 2020; IACR ePrint 2020/852.
- [3] C. Gouvêa, C. Komlo, “Re-Randomized FROST”, IACR ePrint 2024/436: <https://eprint.iacr.org/2024/436>
- [4] ZIP 312, “FROST for Spend Authorization Multisignatures” (Draft): <https://zips.z.cash/zip-0312>
- [5] ZIP 374, “Partially Created Zcash Transaction Format” (Draft): <https://zips.z.cash/zip-0374>
- [6] ZIP 244, “Transaction Identifier Non-Malleability”; ZIP 317, “Proportional Transfer Fee Mechanism”; ZIP 2005, “Ironwood Quantum Recoverability”: <https://zips.z.cash/>
- [7] Zcash Protocol Specification, Orchard key components and spend authorization signatures.
- [8] Zcash Foundation, FROST crates: <https://github.com/ZcashFoundation/frost>; issue #1094 on external randomizers and Ironwood viewing keys.
- [9] [zcash/orchard#571](https://github.com/zcash/orchard/issues/571), caller-supplied spend randomizers for FROST: <https://github.com/zcash/orchard/issues/571>
- [10] Zkool, a Zcash wallet with FROST shielded multisignature accounts: <https://github.com/hhanh00/zkool2>
- [11] “Zcash seals \$1.7 billion shielded pool as Ironwood upgrade activates”, CoinDesk, 28 July 2026.
- [12] OpenPGP Wallet project site: <https://khadevs.github.io/openpgp-wallet-site/>
- [13] NoirWallet, “FROST Multi-Device Signing Design”, Zcash Community Forum, 13 October 2026: <https://forum.zcashcommunity.com/t/noirwallet-frost-multi-device-signing-design/58170>
- [14] OpenPGP Wallet source code (MIT): <https://github.com/OpenPGP-Wallet/openpgp-wallet>
- [15] Design review thread, Zcash Community Forum: <https://forum.zcashcommunity.com/t/frost-2-of-3-between-a-users-own-devices-for-shielded-zec-ironwood/58164>