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

Khadim

October 2026 – version 0.4 (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, ZIP 374 and ZIP 2005 and builds on the Zcash Foundation's re-randomized FROST crates. Threshold signing for shielded ZEC already exists in Zkool and in the Zcash Foundation's tools; 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. A standalone prototype already makes 2-of-3 spends on mainnet (section 13). The wallet itself does not use FROST 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; hardware-wallet isolation, since a phone is not a secure element; or 2-of-3 protection against a future quantum computer, since every member keeps the group's sk (section 5), a limit ZIP 2005 states. Sapling, transparent and remaining Orchard funds must be moved to the new account to benefit from it.

This work covers shielded ZEC only. The same architecture could later serve other networks whose signatures match a standard FROST ciphersuite, such as Ed25519 or BIP-340 Schnorr; that would be separate work, funded separately. Networks that sign with ECDSA, transparent ZEC included, need another, heavier family of protocols.

## 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 $(\mathsf{ak}, \mathsf{nk}, \mathsf{rivk})$, 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

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

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 for now (zcash/zips#895 proposes to bring it in); 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.

**Quantum recoverability for shared keys.** ZIP 2005 [6] lets the owner of Ironwood notes recover them after a break of the discrete logarithm, as long as a secret qsk is kept. For keys whose ask is shared, it defines a derivation (`use_qsk`) in which the group agrees on a secret sk and derives nk, qsk and rivk from it, so ak can come from a DKG while the notes stay recoverable.

**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`, whose Ironwood support has been used for 2-of-3 FROST spends on testnet [17]; the Zkool wallet supports shielded multisignature accounts with DKG and FROST [10]; Zafe builds a shared vault [18] and Konclave a group treasury on FROST; 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. Later phases bring the coordinator role to macOS and Linux as well, so that every desktop user can pair a phone, and add signer apps for Android, then iOS.

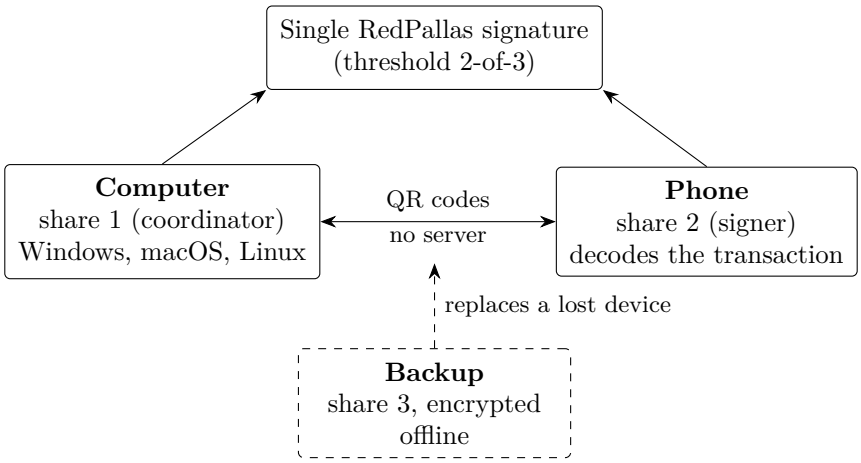



FIG. 1: The daily signers are the user's two devices. The backup share only replaces a lost device.

| Participant | Holds | Role | Protection at rest |
|---|---|---|---|
| Computer | Share 1, sk, full viewing key | Coordinator: builds the PCZT, aggregates the signature, creates proofs, broadcasts | Symmetric key from the password (Argon2id), or sealed in the TPM (Windows Hello), the Keychain (Touch ID) or the Secret Service on Linux |
| Phone | Share 2, sk, full viewing key | Signer: decodes and checks the transaction, co-signs | AES key in Android Keystore, or iOS Keychain item, unlocked by biometrics |
| Backup | Share 3, sk | Recovery only; never signs daily | Encrypted with its own passphrase only (Argon2, AEAD), 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 with the backup passphrase, with no public-key layer (section 8), exported to removable media through the computer, and wiped from the phone.

4) **Viewing components.** The participants agree on a secret sk, each contributing randomness, and derive nk, qsk and rivk from it with the ZIP 2005 derivation for shared keys, so both devices hold the full viewing key and the notes stay quantum recoverable. How sk is agreed during key generation is being standardized in zcash/zips#895; we will follow 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.

**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.

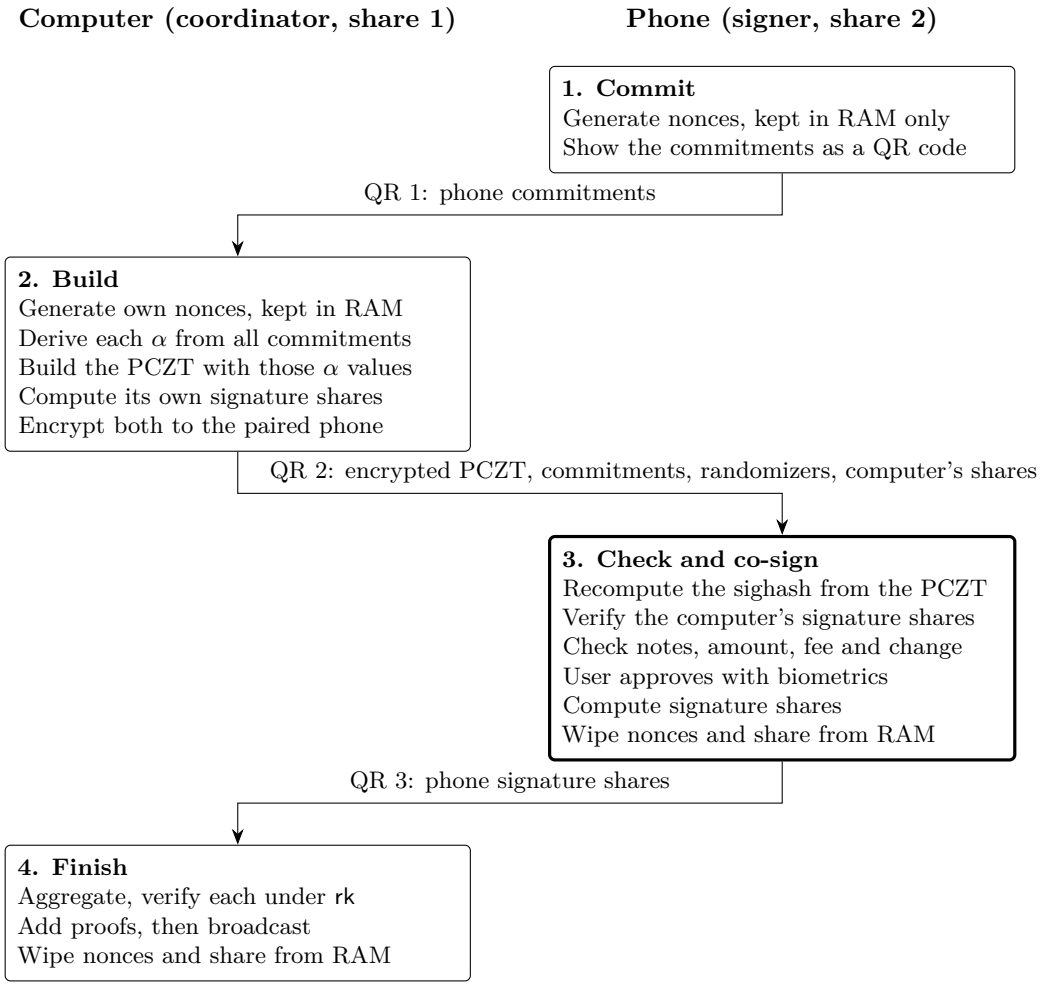



FIG. 2: Signing a transaction: two devices, three QR codes. Nonces never leave RAM.

## 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. The prototype of section 13 already runs most of these checks, with the PCZT verifiers of the `orchard` crate.

## 8. Storing the shares

Each participant keeps its share and `sk`, from which `qsk` is derived, encrypted at rest, and only symmetric encryption protects them: an AEAD under a key held by the platform's secure hardware or derived from a password with Argon2id. No layer relies only on elliptic-curve encryption. A ciphertext recorded today would otherwise become readable to an adversary who later breaks the discrete logarithm, and ZIP 2005 notes that `qsk` alone may let such an adversary steal the funds; the backup ZIP (section 10) asks wallets not to store or send backups that way. Secure Enclave and Android Keystore cannot hold a RedPallas key or run FROST, so the secrets are in normal memory while a device signs.

- **Phone.** On Android, an AES-256 key generated in Android Keystore (StrongBox when present) can be used only after a fresh biometric check. The Secure Enclave of iOS holds P-256 keys only, so the share is not wrapped with one: it is stored in the Keychain, whose items are encrypted with AES keys tied to the device and its passcode, behind a biometric access control. The secrets are used for one signing session, then wiped, and their memory is locked against swap where the platform allows.
- **Computer.** The share and `sk` do not go into the existing OpenPGP vault, which encrypts to a Curve25519 key. They are encrypted under a key derived from the password with Argon2id, or under a random key sealed in the TPM with Windows Hello, or kept in the Keychain with Touch ID on macOS; Linux has no standard biometric unlock, so it uses the password, or the Secret Service. Memory is locked and wiped after use, as today.
- **Backup.** Share 3 and `sk` are encrypted with the backup's own passphrase only: an OpenPGP message with no public-key layer (RFC 9580, with Argon2 and AEAD), which the Sequoia library the wallet already uses supports. It is stored offline and never touches a device except during recovery.

This protects a stolen or powered-off device, and a ciphertext copied today against a later break of the discrete logarithm. 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, Zafe and Konclave build on 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 share backup format is being written as a ZIP, co-owned with aryaethn, whose version 1 is the `zkfrost` string that Zkool already ships, so that existing Zkool backups stay valid. The draft and its test vectors are public [19]: Zkool and `frost-zec` both pass the vectors, and a third decoder written by aryaethn from the specifications alone, with no Zcash library, matches them too. We will take it through the ZIP Editors' review. The QR signing protocol stays a versioned specification for now, and can become a second ZIP once a second implementation uses it. An interoperability test with Zafe is being prepared: one vault shared between a member running Zafe and a member running our prototype.

## 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 and `sk` under symmetric keys: security chip, TPM or password |
| 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

Table 3 records what the specifications and issue trackers establish, and three decisions close the remaining choices for the first version. The backup format's open points are listed in its draft [19].

1) **Viewing key: the ZIP 2005 derivation.** FROST accounts derive nk, qsk and rivk from a shared sk with the ZIP 2005 derivation for shared keys, so their notes stay quantum recoverable. Our prototype implements it, and it matches the independent test vectors proposed by the Zafe team [20]; there are no official test vectors yet. Mainnet tests stay small until official vectors exist and the code has been audited.

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.

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.

| 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 `zcash-sign` does, and track the API |
| How are nk and rivk created? | ZIP 2005 derives them, with qsk, from a secret sk the group agrees on; `zcash-sign` uses a constructor that gives up quantum recoverability; zcash/zips#895 will specify how sk is agreed | We use the ZIP 2005 derivation (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, Zafe, Konclave and Zcash Foundation tools already do FROST for shielded ZEC [10][18]; 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.

## 13. Implementation plan and status

OpenPGP Wallet v2.15.0 runs on Windows, with automated tests and continuous integration (formatting, lints, tests and `cargo audit`). Its code is public under the MIT license [14]. Its

Zcash module spends shielded funds in the Ironwood pool, already tested with real funds, and is built on the NU7 releases of the Zcash libraries.

FROST is not in the wallet yet. A standalone prototype, `frost-zec` [16], tests the design first: a DKG with threshold 2 of 3, keys derived as ZIP 2005 specifies, re-randomized signing over PCZTs built and proved by `zcash-devtool`, and a co-signer, run as a separate process, that checks the PCZT itself before signing; files stand in for the QR codes. On 15 October 2026 it made three 2-of-3 spends on mainnet, one for each pair of signers, two of them in two-device mode (blocks 3,518,638 to 3,518,648). It has not been audited.

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, done). Code published under MIT, Zcash module on v6 transactions and the Ironwood pool, tested with real funds, and the `frost-zec` prototype with its mainnet spends.

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, the backup format submitted as a ZIP to zcash/zips, and an audit scope. The ZIP then goes through the ZIP Editors' review.

4) **Android co-signer.** Pairing, transaction decoding and checks, storage under an Android Keystore AES key, and the approval screen, built on the same crate.

5) **iOS co-signer, macOS and Linux coordinator.** Share storage in the iOS and macOS Keychain behind biometric access 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.

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. A prototype has already signed on mainnet; the code and its verification are meant to be exposed to public review, and to an independent audit, before any use with real funds.

**Changes since version 0.3.** The shares and sk are protected at rest by symmetric encryption only, with no elliptic-curve layer, and the backup is encrypted with its passphrase alone, following the security considerations of the backup ZIP (sections 4, 5, 8, 11 and 13); the ZIP draft, its test vectors and its implementations are public (section 10).

**Changes since version 0.2.** Status of the `frost-zec` prototype and its mainnet spends (sections 7 and 13); the ZIP 2005 derivation for the viewing key (sections 3, 5 and 12); the backup format as a ZIP (sections 10 and 13); other FROST projects (sections 3 and 10); macOS and Linux moved to later phases (section 4); other networks reduced to one paragraph (section 2).

# 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, 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`

[16] `frost-zec`, a FROST 2-of-3 prototype for shielded ZEC (MIT), with the mainnet transaction IDs: `https://github.com/OpenPGP-Wallet/frost-zec`

[17] Ironwood (v6) external signing in `zcash-sign`, ZcashFoundation/frost-tools#593: `https://github.com/ZcashFoundation/frost-tools/pull/593`

[18] Zafe, a FROST shared vault for shielded ZEC: `https://github.com/zafe-cash/zafe`

[19] Draft ZIP, "FROST Member Backup Strings for Shielded Zcash", with test vectors: `https://github.com/KHADEVS/zips/blob/draft-khadim-aryaethn-frost-member-backup/zips/draft-khadim-aryaethn-frost-member-backup.md`; discussion with the ZIP Editors: zcash/zips#1413.

[20] ZIP 2005 `use_qsk` key derivation vectors, ZcashFoundation/frost-tools#610: `https://github.com/ZcashFoundation/frost-tools/pull/610`