Implementing Threshold Signatures for FIDO Authentication*

Javier Garcia Martinez[1] and Tjerand Silde[1,2][0000−0002−5455−0409]

[1] Norwegian University of Science and Technology, Trondheim, Norway
tjerand.silde@ntnu.no
[2] PONE Biometrics, Oslo, Norway
https://ponebiometrics.com

Abstract. A hardware authenticator is more secure than a password but remains a single point of failure: whoever holds it can authenticate, and whoever destroys it locks the user out. Threshold signatures remedy both. The signing key is split across N devices so that any T of them can sign, while the relying party still sees one public key and ordinary signatures.

We port the `FROST(secp256k1, SHA-256)` ciphersuite of RFC 9591, the Flexible Round-Optimized Schnorr Threshold signature scheme, to two ARM Cortex-M boards running Zephyr RTOS, one of them the development board of a commercial biometric FIDO2 authenticator, and run a 2-out-of-3 configuration from distributed key generation to a verified signature. Because such devices cannot reach one another, a computer relays every message; we therefore encrypt each secret share end-to-end so that the relay learns nothing, authenticate every message under a device key so that the relay cannot equivocate, and add a blaming protocol that identifies a misbehaving party. The elliptic curve arithmetic fits this hardware comfortably: a signature share takes about half a second, aggregation two milliseconds, and one authentication about a second, half of it storage and coordination.

The signatures our prototype produces are not FIDO2-compliant, but the gap is narrower than it appears: FROST outputs an ordinary Schnorr signature, WebAuthn already registers `EdDSA`, and Ed25519 is itself a Schnorr signature. This work therefore lays the foundation for threshold EdDSA and ECDSA in FIDO authentication for high-security environments.

Keywords: Threshold signatures · FROST · FIDO2 · WebAuthn · Distributed key generation · Embedded cryptography

* This paper is based on the first author's master's thesis [16], carried out in the NTNU Applied Cryptology Lab in collaboration with PONE Biometrics; an extended version is available as a technical report from PONE Biometrics. The work was funded by the Research Council of Norway under project number 321619 (OFFPAD). Source code and build instructions: https://github.com/JavierGarcia22/FROST_Threshold.

1 Introduction

Passwordless authentication with FIDO2 replaced the shared secret with a key pair, but kept one structural weakness: everything depends on a single object. A FIDO2 credential is a private key held by one authenticator. Whoever obtains that authenticator and defeats its user verification can authenticate as the user; a user who loses it is locked out unless a fallback exists, and fallbacks are usually the weakest part of a deployment. In high-assurance settings, both halves matter.

Threshold signatures address both at once. Splitting the signing key across N devices so that any T can sign means an attacker must compromise T devices rather than one, and a user can lose $N - T$ devices without losing the account. For authentication the idea is attractive because it is invisible from the outside: the group public key and the signature are ordinary, so relying parties need no changes at all.

The gap between that observation and a working system is cryptographic engineering, and it is the subject of this paper. Threshold protocols are usually specified for servers that can address each other and have memory to spare. An authenticator is neither: it has kilobytes of RAM, no network stack, and a single physical link to a host, which is a party the user should not have to trust.

1.1 Contributions

The contributions of this paper are the following.

FROST on embedded FIDO authenticator hardware. We port the `FROST(secp256k1, SHA-256)` ciphersuite of RFC 9591 [9] to two ARM Cortex-M boards running Zephyr RTOS 3.7.0, one being the development board of the OFFPAD, a commercial biometric FIDO2 authenticator, and run a 2-out-of-3 configuration from distributed key generation to a verified signature. To our knowledge this is the first reported implementation of FROST key generation and signing on FIDO authenticator hardware.

A key generation hardened for a relaying coordinator. Since the boards cannot reach each other, all traffic passes through a coordinating computer. We encrypt every secret share end-to-end under hashed ElGamal [1] so that the relay learns nothing, sign every message under a long-term device key against a list of public keys provisioned out of band so that the relay cannot equivocate, and add a blaming protocol that resolves a disputed share publicly (Sections 4.2–4.4). One message format carries both phases over UART and USB HID, and the coordinator serves all devices concurrently (Section 4.6).

Measurements that locate the cost. We report computation, memory and per-transport communication costs for the signing phase and for a dealer-based key generation, and find that the curve arithmetic accounts for about half of an authentication, with the rest spent in flash, serialisation and coordination rather than on either link (Section 5). We also state what our evaluation does not yet establish (Section 5.4).

The residual gaps of a relayed deployment. We state the threat model, what the coordinator can and cannot do, how nonce freshness becomes a storage property once nonces are persisted to flash, and the gaps that remain between the prototype and a deployable authenticator, with remediations (Section 6).

A precise statement of the FIDO2 compliance boundary. We show in Section 7 that the barrier is not Schnorr versus ECDSA but the absence of a COSE identifier for the specific curve-and-algorithm pair, and derive a roadmap that is cheaper than a change of signature scheme.

1.2 Related Work

Threshold Schnorr signatures. FROST [23] reduced threshold Schnorr signing to two rounds while permitting unrestricted concurrent sessions, and is now standardised as RFC 9591 [9]. Bellare et al. [4] gave a systematic security treatment of non-interactive threshold signatures, and ROAST [28] wraps FROST-like schemes to obtain robustness rather than the identifiable abort FROST provides alone. Crites, Komlo and Maller [10] gave Sparkle, secure against *adaptive* corruption, which matters because the analysis we inherit assumes static corruption (Section 3.3). NIST's first call for multi-party threshold schemes [8] was finalised in January 2026, with a companion draft on threshold EdDSA and Schnorr [7] bearing on the migration path we recommend. All of this work targets the protocol layer rather than the devices that run it.

Threshold signing on constrained hardware. Mavroudis et al. [26] distributed signing across a board of smart-card ICs so that no single chip has to be trusted, and FROST now runs on microcontroller-class signing devices for cryptocurrency custody [14]; both show that the arithmetic fits such hardware. Our setting differs in what the participant is and whom it talks to: a FIDO authenticator with one link to an untrusted host, and an unmodified relying party.

Threshold ECDSA. This matters because `ES256` is the one algorithm every relying party accepts. Lindell [24] gave fast two-party ECDSA, Gennaro and Goldfeder [17] extended it to the multiparty setting with distributed key generation, and Doerner, Kondi, Lee and shelat [12] gave a three-round protocol replacing zero-knowledge proofs with lightweight consistency checks.

Removing the single point of failure in FIDO2. Two directions have been taken. The first is *key derivation for backup and delegation.* Asynchronous Remote Key Generation (ARKG), introduced by Frymann et al. [15] in an analysis of a Yubico proposal for WebAuthn, lets a sender generate a fresh public key pk' for a receiver such that only that receiver can later derive sk', so a backup authenticator can be registered without being present; Manulis and Nartz [25] extended this to several receivers, any threshold of whom can derive sk'. Distributed ARKG (DARKG) is the closest published work in intent, but its threshold structure governs who can *derive* the credential's private key, whereas ours governs who can *use* it, and the key is never assembled. The two are complementary, and combining them has to

our knowledge not been pursued. ARKG and DARKG are also protocol-design papers with proofs and no embedded implementation, whereas we contribute an implementation and measurements rather than a new protocol.

The second direction is *splitting the signing operation*, which is ours. Larch [11] splits the ability to produce FIDO2 signatures between the user's device and a log server using two-party ECDSA, so every authentication leaves an audit record while unmodified relying parties continue to work. Larch distributes trust between a user and a service to obtain accountability; we distribute the key across several devices the user holds, with no server-side participant, to remove the single point of failure. Both designs are transparent to the relying party.

Formal analysis of FIDO2. Barbosa et al. [3], Bindel, Cremers and Zhao [6], and Hanzlik, Loss and Wagner [19] gave provable-security treatments of FIDO2, CTAP 2.1 and WebAuthn 2. All three model the authenticator as one party holding one key; extending them to several devices behind a relay is open.

2 Background

2.1 Threshold Schnorr Signatures and FROST

We write the group additively, with $[a]P$ the scalar multiplication of a point P by a scalar a, uppercase letters for group elements and lowercase for scalars. A signing key pair is $(\mathsf{sk}, \mathsf{pk})$; the key pair protecting shares in transit is $(\mathsf{dk}, \mathsf{ek})$.

A Schnorr signature [29] on m under $(\mathsf{sk}, \mathsf{pk} = [\mathsf{sk}]G)$ in a group $\mathbb{G}$ of prime order q with generator G is a pair (R, s) with $R = [r]G$ for a fresh nonce r, $c = \mathsf{H}(R, \mathsf{pk}, m)$ and $s = r + c\mathsf{sk}$; verification checks $[s]G = R + [c]\mathsf{pk}$. The signature is linear in the nonce and in the key, which allows it to be assembled from shares but also makes nonce reuse catastrophic: two signatures under the same R on distinct messages yield $\mathsf{sk} = (s - s')/(c - c')$.

In a (T, N) threshold scheme the key is shared with Shamir's scheme [30], so $\mathsf{sk} = \sum_{i \in \mathcal{J}} \lambda_i \mathsf{sk}_i$ for any coalition $\mathcal{J}$ of size T, with λ_i the Lagrange coefficient at 0. The difficulty is the nonce: participants must contribute nonce material without any of them learning the aggregate discrete logarithm, and the aggregate must be bound to one session, or an adversary running many concurrent sessions can mount a ROS-style attack [5]. FROST [23] has each participant P_i draw *two* nonces, a hiding nonce d_i and a binding nonce e_i, and publish the commitments $D_i = [d_i]G$ and $E_i = [e_i]G$. A binding factor ρ_i is then derived from the message and the full commitment list, so that the effective nonce $D_i + [\rho_i]E_i$ is bound to a single session and cannot be reused.

We use `FROST(secp256k1, SHA-256)`, one of five ciphersuites in RFC 9591 [9]. A ciphersuite fixes the group, the encodings, and five domain-separated hash functions H_1 to H_5, for binding factors, challenge, nonce derivation, message and commitment list respectively. The domain separation matters: an implementation

Table 1. COSE algorithm identifiers relevant to a threshold FIDO2 authenticator.

Name	ID	Scheme	Practical status
`ES256`	`-7`	ECDSA / P-256	Universal; the de facto default
`EdDSA`	`-8`	Ed25519, a Schnorr signature	Registered; some authenticators, a minority of relying parties
`ES256K`	`-47`	ECDSA / `secp256k1`	Registered by RFC 8812 [21]; we know of no relying-party support
—	—	Schnorr / `secp256k1`	**No identifier exists**

that instantiates all five as plain SHA-256 interoperates only with itself and falls outside the security analysis.

2.2 What FROST Does and Does Not Guarantee

Two guarantees must be distinguished. *Unforgeability* holds against an adversary that corrupts up to $T - 1$ participants statically, sees their state, and runs arbitrarily many concurrent signing sessions with the honest ones. *Robustness*, meaning guaranteed termination with a signature despite active misbehaviour, does not hold: a participant that submits an invalid share is detected and identified, but the session aborts and must be retried with a different coalition. This is *identifiable abort*, and ROAST [28] is the wrapper that upgrades it to robustness. For a 2-out-of-3 deployment this means that a compromised device can never forge, but a hostile one that keeps volunteering can deny service.

2.3 How WebAuthn Names Signature Algorithms

The part of WebAuthn that matters most for a threshold authenticator is its algorithm negotiation. A relying party lists the algorithms it accepts in `pubKeyCredParams`, each identified by a COSE algorithm identifier from the IANA registry [20]; the authenticator selects the first it supports and aborts if it supports none. The credential public key it returns is a COSE_Key structure carrying key type, algorithm, curve and coordinates, and verification at the relying party is driven by the algorithm field. Table 1 lists the identifiers that matter here.

3 System Design

3.1 Setting and Constraints

The participants are authenticator-class devices: ARM Cortex-M microcontrollers with tens of kilobytes of usable RAM, running Zephyr RTOS, each with one

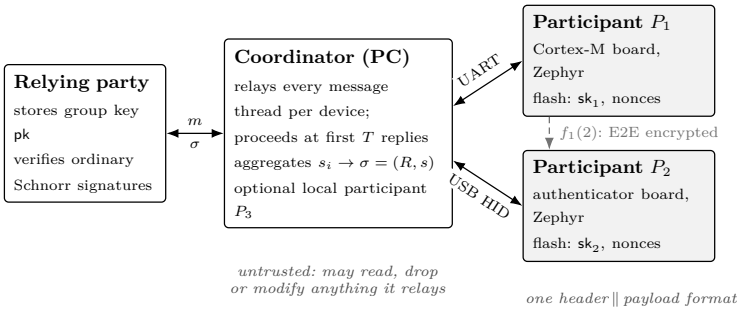

Fig. 1. The prototype. The coordinator relays all traffic over two transports and aggregates signature shares. Secret shares cross it end-to-end encrypted, and every message is signed under the sender's device key, so the coordinator can neither read shares nor alter the transcript. The relying party sees one ordinary key and signature.

physical link to a host. Concretely they are an STM32 Nucleo-L476RG (Cortex-M4, 128 KB SRAM) speaking UART, and the STM32-based development board of the OFFPAD authenticator speaking USB HID; the cryptography on both is the C library `secp256k1-frost` [2], a fork of `libsecp256k1` with a FROST module, whose 64-byte internal point form we inherit on the wire. Figure 1 shows the resulting system. Three constraints shape every subsequent decision: devices cannot address each other, so every message between participants passes through the host, which becomes a *coordinator*; different devices expose different links; and heap and stack are fixed at build time while peak allocation grows with T and N.

3.2 The Coordinator, and What It Is Trusted For

Since this hardware cannot carry $N - 1$ links per board, we adopt the coordinator model: a computer relays all traffic and acts as signature aggregator. We distinguish three properties. *Confidentiality* requires no trust: every share is encrypted end-to-end under the recipient's public key. *Consistency* requires no trust either, but encryption alone does not provide it, since the DKG assumes an authenticated broadcast channel and a relay is not one; every participant therefore signs its messages under a long-term device key verified against a list provisioned out of band (Sections 4.2 and 6.1). What remains is *availability*: the coordinator can drop, delay or corrupt traffic, and honest participants are then drawn into the blaming protocol of Section 4.4, which names the party at fault and aborts. This is identifiable abort, not robustness (Section 2.2): the coordinator can stop the protocol, but it cannot forge, learn the key, or shift the blame onto an honest device.

3.3 Threat Model

We assume static corruption of at most $T - 1$ participants, with the adversary seeing their full state. The coordinator is untrusted: it may read, drop, delay, reorder or modify anything it relays, and if it holds a share it counts against the corruption budget. We make no availability claim against it, since a relay that stops relaying halts any protocol. The list of the N device public keys is provisioned out of band, and the adversary cannot alter it on an honest device. The physical links provide neither confidentiality nor authenticity and are treated as a network under the coordinator's control. The boards are assumed not to be physically compromised beyond the corruption budget: scalar multiplication is constant-time because the underlying library provides it, but we make no claim about power or electromagnetic analysis, and an attacker who can rewrite a board's flash is outside the model. Each honest device is assumed to have a working entropy source, which on both boards is the hardware random number generator.

The claim is about who must participate, not about which messages are signed. An adversary corrupting fewer than T participants and the coordinator besides cannot produce a signature without an honest participant's cooperation in that session, learns nothing about the group key beyond what its own shares and the public key already determine, and cannot cause an honest participant to accept an inconsistent view of the key generation; the most it can do is abort the protocol, with the blame assigned to itself. What an honest participant agrees to sign is, in the prototype, whatever the coordinator presents, so until signing is gated on user verification (Section 6.3) the guarantee protects the key, not the individual authentication. We do not claim robustness (Section 2.2), resistance to physical attack, or any property of the browser-side demonstrator.

4 Implementation

4.1 Distributed Key Generation

A trusted dealer would be simpler and cheaper for the participants, but it concentrates the whole key in one place at one moment, which contradicts the property that motivates the system. RFC 9591 specifies only a trusted dealer, so we implement the two-round DKG of Komlo and Goldberg [23], which is Pedersen's DKG as analysed by Gennaro et al. [18] plus a proof of knowledge against rogue-key attacks: every participant deals its own polynomial and the group key is the sum of the constant terms. ChillDKG [27], a DKG designed for FROST, also dispenses with a broadcast channel, by having participants certify a common transcript; we instead sign every message (Section 4.2), because the same signatures make the blaming protocol sound.

Each participant P_i samples $f_i(x) = a_{i0} + a_{i1}x + \cdots + a_{i(T-1)}x^{T-1}$ over $\mathbb{Z}_q$, broadcasts the commitments $C_{ij} = [a_{ij}]G$ together with a Schnorr proof of knowledge of a_{i0}, verifies everyone else's proof, sends each P_j the share $s_{ij} = f_i(j)$,

verifies each received share s_{ji} against the sender's commitments by checking $[s_{ji}]G = \sum_k [i^k]C_{jk}$, and finally sets $\mathsf{sk}_i = \sum_j s_{ji}$.

The group key and *every* participant's verification share follow from the broadcast commitments alone, for each index $\ell \in \{1, \ldots, N\}$,

$$\mathsf{pk} = \sum_{j=1}^{N} C_{j0}, \qquad \mathsf{pk}_\ell = \sum_{j=1}^{N} \sum_{k=0}^{T-1} [\ell^{\,k}] C_{jk},$$

and an implementation that computes only its own $\mathsf{pk}_i = [\mathsf{sk}_i]G$ cannot check anyone else's signature share, which the aggregator must do.

4.2 Message Authentication

Every device holds a long-term signing key pair, and the list of all N device public keys is provisioned to every device out of band at enrolment. Every protocol message carries a signature under the sender's device key, which the receiver verifies against the provisioned list before processing the payload. This turns the relay into an authenticated channel: the coordinator can still drop or delay a message, but it cannot alter a commitment set, substitute an encryption key, or attribute a message to a device that did not send it; a message that fails verification aborts the session, so the coordinator's only remaining capability is denial of service. The provisioned list is the system's root of trust (Section 6.1).

4.3 End-to-End Encryption of Shares

Every share is encrypted before it leaves the sending device. Textbook ElGamal [13] does not apply directly, because our plaintext is a scalar rather than a group element, so we use hashed ElGamal, the KEM form underlying ECIES and DHIES [1]. Each device generates a key pair $(\mathsf{dk}_j, \mathsf{ek}_j = [\mathsf{dk}_j]G)$ and publishes ek_j, signed under its device key, through the coordinator before any share is sent.

To send s_{ij}, the sender samples $r_{ij} \xleftarrow{\$} \mathbb{Z}_q$, sets $U_{ij} = [r_{ij}]G$, derives $\mathsf{shared}_{ij} = \mathsf{SHA256}(\mathsf{SerializeElement}([r_{ij}]\mathsf{ek}_j))$ from the canonical 33-byte compressed point encoding, and sends (U_{ij}, v_{ij}) with $v_{ij} = \mathsf{SerializeScalar}(s_{ij}) \oplus \mathsf{shared}_{ij}$. The recipient recovers the key as $\mathsf{SHA256}(\mathsf{SerializeElement}([\mathsf{dk}_j]U_{ij}))$. Unlike DHIES we do not include U_{ij} in the hash input; every U_{ij} is fresh, so nothing is lost.

Three implementation details determine interoperability. Hash the canonical encoding, not an in-memory point representation. Treat the masking key as a one-time pad: two shares encrypted under the same ephemeral scalar towards the same recipient reveal their XOR, so each share gets a fresh r_{ij}. Finally, treat "does not decode to an element of $\mathbb{Z}_q$" as a possible outcome of decryption: a corrupted ciphertext unmasks to a uniformly random 256-bit string, which lies outside $\mathbb{Z}_q$ with probability about 2^{-128}.

This encryption provides confidentiality but not integrity: the scheme is malleable, and a bad share is rejected by the verifiable secret sharing (VSS) check after

decryption. Confidentiality therefore comes from the encryption, the integrity of each share from the VSS check, and the integrity of the transcript from the device signatures.

4.4 Blaming

The VSS check tells a participant that a share is wrong but not who is at fault, and since the share travelled encrypted no third party can adjudicate by inspection; our blaming protocol makes the adjudication public. The accuser P_j publishes the indices and the ciphertext (U_{ij}, v_{ij}) it received; the accused P_i responds with the ephemeral scalar r_{ij} for that ciphertext. Anyone can then check that $[r_{ij}]G = U_{ij}$, recompute the masking key, decrypt, and test the result against P_i's broadcast commitments. If the recovered share is valid, P_j lied; otherwise P_i is at fault.

The procedure publishes one evaluation of f_i, which with a corruption budget of $T-1$ is exactly enough to determine a_{i0}. This is acceptable because a complaint means the key generation has failed: the offending party is excluded and the DKG is rerun with fresh polynomials, so the revealed evaluation belongs to a discarded execution. Implementations must enforce this; shares from an execution in which a blame proof was published must never be combined into a signing key.

Because ciphertexts are signed under the sender's device key, an accuser cannot present one the sender did not produce, and a coordinator that modifies one in transit causes a verification failure rather than a false accusation. A party that does not answer a complaint is at fault. Every outcome is an identifiable abort (Section 2.2): the protocol names the responsible party and stops.

4.5 Signing

Signing follows RFC 9591's two rounds. In the first, each participant generates a nonce pair (d_i, e_i) and publishes the commitments $(D_i, E_i) = ([d_i]G, [e_i]G)$. In the second, the aggregator sends the coalition $\mathcal{J}$ the message m and the commitment list B, and each participant P_i, $i \in \mathcal{J}$, computes the binding factor of every member of the coalition, so that all of them derive the same R:

$$\begin{aligned} \rho_j &= \mathsf{H}_1\big(\mathsf{SerializeElement}(\mathsf{pk}) \,\|\, \mathsf{H}_4(m) \,\|\, \mathsf{H}_5(B) \,\|\, \mathsf{SerializeScalar}(j)\big), \\ R &= \textstyle\sum_{j \in \mathcal{J}} (D_j + [\rho_j] E_j), \\ c &= \mathsf{H}_2\big(\mathsf{SerializeElement}(R) \,\|\, \mathsf{SerializeElement}(\mathsf{pk}) \,\|\, m\big), \end{aligned}$$

and returns $s_i = d_i + e_i\rho_i + \lambda_i \mathsf{sk}_i c$. The aggregator checks each share against $[s_i]G = R_i + [c\lambda_i]\mathsf{pk}_i$, where $R_i = D_i + [\rho_i]E_i$, which identifies a misbehaving signer, and outputs $\sigma = (R, \sum_i s_i)$.

Nonce generation. The boards draw the nonce pair from the hardware random number generator through Zephyr's cryptographic random interface; on a microcontroller the generator is a build-time choice, and the non-cryptographic test generator that Zephyr offers for development must not be selected, since nothing

in the protocol's behaviour would reveal the mistake. RFC 9591 goes one step further and derives the nonce as $\mathsf{H}_3(rnd \,\|\, \mathsf{SerializeScalar}(\mathsf{sk}_i))$ from 32 random bytes *rnd* mixed with the signing share, a cheap hedge against biased or short randomness that we recommend as an addition (Section 6.2); it does not help if the randomness repeats exactly, and it is distinct from Ed25519's deterministic derivation from the message, which RFC 9591 warns enables key recovery in the multiparty setting.

Ordering of invalidate and sign. Because the boards must survive power cycles between the two rounds, nonces are persisted in flash, and the order of operations around them is a security property. The firmware marks the pair consumed in flash, flushes, and only then computes and transmits the share, so that a crash between signing and invalidation cannot reuse the nonce after reboot. The cost is one flash write of a few milliseconds in the critical path, against a half-second share computation; the benefit is that nonce reuse after a crash becomes impossible.

4.6 The Transport Layer

All messages, in both protocol phases and over both physical links, share one format: a 12-byte header carrying a magic value, a version, a message type, a payload length and a participant identifier, followed by the payload, so a receiver parses identically regardless of how the bytes arrived. Over UART, at 115200 baud 8N1, reception is interrupt-driven into a ring buffer. Over USB HID reports are a fixed 64 bytes of which the firmware reserves three for framing, so every message is chunked into 61-byte pieces and reassembled against the header length. Assembled messages are handed to Zephyr's work queue so interrupt handlers never block on flash writes.

Group elements travel in the library's 64-byte internal form, scalars in 32 bytes and indices in 4; excluding the header and the device signature that every message carries, a secret share or signature share is 36 bytes, the public key structure 136, the VSS commitments $104{+}64T$ (232 at $T = 2$), the round-1 response 268, and the round-2 request $36{+}132T$ (300 at $T = 2$). Two of these grow with the threshold, and over HID each increment costs additional reports, so raising T costs more in transmission than in computation. Every payload is defined field by field rather than by serialising a library structure wholesale. The distinction matters because a serialised in-memory structure transmits every field it holds, and the library's key-pair structure holds the signing share next to the public key; the round-1 response therefore carries the nonce commitment and the public key structure, and nothing else.

Coordination is concurrent: the coordinator runs one thread per connected device and proceeds as soon as the first T valid responses arrive, so a round costs what the T fastest devices cost rather than what the slowest of N costs. This is where the threshold structure reduces latency; we have not yet quantified the gain.

Table 2. Computation and memory, $T = 2$, $N = 3$; times are the range over 50 runs.

Operation	Where	Time	Memory
Key generation	Coordinator	102–144 ms	204 KB (3.3%)
Signature share	Board	503–523 ms	—
Share aggregation	Coordinator	2 ms	728 KB (11.9%)

Table 3. Time per message in milliseconds, as measured and with the fixed pauses of the instrumented build subtracted (host: 100 ms per HID chunk in key generation; 200 ms per HID chunk and 500 ms per message on both links in signing. Board firmware: 1 s before the signature share plus 50 ms per chunk, the 1050 ms of the last row).

		UART		USB HID	
Phase	**Message (payload)**	meas.	corr.	meas.	corr.
Key generation	Secret share (36 B)	1	1	102	2
Key generation	Public key structure (136 B)	5	5	324	24
Key generation	VSS commitments (232 B)	11	11	433	33
Signing	Round-2 payload (300 B)	521	21	1788	88
Signing	Signature share collection (36 B)	526	526	1223	173

4.7 Board Configuration

Three categories of Zephyr configuration require attention. Heap pool and thread stack sizes must be raised from their defaults, since peak allocation grows with T and N and the failure mode otherwise is a fault inside the library, far from its cause. Flash storage holds the key material and the nonce state, both of which must survive a power cycle. The random subsystem must also be routed to the hardware entropy driver in every build. We pin Zephyr to 3.7.0, since later releases break compatibility with the cryptographic library.

5 Evaluation

5.1 Setup and Measurements

We evaluate a 2-out-of-3 configuration over 50 runs, with one participant local to the coordinator and two on boards; the measured coalition therefore always contains the coordinator's own participant, a measurement convenience rather than the deployment target. Table 2 reports computation and memory, Table 3 the communication cost per transport. Board times come from the kernel's millisecond uptime counter, coordinator times from the high-resolution performance counter; memory is the coordinator process's working set, and the percentages its growth over the value at start-up.

5.2 What the Numbers Say

Four observations follow from the tables.

The measured HID column is pacing, not transport. The HID path has no acknowledgement above the transport: the host delivers each 61-byte chunk as a `SET_REPORT` control transfer that the firmware drains inside the USB callback, so the instrumented build pauses for a fixed time after every chunk and message to keep reports from overtaking one another. Those pauses are the measured column: one, three and four chunks take 102, 324 and 433 ms against 100 ms of pause per chunk. Subtracting them, every message crosses either link in tens of milliseconds, HID costing about 8 ms per control transfer. Replacing the pauses by the stack's own back-pressure would make the corrected column the measured one.

The UART key generation figures fall below the line rate. At 115200 baud 8N1 the three messages need at least 4.2, 12.8 and 21.2 ms; the measured 1, 5 and 11 ms show that the timer stops when the host write call returns, so that column bounds call latency. The signing rows wait for the board and are unaffected.

Half of an authentication is the curve arithmetic. A signature share costs 503–523 ms and aggregation 2 ms. The corrected share-collection rows show what the boards spend beyond that: the Nucleo delivers its share about 0.5 s after the arithmetic, the authenticator board within about 0.2 s, the difference being flash writes for nonce invalidation, serialisation and the diagnostic logging of the instrumented firmware, not the link.

Key generation is measured on the dealer, not on the DKG of Section 4.1, which adds the commitment broadcasts and $N(N-1)$ encrypted share transmissions to the same arithmetic.

5.3 An End-to-End Estimate

No timer spans a complete run, but the components support an estimate. The two signing rows are consecutive and cover the request, the board's computation and the return of the share, and the concurrent coordinator overlaps the T participants, so they sum once: $521 + 526 + 2 \approx 1.05$ s over UART and $1788 + 1223 + 2 \approx 3.01$ s over HID as measured. Removing the pauses barely changes the UART figure, since the host's 500 ms pause overlapped the board's computation; it reduces the HID figure to $88 + 173 + 2$ ms plus the 0.5 s of board work the pause hid, just under 0.8 s. Round one adds tens of milliseconds on either link. Authentication therefore costs **about one second**, the remaining difference between the links being due to the boards rather than the transports; enrolment with the dealer takes roughly 0.12 s over UART and 0.17 s over HID.

5.4 Limitations of This Evaluation

Three gaps bound what can be concluded from these numbers. There is no single-device baseline: we report what a signature share costs but not what an ordinary signature costs on the same board, so the overhead factor cannot be recovered, which is the most valuable missing measurement. The structure of the

two computations bounds it: an ordinary Schnorr signature needs one fixed-base scalar multiplication, whereas a FROST share needs two in round one and T variable-base multiplications in round two to recombine the commitments, so a single-device signature on the same library should cost a fraction of the 503–523 ms share; on the device, the threshold overhead is chiefly those T variable-base multiplications, together with the second round and its storage and transport. Key generation is measured for the dealer rather than the DKG. The boards are not instrumented either: peak heap, stack high-water marks and flash footprint go unreported, and those decide whether the design fits a production device at $N > 3$.

Beyond these, no end-to-end run is timed. All figures are for $T = 2$, $N = 3$; per-participant DKG computation grows linearly in N while share transmissions grow as $N(N-1)$, so communication rather than computation will limit larger configurations. Energy per operation is absent and not derivable from the timings. Finally, conformance against RFC 9591's test vectors, and cross-checking the DKG against an independent implementation [31], should precede any further measurement, since they establish that what is measured is FROST as specified.

6 Security Analysis

The cryptographic core is sound under the threat model of Section 3.3. This section covers what the implementation does to meet that model in practice, and the gaps that remain between it and a deployable authenticator; Table 4 collects the gaps in the order we would close them.

6.1 The Coordinator and the Provisioned Key List

The DKG is analysed assuming an authenticated broadcast channel, which a single relaying computer does not provide. Encryption alone would leave the coordinator able to present different commitment sets to different participants, or to replace an encryption key ek_j with its own and read the shares sent to P_j. The device signatures of Section 4.2 close both: an altered or substituted message fails verification against the provisioned key list and the session aborts with the coordinator identified as the cause. What the coordinator retains is denial of service; a coordinator that drops or corrupts messages draws the honest participants into the blaming protocol, which ends in an identifiable abort rather than a signature, and robustness against it would require ROAST-style retries [28] through a different coordinator.

The arrangement rests on the key list. Whoever controls its provisioning controls group membership, and a device whose list has been tampered with accepts messages from the tamperer's keys. The prototype provisions the list manually before the first run; a product needs an enrolment procedure that establishes it under the user's control, for instance by pairing the devices directly, and fixes it on the device afterwards. This is the counterpart of credential enrolment in FIDO2, and it is where the security of a threshold authenticator is decided.

Table 4. Residual gaps, in suggested order of remediation.

Gap	Consequence today	Remedy
No user-verification binding	Signs on any request	Gate signing on the user gesture
Key list provisioned manually	Enrolment untooled	Pairing procedure fixing the list
Nonces sampled directly	Biased entropy risk	Hedged $\mathsf{H}_3(rnd \,\|\|\, \mathsf{sk}_i)$
Demonstrator bridge is open	Any local page can drive it	Replace it with CTAP

6.2 Nonce Persistence

Persisting nonces to flash makes nonce freshness a storage property, and three requirements follow. The used pair must be invalidated before the share is transmitted (Section 4.5). Invalidation must reach the flash, which means explicit page erasure rather than filesystem deletion over a wear-levelling layer. Lastly, the nonce must come from a cryptographic generator; the hedged derivation of Section 4.5 adds a margin against a generator that is biased without being broken, and is the one hardening step we have not taken. Rollback of the flash partition by an attacker with physical access lies outside the model.

6.3 Session Binding, User Verification, and the Demonstrator

FROST's concurrency safety rests on the binding factor covering the message and the complete commitment list, so the list must be canonically encoded and sorted, and a participant must refuse to produce two shares from one nonce pair, which requires the used-pair check to consult persistent state. Our prototype signs whatever the coordinator presents; a deployable authenticator must gate signing behind the user verification the hardware already provides. The browser-side demonstrator connects through an unauthenticated local WebSocket bridge, a testing convenience for which CTAP is the correct replacement.

7 The FIDO2 Compliance Boundary

The prototype is not a FIDO2 authenticator. A common explanation would be that FIDO2 supports only RSA and ECDSA; that is wrong in both directions, and this section identifies the actual obstacle and the routes past it.

7.1 A Substitution Experiment

We drove a standard WebAuthn registration against a platform authenticator, then substituted our group public key into the attestation object, building the COSE_Key bytes by hand. It fails in three independent ways.

Table 5. Three routes to a relying-party-compatible threshold authenticator.

Route	COSE alg.	Protocol work	Obstacle
FROST(Ed25519, SHA-512)	`EdDSA` (`-8`)	Ciphersuite swap	Relying parties must offer `-8`
Threshold ECDSA on P-256	`ES256` (`-7`)	New protocol, three rounds	Heavier; no Zephyr library
FROST(P-256, SHA-256)	none	Ciphersuite swap	No COSE identifier exists

The key cannot be labelled: the COSE_Key needs an algorithm field, no registered value denotes Schnorr over `secp256k1` (Table 1), and labelling it `ES256K` makes the relying party run ECDSA verification against a Schnorr signature.

The attestation stops attesting: the attestation statement signs the authenticator data, so replacing the credential key inside it invalidates that signature.

The private key does not move: WebAuthn generates the credential key pair inside the platform authenticator and never exports it, so only the public key was ever substituted; the authenticator still holds its own private key and answers every later assertion with it, against a public key the server no longer has.

A group key cannot be injected into an existing credential from outside. A threshold FIDO2 authenticator must itself be the authenticator that the client speaks CTAP to, which is a firmware requirement rather than a data-format one.

7.2 The Obstacle and Three Routes Past It

WebAuthn identifies algorithms by COSE identifier, and no registered identifier denotes Schnorr signatures over `secp256k1`, which our prototype produces. The obstacle is therefore not that Schnorr signatures are excluded from FIDO2: `EdDSA` is a registered algorithm, and Ed25519 is a Schnorr signature. Table 5 lists the three routes that follow from this observation.

The first route is cheap enough to describe in detail. RFC 9591's `FROST(Ed25519, SHA-512)` produces signatures that verify under RFC 8032 [22]. No step of Section 4 changes, nor does Section 6, since message authentication, nonce ordering, and session binding are properties of the construction rather than of the group. The changes are confined to the layer below the group, in four places. The hash layer is a different *construction*, not a different hash: `expand_message_xmd` is removed rather than reparameterised. The challenge hash H_2 loses its domain-separation prefix, which the route requires, since RFC 8032 computes the challenge as SHA-512$(R \,\|\, A \,\|\, M)$ with A the public key. Scalar endianness flips, affecting every serialisation path including the values written to flash. Cofactor 8 reinstates the subgroup and canonical-encoding checks that cofactor 1 let us omit.

One caveat remains: a relying party advertising only `ES256` will never select `EdDSA`, so reach is an empirical question. Verification is not a concern: RFC 9591 mandates the cofactored equation $[8][s]G = [8]R + [8][c]\mathsf{pk}$, but an honestly produced FROST signature has R and pk in the prime-order subgroup and satisfies the cofactorless equation too, so any RFC 8032 verifier accepts it.

The second route is the universally compatible one, which is why threshold ECDSA remains on the roadmap. The third is included because it may look attractive: moving FROST to P-256 aligns the curve with `ES256` but not the algorithm, and no identifier denotes Schnorr over P-256.

The roadmap follows. Build the Ed25519 version first, a library swap and a serialisation rewrite that yields a verifiable credential, and use it to exercise the full CTAP path, which must exist regardless of algorithm; treat threshold ECDSA as the next step; and close the gaps of Table 4 in parallel, beginning with enrolment.

8 Conclusions

Threshold signatures suit authentication well: they remove the single point of failure a hardware token represents while remaining invisible to the relying party. We have shown that the whole protocol, not only its signing half, runs on authenticator hardware: two Cortex-M boards under Zephyr generate a key jointly through a relay that can neither read a share nor alter the transcript, resolve a disputed share without a referee, survive a power cycle between signing rounds, and produce a signature any Schnorr verifier accepts. What separates this prototype from a deployable authenticator is a CTAP interface, a ciphersuite, and the remediations of Table 4: none touches the protocol.

Four lessons generalise beyond FROST and FIDO. A relay the participants cannot bypass is an adversary, not a wire: encryption gives confidentiality, but consistency needs every message authenticated against keys established outside it. Ephemeral secrets that must survive a power cycle acquire a storage lifecycle, so the order of operations around them, and the generator behind them, become security properties. A wire format must be specified field by field, since a serialised structure transmits every field it holds. A ciphersuite should be chosen against the target standard, not by library availability.

Acknowledgements. We thank Magnus Ringerud for co-supervising the master's thesis this work builds on, and for his careful feedback on the final draft of this paper.

References

1. Abdalla, M., Bellare, M., Rogaway, P.: The Oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Topics in Cryptology — CT-RSA 2001. Lecture Notes in Computer Science, vol. 2020, pp. 143–158. Springer (2001). https://doi.org/10.1007/3-540-45353-9_12

2. Banca d'Italia: secp256k1-frost: Implementation of Flexible Round-Optimized Schnorr Threshold Signatures on secp256k1. https://github.com/bancaditalia/secp256k1-frost (2022), GitHub repository
3. Barbosa, M., Boldyreva, A., Chen, S., Warinschi, B.: Provable security analysis of FIDO2. In: Advances in Cryptology — CRYPTO 2021, Part IV. Lecture Notes in Computer Science, vol. 12827, pp. 125–156. Springer (2021). https://doi.org/10.1007/978-3-030-84252-9_5
4. Bellare, M., Crites, E., Komlo, C., Maller, M., Tessaro, S., Zhu, C.: Better than advertised security for non-interactive threshold signatures. In: Advances in Cryptology — CRYPTO 2022, Part IV. Lecture Notes in Computer Science, vol. 13510, pp. 517–550. Springer (2022). https://doi.org/10.1007/978-3-031-15985-5_18
5. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ROS. In: Advances in Cryptology — EUROCRYPT 2021, Part I. Lecture Notes in Computer Science, vol. 12696, pp. 33–53. Springer (2021). https://doi.org/10.1007/978-3-030-77870-5_2
6. Bindel, N., Cremers, C., Zhao, M.: FIDO2, CTAP 2.1, and WebAuthn 2: Provable security and post-quantum instantiation. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 1471–1490. IEEE (2023). https://doi.org/10.1109/SP46215.2023.10179454
7. Brandão, L.T.A.N., Davidson, M.: Notes on Threshold EdDSA/Schnorr Signatures. NIST Internal Report (Initial Public Draft) NIST IR 8214B ipd, National Institute of Standards and Technology (Aug 2022). https://doi.org/10.6028/NIST.IR.8214B.ipd
8. Brandão, L.T.A.N., Peralta, R.: NIST First Call for Multi-Party Threshold Schemes. NIST Internal Report NIST IR 8214C, National Institute of Standards and Technology (Jan 2026). https://doi.org/10.6028/NIST.IR.8214C
9. Connolly, D., Komlo, C., Goldberg, I., Wood, C.A.: The Flexible Round-Optimized Schnorr Threshold (FROST) Protocol for Two-Round Schnorr Signatures. RFC 9591, RFC Editor (2024). https://doi.org/10.17487/RFC9591
10. Crites, E., Komlo, C., Maller, M.: Fully adaptive Schnorr threshold signatures. In: Advances in Cryptology — CRYPTO 2023, Part I. Lecture Notes in Computer Science, vol. 14081, pp. 678–709. Springer (2023). https://doi.org/10.1007/978-3-031-38557-5_22
11. Dauterman, E., Lin, D., Corrigan-Gibbs, H., Mazières, D.: Accountable authentication with privacy protection: The Larch system for universal login. In: 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23). pp. 81–98. USENIX Association (2023), https://www.usenix.org/conference/osdi23/presentation/dauterman
12. Doerner, J., Kondi, Y., Lee, E., shelat, a.: Threshold ECDSA in three rounds. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 3053–3071. IEEE (2024). https://doi.org/10.1109/SP54263.2024.00178
13. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (1985). https://doi.org/10.1109/TIT.1985.1057074
14. Frostsnap: Open-source FROST signing devices. https://frostsnap.com (2025), Source code at https://github.com/frostsnap/frostsnap, accessed September 2026
15. Frymann, N., Gardham, D., Kiefer, F., Lundberg, E., Manulis, M., Nilsson, D.: Asynchronous remote key generation: An analysis of Yubico's proposal for W3C WebAuthn. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer

and Communications Security (CCS '20). pp. 939–954. Association for Computing Machinery (2020). https://doi.org/10.1145/3372297.3417292
16. Garcia Martinez, J.: Threshold Signatures for FIDO Authentication. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway (2025), https://nva.sikt.no/registration/019a0aba4573-1868fcdf-2d4d-481a-98a8-f9c492cc0dd8
17. Gennaro, R., Goldfeder, S.: Fast multiparty threshold ECDSA with fast trustless setup. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18). pp. 1179–1194. Association for Computing Machinery (2018). https://doi.org/10.1145/3243734.3243859
18. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: Advances in Cryptology — EUROCRYPT '99. Lecture Notes in Computer Science, vol. 1592, pp. 295–310. Springer (1999). https://doi.org/10.1007/3-540-48910-X_21, journal version: J. Cryptology 20(1):51–83, 2007, https://doi.org/10.1007/s00145-006-0347-3
19. Hanzlik, L., Loss, J., Wagner, B.: Token meets wallet: Formalizing privacy and revocation for FIDO2. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 1491–1508. IEEE (2023). https://doi.org/10.1109/SP46215.2023.10179373
20. IANA: CBOR Object Signing and Encryption (COSE) Algorithms Registry. https://www.iana.org/assignments/cose/cose.xhtml#algorithms, Accessed September 2026
21. Jones, M.B.: CBOR Object Signing and Encryption (COSE) and JSON Object Signing and Encryption (JOSE) Registrations for Web Authentication (WebAuthn) Algorithms. RFC 8812, RFC Editor (2020). https://doi.org/10.17487/RFC8812
22. Josefsson, S., Liusvaara, I.: Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, RFC Editor (2017). https://doi.org/10.17487/RFC8032
23. Komlo, C., Goldberg, I.: FROST: Flexible round-optimized Schnorr threshold signatures. In: Selected Areas in Cryptography — SAC 2020. Lecture Notes in Computer Science, vol. 12804, pp. 34–65. Springer (2021). https://doi.org/10.1007/978-3-030-81652-0_2
24. Lindell, Y.: Fast secure two-party ECDSA signing. In: Advances in Cryptology — CRYPTO 2017, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 613–644. Springer (2017). https://doi.org/10.1007/978-3-319-63715-0_21
25. Manulis, M., Nartz, H.: Distributed asynchronous remote key generation. In: Applied Cryptography and Network Security — ACNS 2025. Lecture Notes in Computer Science, vol. 15825, pp. 486–515. Springer (2025). https://doi.org/10.1007/978-3-031-95761-1_17
26. Mavroudis, V., Cerulli, A., Svenda, P., Cvrcek, D., Klinec, D., Danezis, G.: A touch of evil: High-assurance cryptographic hardware from untrusted components. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017). pp. 1583–1600. ACM (2017). https://doi.org/10.1145/3133956.3133961
27. Ruffing, T., Nick, J., Melnyk, I., Zhvanko, M., Dhakshinamoorthy, S.: ChillDKG: Distributed key generation for FROST. Bitcoin Improvement Proposal draft, https://github.com/BlockstreamResearch/bip-frost-dkg (2024), accessed October 2026
28. Ruffing, T., Ronge, V., Jin, E., Schneider-Bensch, J., Schröder, D.: ROAST: Robust asynchronous Schnorr threshold signatures. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22). pp. 2551–2564. Association for Computing Machinery (2022). https://doi.org/10.1145/3548606.3560583

29. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology **4**(3), 161–174 (Jan 1991). https://doi.org/10.1007/BF00196725
30. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (Nov 1979). https://doi.org/10.1145/359168.359176
31. Zcash Foundation: FROST: Rust implementation of Flexible Round-Optimized Schnorr Threshold signatures. https://github.com/ZcashFoundation/frost (2024), Reference implementation of RFC 9591