

CA-less Mutual Co-Signing of Documents over a Unidirectional Visual Channel with Transported Hardware Attestation

[**TODO:** author name(s)]
[**TODO:** affiliation / independent]
[**TODO:** contact email]

July 10, 2026

Preprint. [**TODO:** decide: anonymous submission? ePrint allows it.]

Abstract

We describe and analyze a protocol for *mutual co-signing* of a document by two mobile devices that (i) communicate only over a one-way, lossy, low-bandwidth optical channel (an animated on-screen code read by the counterparty’s camera), (ii) use no intermediary server on the trust path, and (iii) use no certificate authority. Trust in each party’s public key is instead grounded in a *hardware attestation token* produced by the platform secure element, which is transported in full over the visual channel by a rateless (fountain) code and cryptographically bound into the co-signature. The core technical contribution is a *two-stage hash anchor* that removes the circular signing dependency inherent to interactive co-signing: the first party commits to the document before the identity of the second party is known, and the second party’s identity is later bound to that commitment without invalidating the first signature. We give a threat model, define three security properties (*anchor binding*, *co-signature inseparability*, and *attestation-bound key provenance*), and reduce them to standard assumptions (collision resistance of H and EUF-CMA security of the underlying signature scheme), with the secure element modeled as an ideal signing oracle. We report a working instantiation on iOS/Android using ECDSA P-256 in the Secure Enclave/StrongBox, SHA-256, Apple App Attest / Play Integrity, and an LT-style fountain code, together with an independent third-party verifier that recomputes all anchors and checks both signatures fully offline. [**TODO:** One sentence of quantitative results once you decide what to measure — e.g. frames-to-decode distribution, payload sizes, verification time.]

Keywords. digital signatures · hardware attestation · certificate-less trust · offline protocols · fountain codes · secure element · signature chaining.

Contents

1	Introduction	1
2	Related Work	2
3	Preliminaries and Notation	3

4	System Model and Threat Model	4
4.1	Parties and channel	4
4.2	Adversary	4
4.3	Trust assumptions	4
5	Construction	4
5.1	Canonical field encoding and separator injection defense	5
5.2	Two-stage hash anchor	5
5.3	Signatures and chaining	5
5.4	Transported hardware attestation and key provenance	5
5.5	Rateless framing of the transfer structure	6
5.6	Composite verification hash	6
6	Security Analysis	6
6.1	Anchor binding	6
6.2	Co-signature inseparability	7
6.3	Attestation-bound key provenance	9
6.4	Post-signing tamper evidence	9
7	Implementation	10
8	Limitations and Trade-offs	10
9	Conclusion	11
A	Worked Example (redacted)	11
B	Full finalHash field order (v81)	11

1 Introduction

Problem. Two people meet in person, each holding a phone, and want to co-sign a document so that a third party can later verify *both* signatures and the fact that each signing key lives in genuine device hardware. The setting we target has three hard constraints simultaneously: *no network* at signing time; *no server* that the signed data passes through; and *no certificate authority* to vouch for either public key. The only channel is optical: one phone displays a machine-readable code, the other reads it with its camera. The channel is one-way per turn, lossy (dropped frames), and capacity-limited.

Why this is not trivial. Two difficulties compound.

1. **Circular dependency.** In naive interactive co-signing, each party wants to sign “the final document,” but the final document includes the other party’s identity and signature, which do not exist yet. Party A cannot sign until B is known; B cannot be bound until A has signed. We break this with a two-stage hash anchor (Section 5.2).
2. **Key trust without a CA.** With no CA and no network, a receiver cannot fetch or validate a certificate chain for the sender’s key online. We replace certificate-based trust with *transported hardware attestation*: the full platform attestation token, which cryptographically ties the public

key to key generation inside a genuine secure element, is carried over the optical channel itself and bound into the co-signature (Section 5.4). Because a full attestation object exceeds the capacity of a single static optical code, this is only made practical by rateless framing (Section 5.5).

Contributions.

- A two-stage hash-anchor construction that provably removes the circular signing dependency while keeping the first signature stable (Sections 5.2, 6).
- A signature-chaining rule that makes the second signature *inseparable* from a specific first signature, preventing “re-pasting” a signature onto a different document or a different first signature (Sections 5.3, 6).
- A CA-less key-provenance mechanism based on transporting the *full* hardware-attestation token point-to-point over the visual channel and binding its hash into the signed anchor, with an offline verification path when the attestation scheme offers a certificate chain to a manufacturer root, and a detachable third-party path otherwise (Section 5.4).
- An engineering instantiation and a standalone verifier that reproduces all anchors and both signatures with no network access (Section 7).

Scope and honesty about what this is *not*. This paper is about a *cryptographic protocol*, not about the legal status of its output in any jurisdiction. **[TODO: IMPORTANT: strip every legal/eIDAS/AES claim from the paper. The current product artifacts carry “Art. 207 CCU / AES / eIDAS Art. 26” banners; those must not appear here. A CA-less, attestation-based scheme is interesting to cryptographers on its own terms; legal framing reads as marketing on ePrint and will hurt the paper.]** We also do not claim the scheme is a substitute for a qualified/PKI signature; the absence of a CA is a design goal with explicit trade-offs, discussed in Section 8.

2 Related Work

[TODO: This section must be written carefully and honestly; on ePrint a weak related-work section is the fastest way to lose credibility. Cover at least the following clusters and state precisely what is *different*, not just what exists. Add real citations.]

- **Multi-signatures / signature aggregation / co-signing.** **[TODO: cite. Distinguish: those assume a PKI or an online setting; here neither the ordering trick (two-stage anchor) nor the offline visual transport is present.]**
- **Certificate-less / identity-based public-key cryptography.** **[TODO: cite CL-PKC line. Distinguish: those remove certificates but rely on a key generation center / trusted authority; our trust root is the hardware manufacturer’s attestation root, reached without a network at signing time.]**
- **Remote/hardware attestation (App Attest, Play Integrity, FIDO, TPM quotes, DAA/EPID).** **[TODO: cite. Distinguish: attestation is normally verified server-side; here the *full* token is transported peer-to-peer and bound into a mutual signature.]**

- **Rateless / fountain codes (LT, Raptor, RaptorQ) and animated visual codes.** [TODO: cite Luby LT; Shokrollahi Raptor; and prior “animated QR” transports. Distinguish: prior visual fountain transports are a one-way data pipe; none binds the transport to mutual signing + attestation.]
- **Provenance / capture-time signing (e.g. content authenticity).** [TODO: cite. Distinguish: one-sided, not mutual co-signing.]

The novelty claim is the *combination*: to our knowledge no prior work co-signs a document mutually, offline, with certificate-less key trust established by a transported hardware-attestation token carried over a rateless visual channel. [TODO: Verify this claim against a real literature search before posting.]

3 Preliminaries and Notation

Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^{256}$ be a collision-resistant hash function (instantiated as SHA-256). We write $x \parallel y$ for concatenation and use two non-printing control bytes as field delimiters: $\langle RS \rangle$ (ASCII 30, “record separator”, 0x1E) between top-level fields, and $\langle US \rangle$ (ASCII 31, “unit separator”, 0x1F) inside a compound field. All strings are UTF-8 encoded before hashing.

A *secure-element signature scheme* $\text{Sig} = (\text{KGen}, \text{Sign}, \text{Vrfy})$ is a standard signature scheme (instantiated as ECDSA over NIST P-256 with SHA-256, DER-encoded, Base64-serialized) with the additional property that sk is generated inside, and never leaves, a hardware secure element; the signing operation is gated by a user-presence factor (biometric). We model the secure element as an oracle $\mathcal{O}_{SE}$ that holds sk and, on input m , returns $\text{Sign}(sk, m)$ only after a presence check; $\mathcal{A}$ never obtains sk .

An *attestation scheme* $\text{Att} = (\text{Gen}, \text{AttVer})$ produces, for a secure-element public key pk and a challenge c , a token τ such that AttVer accepts (pk, c, τ) only if pk was generated inside a genuine secure element of a device of the attested platform, and c was bound at generation time. [TODO: State the exact soundness assumption you are willing to make about App Attest / Play Integrity, and cite the platform security model. Be precise about *dev vs. production* attestation — see Remark 1.]

Remark 1 (Development-mode attestation in the current artifact). [TODO: The sample signed document reports an **AppAttest-Dev** token (development environment). A production security claim must use production App Attest / Play Integrity. State this honestly; do not present dev attestation as if it were the full hardware guarantee.]

4 System Model and Threat Model

4.1 Parties and channel

Two mobile devices, D_A (initiator) and D_B (responder), each with a secure element, a camera, and a display. The only communication is the optical channel: $D_A \rightarrow D_B$ (A’s animated code read by B), then $D_B \rightarrow D_A$ (B’s animated code read by A). No server carries signed data; no CA is contacted. [TODO: Also state the online variant (link/relay) and be explicit that in that variant a relay server exists but is *not* on the trust path — data passing through it is already committed and signed.]

4.2 Adversary

We consider (and should analyze separately) at least these adversaries:

$\mathcal{A}_{\text{net}}$ — **hostile network / relay**. Controls any relay or network used for out-of-band transport; may drop, reorder, replay, or inject frames. Goal: alter the signed content, or make a verifier accept a document neither party signed. **[TODO: state assumed powers precisely.]**

$\mathcal{A}_{\text{repaste}}$ — **signature re-paster**. A legitimate but malicious counterparty, or an observer of the optical channel, who has seen valid (pk_A, σ_A) and/or B’s signature and tries to transplant a signature onto a different document or a different first signature.

$\mathcal{A}_{\text{key}}$ — **key-provenance forger**. Tries to make a verifier accept a key as hardware-backed when it is not (e.g. a software key on an emulator), i.e. tries to defeat attestation binding.

$\mathcal{A}_{\text{post}}$ — **post-signing tamperer**. Given the fully signed document, tries to change any field (amount, party, time, geolocation, evidence) without detection.

4.3 Trust assumptions

- H is collision-resistant; Sig is EUF-CMA-secure; the secure element does not leak sk .
- The attestation manufacturer root(s) are authentic and are embedded in the verifier out of band (pinned). **[TODO: state exactly which roots and how pinned.]**
- **Out of scope (state explicitly):** coercion/duress of a present user; malware with full control of a rooted/jailbroken device that can drive the biometric prompt; the legal weight of the output. **[TODO: The code carries a duressFlag; note it is not integrity-protected and is out of the cryptographic threat model.]**

5 Construction

We present the scheme bottom-up: canonical encoding (5.1), the two-stage anchor (5.2), signature chaining (5.3), attestation transport and binding (5.4), rateless framing (5.5), and the final composite verification hash (5.6). Formulas below match the deployed v81 scheme and the worked example in Appendix A.

5.1 Canonical field encoding and separator injection defense

Document content is serialized to a canonical UTF-8 string as an ordered list of named lines joined by `\n`. Every field value v is passed through an escaping map $f(\cdot)$ that replaces any embedded `<RS>` byte with `<US>`, so that no field value can forge a field boundary. Compound fields (nickname `<US>` avatar-index) use `<US>` internally, distinct from the top-level `<RS>`. This prevents concatenation collisions in which two distinct field vectors serialize to the same byte string. **[TODO: Add Figure (patent Fig. 4) and prove the injectivity lemma (Lemma 1).]**

5.2 Two-stage hash anchor

Let C denote the canonical content string containing the document terms and *only party A’s* identity fields (no B fields, no signatures, no public keys). Define the **first anchor**

$$h_1 = H(\text{"v81"} \parallel \langle \text{RS} \rangle \parallel C).$$

Crucially, h_1 is computable by D_A *before B's identity is known*. When B joins, define the **second anchor** as a deterministic function of h_1 and B's identity fields $id_B = (nick_B, avatar_B[, tgHash_B])$:

$$h_2 = H(h_1 \parallel \langle RS \rangle \parallel nick_B \parallel \langle RS \rangle \parallel avatar_B \parallel \langle RS \rangle \parallel tgHash_B).$$

The versioned prefix fixes the scheme and rules out cross-scheme collisions. **[TODO: Add Figure (patent Fig. 5).]**

5.3 Signatures and chaining

With document identifier id , the two signed messages are

$$\begin{aligned} m_A &= "A:" \parallel id \parallel ":" \parallel h_1, \\ m_B &= "B:" \parallel id \parallel ":" \parallel h_2 \parallel ":" \parallel pk_A \parallel ":" \parallel \sigma_A, \end{aligned}$$

and $\sigma_A = \text{Sign}(sk_A, m_A)$, $\sigma_B = \text{Sign}(sk_B, m_B)$, each produced inside the respective secure element under biometric gating. The domain-separating prefixes "A:"/"B:" prevent cross-role message collisions. Because m_B contains *both* pk_A and σ_A , the second signature is bound to one specific, already-existing first signature. **[TODO: Add Figure (patent Fig. 6).]**

5.4 Transported hardware attestation and key provenance

D_A obtains a full attestation token τ_A binding pk_A to secure-element key generation, with challenge c tied to h_1 or the session id. The *full* τ_A (not merely a boolean “is-real”) is placed in the transfer structure; only its context-prefixed hash $H(\text{"pacto_device_A:"} \parallel \tau_A)$ enters the signed anchor set. Verification of provenance has two modes: (i) *fully offline* when τ carries a certificate chain to a pinned manufacturer root (e.g. App Attest), checked locally; and (ii) a *detachable third-party* step when provenance is only verifiable against a manufacturer service (e.g. DeviceCheck / Play Integrity). **[TODO: State precisely which mode your production build uses and prove the challenge-binding prevents token replay (Proposition 1).]**

5.5 Rateless framing of the transfer structure

The self-contained transfer structure (content fields, h_1 , σ_A , pk_A , full τ_A , scheme id, session id) is compressed and split into k base fragments (max ≈ 675 bytes each); base frames carry one fragment, fountain frames carry the XOR of a pseudo-random subset chosen by an LCG (multiplier 1664525, increment 1013904223, modulus 2^{32} , seeded by the frame sequence number), with redundancy factor 3 and degree $(seq \bmod k) + 1$. The decoder peels fragments and confirms completion via a short checksum (prefix of $H(\text{payload})$). Because the code is rateless, no back-channel is needed to request lost frames. **[TODO: Either (a) analyze decoding success probability / expected frames-to-decode for these exact parameters, or (b) explicitly frame the specific LCG-XOR scheme as an engineering choice and defer optimality to standard LT/Raptor analysis. Do NOT overclaim optimality of the custom degree rule.]**

5.6 Composite verification hash

For archival and third-party checking, a composite hash $finalHash$ is computed over an ordered 39-field **v81** record (both parties' public keys, signatures, biometric-backing flags, device-token hashes, timestamps, geolocation, evidence hashes, identity hashes, and h_1), with the B-block included only when $\sigma_B \neq \varepsilon$. Non-deterministic rendering bytes (any PDF-file hash) are *deliberately excluded*, so

the display file can be regenerated without breaking any signature. [TODO: Reproduce the exact field list in Appendix B (it is already in the verifier output) and cross-check byte-for-byte.]

6 Security Analysis

[TODO: This is the section that turns a “system description” into a cryptography paper. Each property below needs (1) a game-based definition and (2) a reduction. The constructions are simple enough that these are writable; the stubs indicate the intended argument.]

6.1 Anchor binding

Definition 1 (Anchor binding). [TODO: Define the game: $\mathcal{A}$ wins if it produces two distinct content vectors (or a content vector and an id_B) mapping to the same h_1 (resp. h_2) accepted by the verifier.]

Theorem 1. *If H is collision-resistant and the canonical encoding is injective (Lemma 1), then the two-stage anchor is binding: any adversary winning Definition 1 yields a collision in H .*

Proof sketch. [TODO: Reduction to CR of H ; the versioned prefix and $\langle RS \rangle / \langle US \rangle$ escaping give injectivity, so distinct inputs cannot collide except via a hash collision.] $\square$

Lemma 1 (Encoding injectivity). *Fix a field schema of arity n (an ordered list of n slots, each either a simple slot or a compound slot of two sub-fields). Let enc be the canonical encoder of Section 5.1: it applies the escaping map f to every field value (replacing every $\langle RS \rangle$ byte inside a value by $\langle US \rangle$), joins the n resulting tokens with the top-level separator $\langle RS \rangle$, and joins the two sub-fields of a compound slot with the inner separator $\langle US \rangle$; the byte $\langle RS \rangle$ appears in the output only as a top-level field separator. Then, restricted to inputs whose sub-field values contain no $\langle US \rangle$ byte in a position that would forge a sub-boundary, enc is injective: for any two field vectors $\vec{v} \neq \vec{w}$ of the same schema, $\text{enc}(\vec{v}) \neq \text{enc}(\vec{w})$.*

Proof. The schema arity n and the compound/simple type of each slot are fixed and public, so a decoder knows exactly where boundaries are expected. We show the encoding is decodable, which implies injectivity.

Given a byte string $s = \text{enc}(\vec{v})$, split s on the byte $\langle RS \rangle$. By construction $\langle RS \rangle$ occurs in s *only* between top-level tokens: inside every field value the encoder has already rewritten each $\langle RS \rangle$ to $\langle US \rangle$ via f , so no $\langle RS \rangle$ survives inside a token. Hence the split yields exactly the n tokens $f(v_1), \dots, f(v_n)$ in order, with no ambiguity about token count or boundary position (arity is fixed). For a simple slot this recovers $f(v_i)$; for a compound slot, split the token on $\langle US \rangle$: because the schema fixes the slot as compound and the two sub-fields are constrained not to introduce a spurious $\langle US \rangle$, this recovers the ordered pair $(f(a_i), f(b_i))$.

Thus $\text{enc}(\vec{v})$ determines the ordered tuple $(f(v_1), \dots, f(v_n))$ (with compound slots split), i.e. f applied component-wise. Two distinct vectors $\vec{v} \neq \vec{w}$ differ in at least one component i ; if $f(v_i) \neq f(w_i)$ the recovered tuples differ and so do the encodings. The only remaining case is $v_i \neq w_i$ but $f(v_i) = f(w_i)$: since f only rewrites $\langle RS \rangle \mapsto \langle US \rangle$ and is otherwise the identity, $f(v_i) = f(w_i)$ with $v_i \neq w_i$ requires two values that differ solely in $\langle RS \rangle$ -vs- $\langle US \rangle$ bytes at the same positions. The schema forbids raw $\langle RS \rangle$ inside sub-fields as an admissible input (values are the escaped canonical forms), so on the admissible domain f is the identity and $f(v_i) = f(w_i) \Rightarrow v_i = w_i$, a contradiction. Hence $\text{enc}(\vec{v}) \neq \text{enc}(\vec{w})$. $\square$

Remark 2 (Domain caveat, stated honestly). Injectivity holds on the admissible domain, i.e. where field values do not themselves contain a raw $\langle \text{RS} \rangle$ that a caller expected to survive as data. The escaping map f enforces this by collapsing $\langle \text{RS} \rangle \mapsto \langle \text{US} \rangle$ inside values, which is a lossy rewrite on non-admissible inputs. In practice the hashed fields are UUIDs, base64/hex digests, ISO-8601 timestamps, and short human strings, none of which legitimately contain the control bytes $\langle \text{RS} \rangle$ ($0\text{x}1\text{E}$) or $\langle \text{US} \rangle$ ($0\text{x}1\text{F}$); the version prefix "v81" additionally domain-separates the scheme. This is the exact property the proofs of Theorems 2 and 3 rely on. **[TODO: If you want a fully unconditional statement, switch to length-prefixed (TLV) encoding and drop the caveat; note the trade-off in the paper.]**

6.2 Co-signature inseparability

Recall $m_B = \text{"B:"} \parallel id \parallel \text{" :"} \parallel h_2 \parallel \text{" :"} \parallel pk_A \parallel \text{" :"} \parallel \sigma_A$ and $\sigma_B = \text{Sign}(sk_B, m_B)$. We formalize the intuition that σ_B cannot be detached from the specific first signature it endorses and re-attached to a different one.

We first fix what the verifier accepts. A *co-signed record* is a tuple $R = (id, C, id_B, pk_A, \sigma_A, pk_B, \sigma_B)$, where C is the canonical A-content string. Let $h_1 = h_1(C)$ and $h_2 = h_2(h_1, id_B)$ as in Section 5.2, and let m_A, m_B be as in Section 5.3. The third-party verifier accepts R , written $\text{Vrfy}^*(R) = 1$, iff

$$\text{Vrfy}(pk_A, m_A, \sigma_A) = 1 \wedge \text{Vrfy}(pk_B, m_B, \sigma_B) = 1.$$

(Attestation checks on pk_A, pk_B are treated in Section 5.4 and are orthogonal here.)

Definition 2 (Co-signature inseparability). Consider the following game between a challenger and a PPT adversary $\mathcal{A}$ (an $\mathcal{A}_{\text{repaste}}$).

1. The challenger runs $(pk_B, sk_B) \leftarrow \text{KGen}(1^\lambda)$ inside an ideal secure element and gives pk_B to $\mathcal{A}$. It initializes an empty set Q of endorsement records.
2. $\mathcal{A}$ interacts with a *B-signing oracle* $\mathcal{O}_B$: on a query $(id, C, id_B, pk_A, \sigma_A)$ with $\text{Vrfy}(pk_A, m_A, \sigma_A) = 1$, the oracle computes $h_2 = h_2(h_1(C), id_B)$, forms m_B , returns $\sigma_B \leftarrow \text{Sign}(sk_B, m_B)$, and records $(id, h_2, pk_A, \sigma_A) \in Q$. $\mathcal{A}$ may query $\mathcal{O}_B$ adaptively and may itself generate as many A-side keypairs and A-signatures as it likes.
3. $\mathcal{A}$ outputs a record $R^* = (id^*, C^*, id_B^*, pk_A^*, \sigma_A^*, pk_B, \sigma_B^*)$.

Let $h_2^* = h_2(h_1(C^*), id_B^*)$. $\mathcal{A}$ *wins* if $\text{Vrfy}^*(R^*) = 1$ and $(id^*, h_2^*, pk_A^*, \sigma_A^*) \notin Q$, i.e. B's signature verifies against a $(id, h_2, pk_A, \sigma_A)$ combination B never endorsed. Denote its advantage $\text{Adv}_{\mathcal{A}}^{\text{insep}}(\lambda) = \Pr[\mathcal{A} \text{ wins}]$.

Remark 3. Winning captures exactly “re-pasting”: keeping σ_B but presenting it under a different document id, a different second-party identity (hence different h_2), a different A-key, or (the headline attack) a *different first signature* $\sigma_A^* \neq \sigma_A$ on the same document.

Theorem 2 (Inseparability). *If Sig is EUF-CMA-secure, then for every PPT $\mathcal{A}$ the advantage $\text{Adv}_{\mathcal{A}}^{\text{insep}}(\lambda)$ is negligible. Concretely, there is a PPT reduction $\mathcal{B}$ with*

$$\text{Adv}_{\mathcal{A}}^{\text{insep}}(\lambda) \leq \text{Adv}_{\mathcal{B}, \text{Sig}}^{\text{euf-cma}}(\lambda).$$

Proof. We build $\mathcal{B}$ that breaks EUF-CMA of Sig using any winning $\mathcal{A}$. $\mathcal{B}$ receives a challenge public key pk^* and access to a signing oracle $\mathcal{O}_{\text{Sign}}(\cdot)$ for the secret key sk^* behind pk^* ; it must output a pair (m, σ) with $\text{Vrfy}(pk^*, m, \sigma) = 1$ and m never queried to $\mathcal{O}_{\text{Sign}}$.

$\mathcal{B}$ runs $\mathcal{A}$, setting the game's $pk_B := pk^*$ (a perfect simulation, since in the real game pk_B is an honestly generated public key and sk_B is used only through the ideal secure element, exactly matching an EUF-CMA signing oracle). It maintains Q and a message log M .

Answering $\mathcal{O}_B$. On query $(id, C, id_B, pk_A, \sigma_A)$ with $\text{Vrfy}(pk_A, m_A, \sigma_A) = 1$, $\mathcal{B}$ computes h_2 , forms m_B , calls $\sigma_B \leftarrow \mathcal{O}_{\text{Sign}}(m_B)$, adds m_B to M and $(id, h_2, pk_A, \sigma_A)$ to Q , and returns σ_B . This is distributed identically to the real oracle.

Extraction. When $\mathcal{A}$ halts with a winning R^* , $\mathcal{B}$ forms $m_B^* = \text{"B:"} \parallel id^* \parallel \text{" :"} \parallel h_2^* \parallel \text{" :"} \parallel pk_A^* \parallel \text{" :"} \parallel \sigma_A^*$ and outputs (m_B^*, σ_B^*) .

Validity of the forgery. Since $\mathcal{A}$ wins, $\text{Vrfy}^*(R^*) = 1$, so in particular $\text{Vrfy}(pk^*, m_B^*, \sigma_B^*) = 1$. It remains to show $m_B^* \notin M$. Suppose for contradiction $m_B^* \in M$; then some oracle query produced $m_B = m_B^*$ for a recorded tuple $(id, h_2, pk_A, \sigma_A) \in Q$. The message m_B is the concatenation, under the fixed public template $\text{"B:"} id \text{" :"} h_2 \text{" :"} pk_A \text{" :"} \sigma_A$, of five components separated by the literal byte $\text{" :"} \text{"}$. We claim this template is uniquely decodable into $(id, h_2, pk_A, \sigma_A)$. Indeed id is a UUID of fixed syntax not containing $\text{" :"} \text{"}$ outside its own hyphenated form [**TODO: state UUID canonical form; it contains no colon**], h_2 is a fixed-length lowercase hex digest (no $\text{" :"} \text{"}$), pk_A has the fixed shape $\text{Base64}(X) \text{" :"} \text{"Base64}(Y)$ occupying exactly the next two colon-delimited base64 tokens of known length, and σ_A is the final base64 token; base64 and hex alphabets exclude $\text{" :"} \text{"}$. Hence the byte string m_B^* determines $(id^*, h_2^*, pk_A^*, \sigma_A^*)$ uniquely, and $m_B = m_B^*$ forces $(id, h_2, pk_A, \sigma_A) = (id^*, h_2^*, pk_A^*, \sigma_A^*)$. But then $(id^*, h_2^*, pk_A^*, \sigma_A^*) \in Q$, contradicting the winning condition $(id^*, h_2^*, pk_A^*, \sigma_A^*) \notin Q$. Therefore $m_B^* \notin M$, and (m_B^*, σ_B^*) is a valid EUF-CMA forgery.

$\mathcal{B}$ runs in time comparable to $\mathcal{A}$ plus the cost of simulating the oracle, and succeeds whenever $\mathcal{A}$ wins, giving the stated bound. $\square$

Remark 4 (Why the chaining fields are load-bearing). The forgery extraction used only that m_B commits, injectively, to $(id, h_2, pk_A, \sigma_A)$. Dropping pk_A or σ_A from m_B (as in the legacy `signMessageB` variants that omit the chain) would let $\mathcal{A}$ present the same σ_B under a different σ_A^* without leaving the endorsed set, and the contradiction step fails. The chain pk_A, σ_A inside m_B is thus exactly what buys inseparability.

6.3 Attestation-bound key provenance

Proposition 1 (Provenance / no replay). [**TODO: Under the attestation soundness assumption and challenge-to- h_1 /session binding, an $\mathcal{A}_{\text{key}}$ cannot make a software/emulator key accepted as hardware-backed, nor replay another device's token for a fresh session.**]

6.4 Post-signing tamper evidence

We now show that no field committed by the first signature can be altered after signing without either forging `Sig` or breaking collision resistance of `H`. Write $\vec{C}$ for the vector of A-content fields (Section 5.1), so that $C = \text{enc}(\vec{C})$, $h_1 = \text{H}(\text{"v81"} \parallel \langle \text{RS} \rangle \parallel C)$, and $m_A = \text{"A:"} \parallel id \parallel \text{" :"} \parallel h_1$.

Definition 3 (Content tamper resistance). Consider the following game with PPT adversary $\mathcal{A}$ (an $\mathcal{A}_{\text{post}}$).

1. The challenger generates $(pk_A, sk_A) \leftarrow \text{KGen}(1^\lambda)$ in an ideal secure element and gives pk_A to $\mathcal{A}$.
2. $\mathcal{A}$ adaptively queries an *A-signing oracle*: on a content vector $\vec{C}$ (with document id id) the oracle returns $\sigma_A \leftarrow \text{Sign}(sk_A, m_A)$ where m_A is derived from $\vec{C}$ as above, and records $\vec{C}$ in a set Q_A .
3. $\mathcal{A}$ outputs $(\vec{C}^*, \sigma_A^*)$.

$\mathcal{A}$ wins if $\text{Vrfy}(pk_A, m_A^*, \sigma_A^*) = 1$ (where m_A^* is derived from $\vec{C}^*$) and $\vec{C}^* \notin Q_A$, i.e. the verifier accepts A's signature over a content vector A never signed. Let $\text{Adv}_{\mathcal{A}}^{\text{tamper}}(\lambda) = \Pr[\mathcal{A} \text{ wins}]$.

This is the right formalization of “any change to a signed field is detected”: if an attacker takes a legitimately signed document, alters one committed field (amount, party, timestamp, geolocation, evidence digest, ...), and the altered document still verifies, then the attacker has produced an accepting $(\vec{C}^*, \sigma_A^*)$ with $\vec{C}^* \notin Q_A$.

Theorem 3 (Tamper evidence). *If Sig is EUF-CMA-secure and H is collision-resistant, then for every PPT $\mathcal{A}$,*

$$\text{Adv}_{\mathcal{A}}^{\text{tamper}}(\lambda) \leq \text{Adv}_{\mathcal{B}_1, \text{Sig}}^{\text{euf-cma}}(\lambda) + \text{Adv}_{\mathcal{B}_2, \text{H}}^{\text{cr}}(\lambda),$$

for PPT reductions $\mathcal{B}_1, \mathcal{B}_2$.

Proof. Let $\mathcal{A}$ win, producing $(\vec{C}^*, \sigma_A^*)$ that verifies with $\vec{C}^* \notin Q_A$. Let $C^* = \text{enc}(\vec{C}^*)$, $h_1^* = \text{H}(\text{"v81"} \parallel \langle \text{RS} \rangle \parallel C^*)$, and $m_A^* = \text{"A:"} \parallel id^* \parallel \text{" :"} \parallel h_1^*$. Consider the message m_A^* and split into two mutually exclusive events.

Event E_1 : m_A^* was never returned/queried by the signing oracle. We build $\mathcal{B}_1$ against EUF-CMA. $\mathcal{B}_1$ receives pk^* and a signing oracle, sets $pk_A := pk^*$ (perfect simulation, as sk_A is used only via the ideal secure element), answers each A-signing query $\vec{C}$ by forming m_A and calling its own oracle, and logs the messages it queried. On $\mathcal{A}$'s output it returns (m_A^*, σ_A^*) . Under E_1 the message m_A^* is fresh (not in the query log), and it verifies by assumption, so this is a valid EUF-CMA forgery. Thus $\Pr[\mathcal{A} \text{ wins} \wedge E_1] \leq \text{Adv}_{\mathcal{B}_1, \text{Sig}}^{\text{euf-cma}}$.

Event E_2 : m_A^* equals some message m_A produced for an oracle query on $\vec{C} \in Q_A$. The message template $\text{"A:"} id \text{" :"} h_1$ is uniquely decodable (id a UUID with no $\text{" :"} \text{"}$, h_1 a fixed-length hex digest), so $m_A^* = m_A$ forces $id^* = id$ and, crucially, $h_1^* = h_1$, i.e.

$$\text{H}(\text{"v81"} \parallel \langle \text{RS} \rangle \parallel C^*) = \text{H}(\text{"v81"} \parallel \langle \text{RS} \rangle \parallel C).$$

Now $\vec{C}^* \neq \vec{C}$: they are distinct because $\vec{C} \in Q_A$ while $\vec{C}^* \notin Q_A$. By encoding injectivity on the admissible domain (Lemma 1, Remark 2), $\vec{C}^* \neq \vec{C} \Rightarrow C^* \neq C \Rightarrow \text{"v81"} \parallel \langle \text{RS} \rangle \parallel C^* \neq \text{"v81"} \parallel \langle \text{RS} \rangle \parallel C$. Hence the two hash inputs are distinct but their H-images coincide: a collision. $\mathcal{B}_2$ simulates the game for $\mathcal{A}$ exactly (it can, holding sk_A itself), and on event E_2 outputs the colliding pair $(\text{"v81"} \parallel \langle \text{RS} \rangle \parallel C^*, \text{"v81"} \parallel \langle \text{RS} \rangle \parallel C)$. Thus $\Pr[\mathcal{A} \text{ wins} \wedge E_2] \leq \text{Adv}_{\mathcal{B}_2, \text{H}}^{\text{cr}}$.

Since E_1, E_2 are exhaustive and disjoint, summing the two bounds gives the theorem. $\square$

Remark 5 (Coverage: what tamper evidence protects). The committed vector $\vec{C}$ comprises the document terms and A-identity fields listed in Section 5.1; every one of them is inside h_1 and hence under σ_A . The full 39-field *finalHash* record (Appendix B) additionally binds both signatures, both public keys, the biometric-backing flags, device-token hashes, and h_1 itself, so the same argument extends to post-signing tampering of the B-block once σ_B is present (via Theorem 2 for the chain and this theorem for the content). Fields *deliberately excluded* from the commitment—the non-deterministic rendering bytes of any display file (Section 5.6)—are, by design, *not* covered: the display artifact may be regenerated freely. This is a feature, not a gap, and should be stated as such.

7 Implementation

- **Primitives.** ECDSA P-256 (prime256v1), SHA-256; keys serialized Base64(X) : Base64(Y); signatures DER + Base64. Secure element: Apple Secure Enclave / Android StrongBox-Keystore;

software keys are *disabled* on the signing path (no silent software fallback). [TODO: confirm against the shipped build; the code removes the SW-fallback and returns an error instead.]

- **Attestation.** [TODO: App Attest / Play Integrity; state prod vs dev, cf. Remark 1.]
- **Transport.** Animated optical code with the LT-style fountain code of Section 5.5; alternative channels (NFC, BLE, ultrasound, file-embedded) supported.
- **Verifier.** A standalone script recomputes h_1 , h_2 , *finalHash* and checks σ_A, σ_B with no network. [TODO: Include a redacted transcript; you already have one showing the 39-field recomputation matching the stored final hash.]

[TODO: Optional but strong for ePrint: publish the verifier as an artifact and reference it. Reviewers/readers love a runnable checker.]

8 Limitations and Trade-offs

- **No CA is a trade, not a free win.** Key trust rests entirely on the attestation root and its soundness; a break there breaks provenance. State this.
- **Provenance is per-device, not per-natural-person** unless combined with an external identity step. [TODO: The product binds a national e-ID once at onboarding; that is an application-layer add-on, not part of the offline core — present it as such, not as a cryptographic guarantee of the core.]
- **Attestation strength is platform-dependent** (StrongBox vs TEE vs software) and dev-vs-prod matters (Remark 1).
- **Duress / malware / coerced biometric** are out of scope.
- **Fountain parameters** are an engineering choice; no optimality is claimed for the custom degree rule.

9 Conclusion

[TODO: 2–3 sentences. Restate the combination result and the offline, CA-less, attestation-bound property. No legal claims.]

A Worked Example (redacted)

[TODO: Insert the real (test) artifact values you already have: one C/h_1 , one m_A , one h_2 , one m_B , and the *finalHash* preimage field list, so a reader can reproduce every hash by hand. Redact personal data (RNOKPP hash is fine to show; raw PII is not).]

```
# Example (from the working verifier; values abbreviated):
# m_A = A:2d210b9c-...:64053e3605de4f9af0ce9ab335e611ce77ebb4d069da73153955caf65c4eeaf
# h_2 = SHA256(h_1 <RS> nickB <RS> avatarB) -> 66a7b60c176daa9033...
# m_B = B:2d210b9c-...:66a7b60c-...:<pkA>:<sigA>
# finalHash (39 fields, v81) -> 88430
    ddbc06a73373f95d8d034ea895eaac674409521ad66574555c2ab619213
```

B Full finalHash field order (v81)

The composite hash is $finalHash = H(\text{utf8}(F_0 \langle RS \rangle F_1 \langle RS \rangle \dots \langle RS \rangle F_{38} \langle RS \rangle))$, i.e. the separator $\langle RS \rangle$ (0x1E) is written *after every field including the last*. Each value F_i is passed through the escaping map f of Section 5.1 unless noted. Compound profile fields use the inner separator $\langle US \rangle$ (0x1F) between nickname and avatar index. The B-block (indices 28–35) is included *iff* $\sigma_B \neq \varepsilon$ (the second party has signed); indices 36–38 always follow. This ordering is reproduced verbatim by the reference verifier; the “Example” column shows the values of the worked artifact of Appendix A (long digests abbreviated).

#	Field	Fallback / rule	Example (abbrev.)
0	schema_prefix	literal v81	v81
1	contractId	—	2d210b9c-...-f04bb03a8d4d
2	contractType	—	
3	contractCategory	—	
4	title	—	icic
5	details	—	fncn
6	amount	(may be empty $\rightarrow$ 0 bytes)	(empty)
7	currency	—	UAH
8	contractLanguage	—	ru
9	jurisdiction	(empty $\rightarrow$ nojuris)	RFC 3161 TSA · ...
10	createdAt	ISO-8601 UTC	2026-07-06T19:32:46.000Z
11	tsaTime	ISO-8601 UTC	2026-07-06T19:32:46.000Z
12	timeSource	—	TSA:zeitstempel.dfn.de
13	publicKeyA	B64(X):B64(Y)	kIBx89P09u...
14	signatureA	DER+B64	MEYCIQDH7Q...
15	biometricBackedA	(empty $\rightarrow$ SW)	HW
16	deviceTokenHashA	$H(\text{"pakto_device_A:"} \parallel \tau_A)$; (empty $\rightarrow$ nodeviceA)	bc89576ec9...
17	geoHash	(empty $\rightarrow$ nogeo)	795ca83ffd628bb5
18	geoCoordinates	(empty $\rightarrow$ nocoords)	51.6515:11.5041
19	tgPhoneHashA	(empty $\rightarrow$ notg)	notg
20	emailA	(empty $\rightarrow$ noemail)	noemail
21	identityRnokppHashA	(empty $\rightarrow$ noidentity)	12ea976f6d...
22	identityQualifiedA	qualified/notqualified	qualified
23	tgPhoneHashB	(empty $\rightarrow$ notg)	notg
24	evidenceHashesJson	(empty $\rightarrow$ nophoto)	["bc118af04b..."]
25	photoHashA	(empty $\rightarrow$ nophoto)	e9bd9e8c67...
26	photoHashB	(empty $\rightarrow$ nophoto)	nophoto
27	tsaHashA	(empty $\rightarrow$ notsa)	f056c5b37f...
<i>B-block, indices 28–35, present iff $\sigma_B \neq \varepsilon$:</i>			
28	publicKeyB	B64(X):B64(Y)	VdE74RsWzz...
29	signatureB	DER+B64	MEQCIFPWrV...
30	biometricBackedB	(empty $\rightarrow$ SW)	HW
31	deviceTokenHashB	$H(\text{"pakto_device_B:"} \parallel \tau_B)$; (empty $\rightarrow$ nodeviceB)	e747d8d77c...
32	tsaTokenHashBExecutor	(empty $\rightarrow$ notsaB)	a0d6fb137d...
33	emailB	(empty $\rightarrow$ noemail)	noemail
34	identityRnokppHashB	(empty $\rightarrow$ noidentity)	noidentity
35	identityQualifiedB	qualified/notqualified	notqualified
<i>always present:</i>			
36	contentHash (= h_1)	(empty $\rightarrow$ nocontent)	64053e3605...
37	profileA	$nick_A \langle US \rangle avatar_A$; (empty $\rightarrow$ noprofile)	Dmytro $\langle US \rangle$ 1
38	profileB	$nick_B \langle US \rangle avatar_B$; (empty $\rightarrow$ noprofile)	$\langle US \rangle$ 0

Cross-check. For the artifact of Appendix A the above record hashes to

finalHash = 88430ddbc06a73373f95d8d034ea895eaac674409521ad66574555c2ab619213,

matching the stored value and the reference verifier’s recomputation. Note the two device-token fields (16, 31) are *context-prefixed* hashes of the full attestation tokens, not the tokens themselves; the full tokens travel in the transfer structure (Section 5.4), keeping the record compact while still binding provenance. **[TODO: Optionally include the raw preimage bytes (with literal 0x1E/0x1F shown as escapes) so a reader can reproduce the digest with a one-liner; you already have this from the verifier.]**

Acknowledgments

[TODO: optional / omit if anonymous.]

References

- [1] **[TODO: M. Luby, LT Codes, FOCS 2002.]**
- [2] **[TODO: A. Shokrollahi, Raptor Codes.]**
- [3] **[TODO: Al-Riyami, Paterson, Certificateless Public Key Cryptography, ASIACRYPT 2003.]**
- [4] **[TODO: Apple, App Attest / DeviceCheck documentation.]**
- [5] **[TODO: Google, Play Integrity API documentation.]**
- [6] **[TODO: FIPS 186 / SEC1 for ECDSA P-256.]**