Signature Chain: Byte-to-Byte and Peer-to-Peer

Darshan P.
Independent Researcher and Computer Programming Specialist
drshnp@numtrade.in

May 26, 2025

Abstract

We present an append-only signature scheme for PDF documents that embeds blockchain-like integrity without external infrastructure. Each signer appends a cryptographically linked block containing the original document's hash, a reference to the prior block, and their public-key signature. Single-pass verification ensures tampering or unauthorized modification is provably prevented under standard cryptographic assumptions.

1 Problem Statement

PDF signing today relies on third-party services (e.g., Abode Sign:)) that mediate requests, collect signatures, and return documents. This model cedes control to intermediaries, exposes users to metadata stripping, vendor lock-in, and centralized power over workflow. Adding visible marks or metadata after signing irrevocably changes the document hash, breaking cryptographic integrity, and entrusting platforms with the authority to alter or revoke signatures.

2 Chicken–Egg Paradox

In current practice, signing a PDF via email exchanges or government-issued DSC tokens conflates authentication and authorization: possession of a DSC pen drive or email credentials becomes the de facto power to sign. Embedding a signature trailer directly into the PDF modifies its byte-level hash, rendering the original hash irrecoverable. Ad hoc placeholder schemes attempt to preallocate space but depend on fragile workflows. By contrast, our method leverages the mathematical one-way property of SHA-256—computationally infeasible to invert—and builds an append-only chain of trailers whose digests link back to the pristine document. This transfers signing authority from intermediaries and hardware controls to pure cryptographic principles.

3 Notation

Let

$$D \;=\; \text{byte-sequence of the original PDF (no trailers)}, \qquad H \;=\; \mathrm{SHA256}(D).$$

The constant genesis value is $\mathbf{0}_{256} = 00\ldots00$ (32 bytes of 0).

Trailer framing (per block)

$$\texttt{\textbackslash n} \parallel \texttt{\#\#SIGSTART} \parallel \langle \text{8-byte len}_{\text{LE}} \rangle \parallel \texttt{\#\#} \parallel \Pi$$

where Π is the compact JSON payload.

4 Signing Algorithm (Signer i)

$$\begin{aligned} P_1 &:= \mathbf{0}_{256}, & P_i &:= \text{SHA256}(T_{i-1}) \quad (i \geq 2), && \text{(prev\_block)} \\ M_i &:= H \parallel P_i, && && \text{(message)} \\ d_i &:= \text{SHA256}(M_i), && && \text{(digest)} \\ s_i &:= \text{RSA\_Sign}_{\text{sk}_i}(d_i). && && \text{(signature)} \end{aligned}$$

Payload $\Pi_i = \{\texttt{ver:1},\ P_i,\ H,\ s_i,\ \text{metadata}\}$ Trailer $T_i = \big[\text{framing of } \Pi_i\big]$ Append T_i to the file.

1. Extract trailers $T_1, \ldots, T_n$ (oldest→newest) by sentinel search.
2. Compute $\hat{H} \leftarrow \text{SHA256}(D)$ and initialise $\hat{P}_1 \leftarrow \mathbf{0}_{256}$.
3. For $i = 1$ to n:

$$\hat{d}_i \leftarrow \text{SHA256}(\hat{H} \parallel \hat{P}_i), \quad \text{RSA\_Verify}_{\text{pk}_i}(s_i, \hat{d}_i) = \text{true}, \quad \hat{P}_{i+1} \leftarrow \text{SHA256}(T_i).$$

   Abort on the first failure.

If the loop terminates without error, every block and the underlying PDF are authentic.

5 Security Discussion

Transplant attack. Copying trailers $\{T_i\}$ onto a foreign PDF D' changes the computed hash $\text{SHA256}(D') \neq H$, so verification fails at $i = 1$. **Re-ordering.** Swapping T_j and T_{j+1} makes $P_{j+1} \neq \text{SHA256}(T_j)$, breaking the chain. **Mutation.** Editing any byte in D alters H; editing any byte in T_i alters $\text{SHA256}(T_i)$. Both invalidate signatures. Under standard collision resistance of SHA-256 and unforgeability of RSA, the scheme is tamper-evident and self-contained.

6 Implementation Results

```
$ sha256sum bitcoin.pdf
b1674191a88ec5cdd733e4240a81803105dc412d6c6708d53ab94fc248f4f553  bitcoin.pdf
$ python sign_pdf.py bitcoin.pdf Isaac_private_key.pem --name "Isaac Newton"
    ↪ --pubkey Isaac_public_key.pem
DONE Wrote bitcoin.pdf
$ python show_block.py bitcoin.pdf
LEDGER  1 block(s) in bitcoin.pdf

LEGIT Block 1 @ byte 184293•
```

```
   Signer     : Isaac Newton•
   Timestamp  : 2025-05-26T12:59:51Z•
   Signature  : RSA-PKCS1v15-SHA256•
   PDF Hash   : …ae63c72f34f3ebf4•
   Prev Hash  : …0000000000000000•
   Public Key (b64): …LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0t

$ python verify_sign_block.py bitcoin.pdf•
 Block 1 @   184293: ok

GOOD all good - SHA256(original PDF) =
   ↪ ae63c72f34f3ebf4f17134358f4716a5e35183cb5cc5797d0aff6bd58c9e4bba
$ python sign_pdf.py bitcoin.pdf Nikola_private_key.pem --name "Nikola Tesla"
   ↪ --pubkey Isaac_public_key.pem
FALSE pubkey  privkey
$ python sign_pdf.py bitcoin.pdf Nikola_private_key.pem --name "Nikola Tesla"
   ↪ --pubkey Nikola_public_key.pem
DONE Wrote bitcoin.pdf
$ python show_block.py bitcoin.pdf
LEDGER  2 block(s) in bitcoin.pdf

LEGIT Block 1 @ byte 184293•
   Signer     : Isaac Newton•
   Timestamp  : 2025-05-26T12:59:51Z•
   Signature  : RSA-PKCS1v15-SHA256•
   PDF Hash   : …ae63c72f34f3ebf4•
   Prev Hash  : …0000000000000000•
   Public Key (b64): …LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0t

LEGIT Block 2 @ byte 185718•
   Signer     : Nikola Tesla•
   Timestamp  : 2025-05-26T13:00:48Z•
   Signature  : RSA-PKCS1v15-SHA256•
   PDF Hash   : …ae63c72f34f3ebf4•
   Prev Hash  : 811…e47433d004045•
   Public Key (b64): …LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0t

$ python verify_sign_block.py
usage: verify_sign_block.py file.pdf
$ python verify_sign_block.py bitcoin.pdf•
 Block 1 @   184293: ok•
 Block 2 @   185718: ok

GOOD all good - SHA256(original PDF) =
   ↪ ae63c72f34f3ebf4f17134358f4716a5e35183cb5cc5797d0aff6bd58c9e4bba
$ python extract_pristine.py bitcoin.pdf | sha256sum
121cd1634a55388e916587848728db1a9f0cc6f3d7c11a96b967d08caf91aa46  -
$ python extract_unsigned.py --strip-newline bitcoin.pdf | sha256sum
b1674191a88ec5cdd733e4240a81803105dc412d6c6708d53ab94fc248f4f553  -
$ sha256sum bitcoin.pdf
5feaa998d1a261efe32c31fd82cf4869b19fdc33b142cb1ab99cecfa1542259c  bitcoin.pdf
$ sha256sum ../testing_pdfs/bitcoin.pdf
b1674191a88ec5cdd733e4240a81803105dc412d6c6708d53ab94fc248f4f553  ../
   ↪ testing_pdfs/bitcoin.pdf
```

7 Conclusion

We introduce a peer-to-peer, byte-level signature chain embedded directly in the PDF file, preserving the original SHA-256 hash and eliminating reliance on external signing services. Each trailer appends a cryptographic linkage to the prior state, forming an append-only log that allows multiple participants to sign in sequence. By rooting trust in mathematical primitives rather than third-party conventions, authority over the document remains with its users. Although most PDF readers ignore trailing data without issue, some processing tools may strip or modify appended bytes—implementations should ensure trailers reside in truly ignored zones. Overall, this method provides a provably tamper-evident integrity layer that integrates seamlessly with existing PDF workflows.

References

- SHA-2 (SHA-256) — *Wikipedia.*
  `https://en.wikipedia.org/wiki/SHA-2`
- OpenSSL dgst Documentation — *OpenSSL Official Documentation.*
  `https://docs.openssl.org/master/man1/openssl-dgst/#notes`