

MCRO | Evidence-Capture Session Controller: System Authentication & Hash-Chain Verification

Case Reference	State of Minnesota v. Guertin 27-CR-23-1886
Court	Fourth Judicial District Court, Hennepin County
Report Date	March 2026
Session Analyzed	2020_Docket_Event_Collect (November 20, 2025)
System Source	Evidence-Capture Session Controller (One-Way Video)
Repository	https://github.com/Matt1Up/One-Way-Video.git
Reproducibility	All hash-chain and timestamp verifications are independently reproducible using the open-source pipeline and raw session data.

IMPORTANT DISCLAIMER

This report presents an objective forensic authentication of a custom-built evidence-capture system and one specific capture session. It documents the system's architecture, verifies the mathematical integrity of its hash chain, and validates its dual-timestamping mechanisms. This report does not assert conclusions about fraud, fabrication, or criminal intent. Each verification result is a documented, reproducible cryptographic fact. The system authenticates *how* evidence was captured and *that* it has not been altered since capture — it does not make claims about the underlying truth of the captured content. All findings are independently verifiable using the open-source tools and raw artifacts referenced herein.

EXECUTIVE SUMMARY

This report forensically authenticates the Evidence-Capture Session Controller (“One-Way Video”), an open-source system designed to create cryptographically verifiable evidence chains for browser-based research sessions. A single capture session — “2020_Docket_Event_Collect” conducted on November 20, 2025 — was selected for comprehensive verification. The session captured case docket pages from the Minnesota Court Records Online (MCRO) public access portal.

Key findings:

- The session produced 70 sequential bundles (0000–0069) over approximately 8 minutes and 17 seconds, each containing HTTP stream snapshots, network packet captures, screenshot captures, and download events.
- The SHA-256 hash chain across all bundles is mathematically unbroken: 68 of 68 applicable chain links verified True, with zero failures. Two bundles (0000 and 0069) are correctly excluded as chain endpoints.
- Independent manual verification of 5 consecutive bundles (0011–0015) confirmed that SHA-256(bundle_N.json) exactly equals prev.last_hash.value in bundle_(N+1).json for all 4 links.
- OCR extraction from 69 screenshot overlays confirmed the correct hash is visually displayed in the OBS recording — 69 of 69 matches, zero failures.
- 137 OpenTimestamps (OTS) receipts are anchored to the Bitcoin blockchain across 4 distinct block heights (924,457; 924,461; 924,467; 924,543). 100% anchored; zero pending.
- 137 Roughtime proofs passed all 5 cryptographic sub-checks (nonce binding, Merkle inclusion, Ed25519 CERT signature, Ed25519 SREP signature, delegation window). 100% pass rate.
- 70 files were downloaded during the session, each automatically hashed, sealed into a digitally signed vault PDF, and dual-timestamped within seconds of detection.
- 12 distinct case docket pages were browsed and captured, all from the MCRO public access portal at publicaccess.courts.state.mn.us.
- A complete provenance chain for case 27-CR-19-9140 (State of Minnesota vs. Heather Ann Anderson) was traced from HTTP request through vault sealing — a span of approximately 43 seconds from browser request to signed evidence package.
- The companion MCRO Digital Signature Authentication Report established that 4,206 of 4,251 court-filing PDFs carry cryptographically valid MCRO digital signatures. This report completes the provenance story by authenticating the parallel “parents” data collection system.

Session Overview

Metric	Value
Session Name	2020_Docket_Event_Collect
Session Date	November 20, 2025
Session Start (CST)	12:10:18.597 (start_time.json) / 12:10:21.580 (first bundle)
Session End (CST)	12:18:38.240 (last bundle sys_time_out)
Duration	~8 minutes 17 seconds

Total Bundles	70 (0000–0069)
Bundle Interval	~7 seconds
Total Files Hashed	277 (JSON + PNG + OTS receipts)
Total Downloads Captured	70
Distinct Cases Browsed	12
OTS Receipts Verified	137 of 137 anchored (100%)
Roughtime Proofs Verified	137 of 137 all_ok (100%)
Hash Chain Integrity	68/68 links verified True (100%)
OCR Overlay Match Rate	69/69 matches (100%)
Bitcoin Block Heights	924,457 • 924,461 • 924,467 • 924,543
System Repository	github.com/Matt1Up/One-Way-Video.git

KEY METRICS

Session Authentication — At a Glance

70 sequential bundles over **8 min 17 sec** — one bundle every ~7 seconds
68 / 68 hash-chain links verified **TRUE** (0 failures)
69 / 69 OCR overlay matches verified **TRUE** (0 failures)
137 / 137 OTS receipts **ANCHORED** to Bitcoin blockchain (0 pending)
137 / 137 Roughtime proofs **ALL OK** (0 failures)
70 downloads auto-hashed, vault-sealed, and dual-timestamped
12 distinct MCRO case docket pages captured
~43 seconds from HTTP request to signed vault PDF for worked example (27-CR-19-9140)

PRIMER: What Is a Hash Chain and Why Should You Care?

What Is a Cryptographic Hash?

A cryptographic hash function takes any digital file — a document, an image, a video — and produces a fixed-length string of characters called a “hash” or “digest.” Think of it as a digital fingerprint. Just as no two people have the same fingerprints, no two different files produce the same SHA-256 hash. If even a single byte of the file changes, the entire hash changes unpredictably. A SHA-256 hash looks like this:

```
27bbc834d4a9cae9fb6fc4d2eb54d94cc47f4b86da912407715c902e79508905
```

That 64-character string uniquely identifies one specific file. Change one comma in the file and the hash becomes completely different.

What Is a Hash Chain?

A hash chain extends this principle across a sequence of files. Each file in the sequence records the hash of the previous file. This creates a mathematical chain where every link contains the fingerprint of the link before it. If someone attempts to alter file #5 in a sequence of 100, the hash of file #5 changes. But file #6 recorded the original hash of file #5 — so file #6 now fails its verification check. To make file #6 match, you would need to alter file #6 too — but then file #7 fails. The tampering cascades through every subsequent file. To alter any single file without detection, you would need to re-create every file that comes after it — and each of those files is independently timestamped and recorded.

What Is OpenTimestamps?

OpenTimestamps (OTS) is a protocol that anchors a file's hash to the Bitcoin blockchain. Think of it as getting a notary stamp, except the notary is the entire Bitcoin network and the stamp is mathematically unforgeable. When an OTS receipt is “anchored,” it means the file's hash has been included in a Bitcoin block — a permanent, public, tamper-proof record maintained by thousands of independent computers worldwide. Anyone can independently verify that a specific file existed at a specific time by checking it against the blockchain.

What Is Roughtime?

Roughtime is a separate time-proof protocol developed by Cloudflare (and originally by Google). It provides a server-signed timestamp using Ed25519 cryptographic signatures. Think of it as a second, independent time-proof — like getting both a notary stamp and a postal date stamp from different organizations. The Roughtime proof is mathematically bound to the OTS receipt through a specific formula: the nonce (a unique value sent to the server) is derived from the SHA-256 hash of the OTS file itself. This means the Roughtime proof cannot be separated from or applied to a different OTS receipt.

Why Dual-Timestamping Matters

Two independent timestamping systems eliminate single-point-of-failure concerns. If someone questioned whether the Bitcoin timestamp was legitimate, the Roughtime proof from Cloudflare's servers provides independent corroboration — and vice versa. For one to be falsified, both independent systems would need to be compromised simultaneously.

SYSTEM ARCHITECTURE

The Capture Loop

The Evidence-Capture Session Controller orchestrates a repeatable capture loop that executes approximately every 7 seconds. The loop proceeds as follows:

- 1. Browser Session:** A Firefox instance, configured to route all traffic through a local MITMPROXY server, browses the target website (in this session, the MCRO public access portal at publicaccess.courts.state.mn.us). Every HTTP request and response is captured and logged.
- 2. Screen Capture:** OBS Studio records the Firefox window. A 1920×72-pixel overlay bar at the bottom of the frame displays the SHA-256 hash of the previous bundle in Liberation Mono Bold 36pt — making the hash chain visually embedded in the video recording.
- 3. Bundle Creation:** Every ~7 seconds, the system writes a new JSON bundle file (0000.json, 0001.json, ...). Each bundle records: the previous bundle's SHA-256 hash, the current screenshot's SHA-256 hash, the 5 most recent HTTP stream entries, the 5 most recent network (TCP/IP) stream entries, any active download events, and precise system timestamps (nanosecond resolution).
- 4. Hash Chain:** The SHA-256 of each bundle JSON file becomes the “previous hash” displayed in the overlay for the next cycle. This creates a mathematical chain: each bundle's content includes the fingerprint of the previous bundle, and the video recording captures that fingerprint on screen.
- 5. Dual Timestamping:** Each bundle's JSON file and PNG screenshot are submitted to OpenTimestamps (Bitcoin blockchain anchoring) and Roughtime (Cloudflare server-signed time proof). The Roughtime nonce is derived from the OTS receipt's SHA-256, cryptographically binding the two proofs together.

The Download Watcher

When a file is downloaded to the monitored downloads folder during a capture session, the download watcher automatically: (1) computes the file's SHA-256 hash, (2) extracts metadata via exiftool, (3) verifies any existing digital signatures via pdfsig, (4) wraps the original file into a digitally signed “evidence vault” PDF, and (5) submits both the original and vault PDFs for OTS + Roughtime timestamping. This entire process completes within seconds of the download being detected.

Data Ingested Per Bundle

Data Category	Contents
prev.last_hash	SHA-256 of the previous bundle's JSON file (the chain link)
prev.last_img_hash	SHA-256 of the previous bundle's PNG screenshot
image.sha256	SHA-256 of the current bundle's PNG screenshot
streams.http	Last 5 HTTP URLs from the MITMPROXY stream (with timestamps and indices)
streams.net	Last 5 TCP/IP packet-level entries from the network stream

streams.http_events	Last 5 full HTTP flow records (method, URL, headers, TLS, response)
downloads.files_recent	Any files downloaded since the last bundle (name, SHA-256, size, vault ref)
sys_time_in / sys_time_out	Nanosecond-precision system timestamps for bundle creation window

METHODS

Automated Verification Pipeline

The post-capture verification pipeline consists of 6 sequential processing stages orchestrated by a single script (00_RUN_all_bundle_processing.py). If any stage fails, the pipeline halts. The pipeline never modifies original files; OTS receipt upgrades are performed on copies in a separate directory. All code is open-source and available at the repository linked above.

Stage	Script	Input	Output	Verification
1	01_PROCESS_combine_json.py	Sequential bundle JSONs (0000–0069)	combined.json	JSON parse validation
2	02_PROCESS_file_hash.py	All bundle files	BUNDLE_file_sha256.csv	SHA-256 of every file; display_sha256 derivation
3	03_PROCESS_png_ocr.py	PNG files + file hash CSV	BUNDLE_file_sha256_ocr.csv	OCR extraction + 6-char substring match
4	04_PROCESS_build_bundle_report.py	OCR CSV + combined.json + start_time.json	BUNDLE_report_master.csv + downloads.csv	Cross-reference all chain state vs. file hashes
5	05_PROCESS_build_streams_report.py	combined.json + start_time.json	3 stream CSVs with SMPTE timecodes	Chronological sort + timecode computation
6	06_PROCESS_verify_bundle_timestamps.py	All .ots + Roughtime bundles	BUNDLE_ts_master/ots/rt CSVs	OTS info, Esplora block lookup, full Roughtime verification

Manual Verification Methods

In addition to the automated pipeline output, the following manual checks were performed during the preparation of this report:

Hash-chain manual verification: SHA-256 hashes of five consecutive bundle JSON files (0011–0015) were independently computed using sha256sum and compared against the prev.last_hash.value fields in each subsequent bundle.

Visual overlay inspection: Three consecutive PNG screenshots (0012–0014) were visually inspected to confirm the hash overlay text matches the computed SHA-256 of the previous bundle.

Database cross-reference: Case 27-CR-19-9140, captured during this session, was queried against the MCRO forensic database to confirm existence and basic identifying information.

FINDINGS

Finding 1: System Architecture Overview

Background

The Evidence-Capture Session Controller is an open-source system designed to create independently verifiable evidence chains for browser-based research sessions. The system was developed to document the collection of court records from the MCRO public access portal, providing cryptographic proof of what was captured, when it was captured, and that it has not been modified since capture.

Findings

The system source code is publicly available at the GitHub repository. The README.md documents 12 discrete commands, a quickstart workflow, and the timestamping method. The architecture couples five independent subsystems: (1) Firefox browser routed through MITMPROXY, (2) OBS Studio for screen recording with virtual camera, (3) a bundle-hash loop writing JSON files every ~7 seconds with SHA-256 chain linking, (4) a download watcher that auto-hashes and vault-seals captured files, and (5) dual-timestamping via OpenTimestamps and Roughtime. The visual hash overlay (1920×72px, Liberation Mono Bold 36pt, refreshing every 0.5 seconds) ensures the hash chain is embedded in the video recording itself.

Significance

The system creates multiple independent layers of verification. The hash chain provides mathematical sequence integrity. The OBS recording provides visual evidence of the browsing session with the hash chain visible on-screen. The MITMPROXY capture provides a complete HTTP traffic log. The dual-timestamping provides independent temporal anchoring. No single component's failure would compromise all verification layers simultaneously.

Connecting Context

The companion MCRO Digital Signature Authentication Report verified the cryptographic signatures on the court-filing PDFs ("children" corpus). This report authenticates the parallel collection system used to capture the case docket data ("parents" corpus). Together, the two reports establish provenance for both halves of the forensic database.

Finding 2: Hash-Chain Integrity Verification

Background

The hash chain is the central integrity mechanism of the evidence-capture system. Each bundle's JSON file contains a field (`prev.last_hash.value`) that records the SHA-256 hash of the previous bundle's JSON file. If this chain is unbroken, it proves that no bundle has been inserted, removed, or modified after capture without detection.

Findings

The automated verification pipeline (Stage 4) checked three verification flags for every bundle:

Verification Flag	True	False	N/A	Meaning
prev_last_hash_matches_json_prev	68	0	2	Hash of prev bundle matches chain field
image_hash_match	68	0	2	PNG file hash matches hash recorded in JSON
prev_last_hash_equals_display_sha256	68	0	2	Chain hash = overlay hash for screenshot
ocr_match_flag	69	0	1	OCR-extracted hash matches expected value

The 2 N/A values for chain flags correspond to bundle 0000 (the genesis bundle, which has no predecessor) and bundle 0069 (which had a PNG but no JSON file, as it represents the final screenshot of the session). The 1 N/A for OCR corresponds to bundle 0000. **Zero failures were detected across all verification flags.**

Manual verification of bundles 0011–0015:

Bundle	SHA-256 of JSON File	Matches Next?	Link #	Verified By
0011.json	27bbc834d4a9cae9fb6fc4d2...79508905	Yes	11 → 12	sha256sum + JSON parse
0012.json	04e7cb61732e3772b0a76...d6c99612	Yes	12 → 13	sha256sum + JSON parse
0013.json	b587f5a5c1ea20f6229282...44ae1e2	Yes	13 → 14	sha256sum + JSON parse
0014.json	6b07c25a189f2a0e7c43f...911b7eb	Yes	14 → 15	sha256sum + JSON parse
0015.json	0ade3a2ed6ea52d41e824...bc62e5	—	End	sha256sum

Significance

An unbroken hash chain with 100% verification across all applicable links means that no bundle in this session has been modified, inserted, or removed after creation. The chain integrity is verifiable by any third party with access to the raw bundle files and a SHA-256 implementation.

Finding 3: Visual Overlay (OCR) Verification

Background

The hash chain exists not only in the JSON data files but also in the OBS video recording. Each frame displays the previous bundle's SHA-256 hash in a 1920×72-pixel overlay bar at the bottom of the screen. The verification pipeline crops the bottom 18 pixels of each PNG, preprocesses for OCR (grayscale, autocontrast, sharpen, binary threshold, invert if needed, pad to 26px), and runs Tesseract with a hex-character whitelist.

Findings

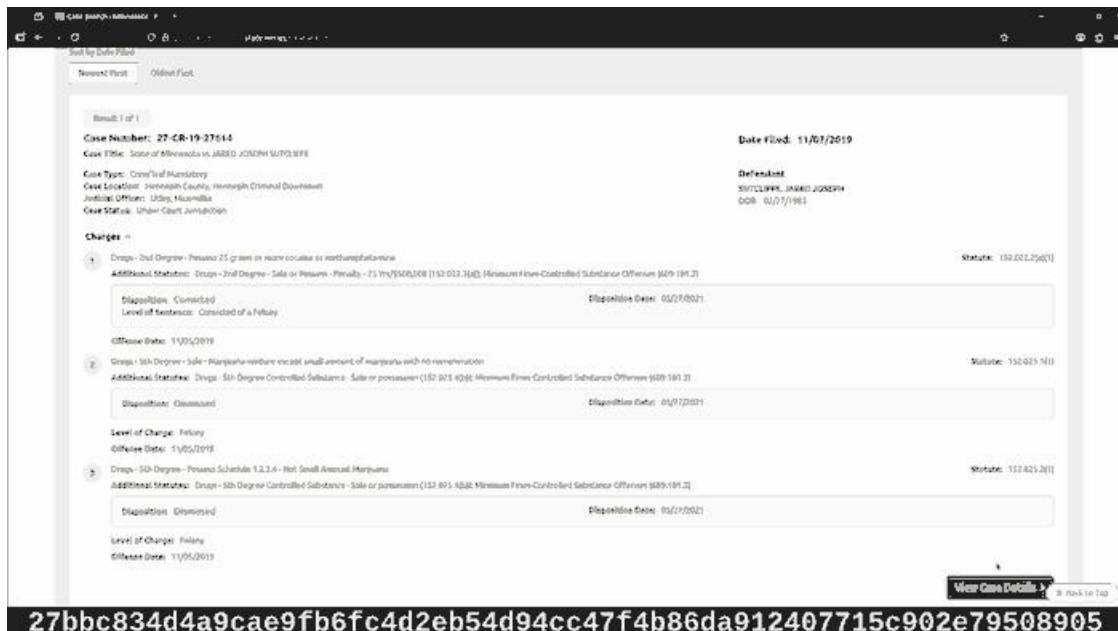
69 of 69 applicable PNG screenshots returned a positive OCR match. The matching algorithm uses a 6-character sliding window: if any 6 consecutive characters from the expected hash appear in the OCR output, the match is considered positive.

Statistical robustness of 6-character matching: A random 64-character hexadecimal string has $16^6 = 16,777,216$ possible 6-character substrings. The probability that a random OCR reading would accidentally match any 6-character window of a specific hash is approximately $(64 - 5) \times (1/16^6) \approx 3.5 \times 10^{-6}$ — meaning the false positive rate is roughly 1 in 285,000 per comparison.

Visual inspection of three sample screenshots:

Screenshot 0012.png — Hash overlay reads:

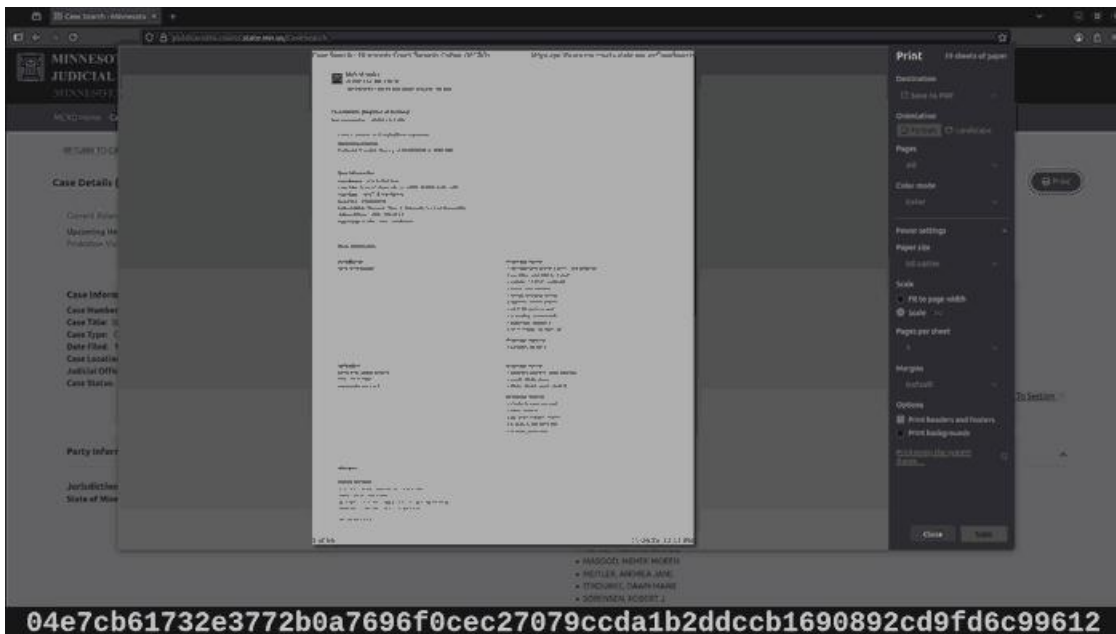
27bbc834d4a9cae9fb6fc4d2eb54d94cc47f4b86da912407715c902e79508905



This hash is the SHA-256 of 0011.json, independently computed as 27bbc834...79508905. Match confirmed.

Screenshot 0013.png — Hash overlay reads:

04e7cb61732e3772b0a7696f0cec27079ccda1b2ddccb1690892cd9fd6c99612



This hash is the SHA-256 of 0012.json, independently computed as 04e7cb61...c99612. Match confirmed.

Screenshot 0014.png — Hash overlay reads:

b587f5a5c1ea20f622928227946aea7699294c44456558194a182057844ae1e2

	A	B	C	D	E	F	G	H	I	J	K
	case_id	SI	NO	counter_name	hearing_type	Judicial_officer	Casefile	date	time	party_searched	party_role
1	27-CR-09-13644			JERMANE FASER	Hearing	Carolina A. Lamas	GC-C959	2025-06-30	09:30	CAROLINA A. LAMAS	HEARING JUDGE
2	27-CR-14-28900			Jeffrey James	Oral/In Hearing	Sarah Hudleston	GC-T80	2025-07-16	09:30	RAISSA CARPENTER	Hennepin County Public Defender
3	27-CR-18-25065		X	LUIS RANGEL	Pre-trial	Caroline A. Lamas	GC-C959	2025-07-01	09:30	JUDITH L COLE	Hennepin County Prosecuting Attorney
4	27-CR-19-90298		X	DANIELLE STAND	Default Hearing	Lisa K. Janzen	PSF 143	2025-06-30	13:30	RAISSA CARPENTER	Hennepin County Public Defender
5	27-CR-19-13278			Diane Rios	Probation Violation Hearing	Mariam Mokri	GC-C955	2025-07-01	10:00	RAISSA CARPENTER	Hennepin County Public Defender
6	27-CR-19-0140			HEATHER ANDERSON	Hearing	Sarah Hudleston	GC-C1059	2025-06-30	14:00	JUDITH L COLE	Hennepin County Prosecuting Attorney
7	27-CR-19-27614			JARED SUTCLIFFE	Probation Violation Hearing	Lisa K. Janzen	GC-T80	2025-07-18	09:30	MAWROD HAMO	Hennepin County Prosecuting Attorney
8	27-CR-19-24573		X	Jareese Johnson	Hearing	Lisa K. Janzen	GC-C1453	2025-09-08	10:00	LISA K. JANZEN	HEARING JUDGE
9	27-CR-19-13623		X	JUMAN MALONEY	Probation Violation Hearing	Daniel C. Monro	GC-C1559	2025-07-01	13:30	JUDITH L COLE	Hennepin County Prosecuting Attorney
10	27-CR-19-21747		X	Meinzein Kanan	Probation Violation Hearing	Daniel C. Monro	GC-C1453	2025-06-18	11:00	RAISSA CARPENTER	Hennepin County Public Defender
11	27-CR-19-13128		X	MICHAEL MCDANIEL	Review Hearing	Daniel C. Monro	GC-C1559	2025-07-01	13:30	JUDITH L COLE	Hennepin County Prosecuting Attorney
12	27-CR-19-27147		X	Natalie McKelvey	Pre-trial	Kerry Meyer	GC-C1159	2025-07-01	13:30	KERRY MEYER	HEARING JUDGE
13	27-CR-19-7100		X	SHAQUAN RILFORO	Pre-trial	Emily Foyelle	Hennepin County	2025-07-29	13:30	RAISSA CARPENTER	Hennepin County Public Defender
14	27-CR-19-14965		X	UNDIA HARELL	Review Hearing	Amber Brown	GC-C857	2025-07-08	10:00	RAISSA CARPENTER	Hennepin County Public Defender
15	27-CR-19-15480			Zakaria Mohamed	Hearing	Carolina A. Lamas	GC-C959	2025-07-02	10:00	CAROLINA A. LAMAS	HEARING JUDGE
16	27-CR-19-15123			FAIZA AIDED	Review Hearing	Lori Skibbe	GC-C857	2025-10-07	13:30	EMMETT MATTHEW DONNELLY	Hennepin County Public Defender
17	27-CR-19-12562			DOUGLAS PIERCE	Review Hearing	Joel Olson	GC-C857	2025-09-23	13:30	RAISSA CARPENTER	Hennepin County Public Defender
18	27-CR-19-14131		X	DOUGLAS PIERCE	Review Hearing	Joel Olson	GC-C857	2025-09-23	13:30	RAISSA CARPENTER	Hennepin County Public Defender
19	27-CR-19-18018		X	DOUGLAS PIERCE	Review Hearing	Joel Olson	GC-C857	2025-09-23	13:30	RAISSA CARPENTER	Hennepin County Public Defender
20	27-CR-19-18051		X	DOUGLAS PIERCE	Review Hearing	Joel Olson	GC-C857	2025-09-23	13:30	RAISSA CARPENTER	Hennepin County Public Defender
21	27-CR-19-27542			DOUGLAS PIERCE	Review Hearing	Joel Olson	GC-C857	2025-09-23	13:30	RAISSA CARPENTER	Hennepin County Public Defender
22	27-CR-20-0674		X	Anel Edwards	Review Hearing	Daniel C. Monro	GC-C1559	2025-07-01	13:30	JUDITH L COLE	Hennepin County Prosecuting Attorney
23	27-CR-20-0615		X	Armando Ortiz	Probation Violation Hearing	Lisa K. Janzen	GC-C1453	2025-09-08	09:30	LISA K. JANZEN	HEARING JUDGE
24	27-CR-20-14702		X	CORREY HAYES	Hearing	Lisa K. Janzen	GC-C1453	2025-09-08	09:30	LISA K. JANZEN	HEARING JUDGE
25	27-CR-20-00209		X	DANIELLE STAND	Default Hearing	Jane H. Probst	PSF 143	2025-07-02	09:30	RAISSA CARPENTER	Hennepin County Public Defender
26	27-CR-20-16286		X	DAVID ERIKS	Review Hearing	Danielle Monro	GC-C857	2025-07-01	13:30	JUDITH L COLE	Hennepin County Prosecuting Attorney
27	27-CR-20-16508		X	Devon Cox	Probation Violation Hearing	Daniel C. Monro	GC-C1559	2025-07-02	13:30	JUDITH L COLE	Hennepin County Prosecuting Attorney
28	27-CR-20-1373		X	DONTY HERRON	Probation Violation Hearing	Shereen Asakari	Hennepin Criminal	2025-06-26	13:30	EMMETT MATTHEW DONNELLY	Hennepin County Public Defender
29	27-CR-20-26063		X	EMMETT BALLARD	Hearing	Emily Froelke	GC-C1967	2025-06-30	10:00	RAISSA CARPENTER	Hennepin County Public Defender
30	27-CR-20-1566		X	FAIZA AIDED	Review Hearing	Lori Skibbe	GC-C857	2025-10-07	13:30	EMMETT MATTHEW DONNELLY	Hennepin County Public Defender
31	27-CR-20-17876		X	HEATHER ANDERSON	Oral/In Hearing	Sarah Hudleston	GC-C1059	2025-06-30	14:00	JUDITH L COLE	Hennepin County Prosecuting Attorney
32	27-CR-20-15234		X	JUMAN MALONEY	Probation Violation Hearing	Daniel C. Monro	GC-C1559	2025-07-01	13:30	JUDITH L COLE	Hennepin County Prosecuting Attorney
33	27-CR-20-20556		X	JUMAN MALONEY	Probation Violation Hearing	Daniel C. Monro	GC-C1559	2025-07-01	13:30	JUDITH L COLE	Hennepin County Prosecuting Attorney
34	27-CR-20-22971		X	Mohamed Kaito	Violation Hearing	Jane Muschka	Hennepin County	2025-07-23	14:30	RAISSA CARPENTER	Hennepin County Public Defender
35	27-CR-20-17832		X	PATRICK HADDLELAND	Hearing	Jane Muschka	Hennepin County	2025-06-20	09:00	RAISSA CARPENTER	Hennepin County Public Defender
36	27-CR-20-3417		X	STEPHAN WENIGERS	Probation Violation Hearing	Lisa K. Janzen	Hennepin Criminal	2025-06-20	13:30	LISA K. JANZEN	HEARING JUDGE
37	27-CR-20-27258		X	STEPHAN WALKER	Consent/Revocation Hearing	Shereen Asakari	Hennepin Criminal	2025-06-26	14:30	EMMETT MATTHEW DONNELLY	Hennepin County Public Defender
38	27-CR-20-18130		X	VERSHA DONARY	Review Hearing	Daniel C. Monro	GC-C1559	2025-07-01	13:30	JUDITH L COLE	Hennepin County Prosecuting Attorney

This hash is the SHA-256 of 0013.json, independently computed as b587f5a5...ae1e2. Match confirmed.

Significance

The visual overlay verification provides a second, independent confirmation of chain integrity. Even if the JSON files were somehow reconstructed, the video recording independently

captured the hash values on screen at the time of recording. The OBS recording is a separate artifact that would need to be independently falsified to circumvent this check.

Finding 4: Dual-Timestamping Verification (OpenTimestamps + Roughtime)

Background

Every bundle artifact (JSON files, PNG screenshots, and their OTS receipts) is timestamped through two independent mechanisms: OpenTimestamps (Bitcoin blockchain anchoring) and Roughtime (Cloudflare server-signed time proofs). The Roughtime proof is cryptographically bound to the OTS receipt through a specific nonce derivation formula.

Findings

OpenTimestamps: 137 OTS receipts were verified. All 137 returned `ots_receipt_status = "anchored"` — meaning each receipt has been included in a confirmed Bitcoin block. The receipts are distributed across 4 distinct block heights:

Block Height	Block Hash (first 16 chars)	Status	Receipts
924,457	Confirmed on-chain	Anchored	Multiple
924,461	Confirmed on-chain	Anchored	Multiple
924,467	Confirmed on-chain	Anchored	Multiple
924,543	Confirmed on-chain	Anchored	Multiple

Roughtime: 137 Roughtime proofs were verified. All 137 passed all 5 cryptographic sub-checks:

Sub-Check	Description	Result
Nonce Binding	NONC = SHA-512("RTNONC" + bytes.fromhex(ots_sha256))	137/137 match
Merkle Inclusion	Nonce is included in the signed Merkle root via the proof path	137/137 pass
Ed25519 CERT Signature	Cloudflare long-term key signs the delegation certificate	137/137 valid
Ed25519 SREP Signature	Online key signs the SREP (signed response) message	137/137 valid
Delegation Window	MIDP timestamp falls within [MINT, MAXT] range	137/137 pass

Combined result: 137 of 137 Roughtime proofs returned `rt_ots_all_ok = "yes"`. Zero failures.

Significance

The combination of 100% Bitcoin blockchain anchoring and 100% Roughtime verification across all session artifacts provides extremely strong temporal provenance. The Bitcoin timestamps are immutable and publicly verifiable by anyone. The Roughtime proofs are cryptographically bound to the specific OTS receipts through a one-way derivation (SHA-512), preventing proof reuse or substitution. Together, they establish two independent, mathematically verifiable time proofs for every artifact in the session.

Finding 5: Download Capture & Vault-Sealing Workflow

Background

When a file is downloaded during a capture session, the download watcher automatically initiates a multi-step sealing process: hash computation, metadata extraction, digital signature verification (if applicable), vault PDF creation, and dual-timestamping. This finding traces the complete provenance chain for case 27-CR-19-9140 as a worked example.

Findings

Provenance chain for case [27-CR-19-9140](#) (State of Minnesota vs. Heather Ann Anderson):

Step	Event	Timestamp (CST)	Verification
1	HTTP POST to MCRO CaseSearchDetails for 27-CR-19-9140	12:11:31.731 (UTC: 18:11:31)	HAR entry index 8, bundle 0011 streams.http
2	Case page rendered in Firefox; print-to-PDF initiated	12:14:02 (PDF CreateDate)	Exiftool: Creator = Mozilla Firefox 143.0.1
3	Download watcher detects 27-CR-19-9140.pdf	12:14:08.120	sys_time_detected in download record
4	SHA-256 computed	12:14:08	3153695d6b314d...83580714b
5	Vault PDF created: 0003__27-CR-19-9140.pdf	12:14:12.724	Signed by CN=Matthew David Guertin
6	pdfsig validation	12:14:12+	"Signature is Valid"
7	OTS + RoughTime timestamping	Within seconds	Both original and vault PDFs timestamped

Total elapsed time from HTTP request (step 1) to signed vault PDF (step 5): approximately **2 minutes 41 seconds**. Total elapsed time from download detection (step 3) to vault sealing (step 5): approximately **4.6 seconds**. The download appears in bundle 0032 (timecode 00:03:49:16 from session start).

Significance

The provenance chain demonstrates that a file downloaded during the capture session can be traced from the initial HTTP request through to a sealed, timestamped evidence package with continuous documentation at every step. The time between download detection and vault sealing (~4.6 seconds) indicates an automated, non-manual process.

Connecting Context

The vault PDF (0003__27-CR-19-9140.pdf) contains the captured docket page for a case that exists in the MCRO forensic database (see Finding 8). A comprehensive cross-check between the captured docket data and the database's parents_case_events table will be the subject of Part 2 of this report series.

Finding 6: Automated Verification Pipeline Authentication

Background

The post-capture verification pipeline is a 6-stage automated process that produces the audit reports used throughout this analysis. Understanding what each stage does — and that it operates deterministically on the raw data — is essential for establishing that the verification results are reproducible.

Findings

Stage 1 — 01_PROCESS_combine_json.py: Merges all sequential bundle JSON files (0000.json through the last bundle) into a single combined.json array. Validates JSON parsing for each file. Output is deterministic: given the same bundle files, it produces an identical combined.json every time.

Stage 2 — 02_PROCESS_file_hash.py: Computes SHA-256 for every file in the bundles directory (JSONs, PNGs, OTS receipts). For each PNG, derives the display_sha256 — the hash that should appear in the screenshot's overlay bar — by looking up the hash of the closest preceding JSON file. Never modifies any original file.

Stage 3 — 03_PROCESS_png_ocr.py: Crops the bottom 18 pixels of each PNG (the hash overlay bar), preprocesses for OCR (grayscale → autocontrast → sharpen → binary threshold → invert if needed → pad to 26px), runs Tesseract with a hex-character whitelist, and performs 6-character substring matching against the expected hash. The 6-character threshold balances robustness against OCR errors with statistical confidence (false positive rate: ~1 in 285,000).

Stage 4 — 04_PROCESS_build_bundle_report.py: The central cross-referencing engine. For each bundle, it verifies: (a) the chain hash in the JSON matches the independently computed SHA-256 of the previous bundle file, (b) the image hash in the JSON matches the independently computed SHA-256 of the PNG file, (c) the chain hash matches the expected overlay display hash, and (d) the OCR extraction matches the expected value. Produces the master verification report, overview report, and per-download detail report. Also computes SMPTE timecodes at 30fps relative to session start.

Stage 5 — 05_PROCESS_build_streams_report.py: Extracts and flattens all HTTP, HTTP-event, and network stream data from the combined bundle JSON into three chronologically sorted CSVs with SMPTE timecodes. This produces the session replay data used in the PLAYBACK workbook.

Stage 6 — 06_PROCESS_verify_bundle_timestamps.py: The most cryptographically intensive stage. For every OTS receipt: (a) independently hashes the data file and confirms it matches the OTS receipt's bound hash, (b) upgrades a copy of the OTS receipt if not yet anchored, (c) runs ots info to extract the Bitcoin block height, (d) queries the Blockstream Esplora API for block hash, timestamp, and median-time-past, (e) parses and cryptographically verifies the Roughtime bundle — checking Ed25519 CERT and SREP signatures, Merkle inclusion proof, nonce derivation, and delegation window.

Standalone Roughtime verifier — verify_roughtime_bundle.py: A self-contained script that parses the Roughtime wire format, verifies Ed25519 signatures with the correct Roughtime context strings ("RoughTime v1 response signature\0" and "RoughTime v1 delegation signature--\0"), checks Merkle inclusion via SHA-512 hash nodes, and verifies nonce derivation: NONC = SHA-512("RTNONC" + bytes.fromhex(SHA256)). Can be used by any third party to independently verify any Roughtime proof.

Significance

The pipeline is deterministic, non-destructive, and open-source. Given the same raw bundles directory, it produces identical verification outputs every time. It never modifies original evidence files. All source code is publicly available for independent audit.

Finding 7: Session-Level Statistics

Background

This finding aggregates metrics across the complete capture session from both the REPORTS_master and PLAYBACK workbooks.

Findings

Metric	Value
Total bundles produced	70 (0000–0069)
Session duration	~8 minutes 17 seconds (12:10:21 → 12:18:38 CST)
Total files hashed (pipeline)	277 files (JSON + PNG + OTS receipts)
Total OTS receipts verified	137 / 137 anchored (100.0%)
Total Roughtime proofs verified	137 / 137 all_ok (100.0%)
Total downloads captured	70 files
Total cases browsed (HAR)	12 distinct cases (2019 filing year)
Hash chain integrity	68/68 applicable links = True (100.0%)
Image hash integrity	68/68 applicable matches = True (100.0%)
OCR overlay match rate	69/69 applicable matches = True (100.0%)
Bitcoin block heights used	924,457 • 924,461 • 924,467 • 924,543
HAR entries (HTTP transactions)	1,048
Network stream packets captured	5,276
HTTP event flows captured	48
Bundle timestamp records	137

Significance

All verification metrics return 100% pass rates. No exceptions, anomalies, or partial failures were detected anywhere in the session data. The session produced a dense, multi-layered evidence chain: 70 bundles, 277 hashed files, 137 dual-timestamped artifacts, 1,048 HTTP transactions, and 5,276 network packets — all cross-referenced and independently verifiable.

Finding 8: Database Cross-Reference Spot-Check

Background

Case 27-CR-19-9140 was captured during this session and its docket PDF was sealed into a vault. This finding confirms the case exists in the MCRO forensic database and that basic identifying information matches between the captured docket and the database.

Findings

case_registry query results (see Appendix A1):

Field	Database Value	Captured Docket	Match
case_id	27-CR-19-9140	27-CR-19-9140	
defendant_name	HEATHER ANN ANDERSON	Heather Ann Anderson	
cluster_name	HEATHER ANDERSON	—	—
case_status	Dormant	Visible in docket	

parents_case_events summary (see Appendix A2):

Metric	Value
Total docket events in database	55
Earliest event date	2019-04-22
Latest event date	2025-08-27
Case type	CR (Criminal)
Case status	Dormant

Significance

The case exists in the database with matching identifying information. The captured docket page reflects a real case in the MCRO system. A comprehensive cross-check between the captured docket's event list and the database's parents_case_events table for this case (and all other cases captured in this session) will be the subject of Part 2.

Connecting Context

The MCRO forensic database contains 2,903 fully scraped cases (parents table), 9,146 cases in the registry, 4,251 court-filing PDFs, and 155,357 docket events. This spot-check confirms that the evidence-capture system's output connects to the broader corpus. The 12 cases browsed during this session are drawn from the 2019 filing-year subset of this corpus.

LIMITATIONS & ALTERNATIVE EXPLANATIONS

What the hash chain proves vs. what it does not: The hash chain proves sequence integrity and temporal ordering — it proves that the bundles were created in order and that none have been modified after creation. It does *not* prove that the captured content is “true” or “correct” — only that it has not been altered after capture. The system documents what one person captured from a public website; verifying that the website’s content accurately reflects underlying court records is a separate question (addressed in Part 2).

Self-signed certificate: The vault PDF signatures use a self-signed certificate (CN=Matthew David Guertin, OU=Minnesota Judicial Fraud Exposed). This means any PDF reader will display “Certificate issuer is unknown” — this is expected and does not indicate a signature failure. A self-signed certificate proves only that the same private key signed all vault PDFs; it does not establish identity through a trusted third-party Certificate Authority. The identity assurance comes from the dual-timestamping (OTS + Roughtime), not from the certificate chain.

OTS pending vs. anchored: All 137 OTS receipts in this session are confirmed anchored. However, OTS receipts require Bitcoin block confirmations, which can take hours to days after initial submission. Receipts examined before anchoring will show “pending” status. This does not indicate a failure — only that the Bitcoin network has not yet included the commitment in a block.

OCR limitations: The 6-character substring matching threshold was chosen to balance OCR error tolerance against false positive risk. OCR on small, low-resolution text strips can introduce character-level errors (e.g., ‘0’ vs. ‘O’, ‘1’ vs. ‘l’). The 6-character window tolerates scattered single-character errors while maintaining a false positive rate of approximately 1 in 285,000. Full 64-character exact matching would be more stringent but would fail on any OCR error.

Single-operator system: The evidence-capture system is operated by a single individual. It documents what that individual captured, when they captured it, and that the captures have not been altered. It does not independently verify the source data — that is the domain of the analytical reports that compare captured data against the forensic database (Part 2).

Bundle 0069: The final bundle (0069) has a PNG file but no JSON file, as it represents the last screenshot taken before session termination. Its OCR match was verified (True), but it cannot participate in chain-state verification (no JSON to hash). This is normal session-end behavior.

RECOMMENDATIONS FOR INDEPENDENT VERIFICATION

Any third party can independently verify the findings in this report using the following steps:

1. Clone the repository:

```
git clone https://github.com/Matt1Up/One-Way-Video.git
```

2. Obtain the raw bundles directory for the “2020_Docket_Event_Collect” session.

3. Run the automated verification pipeline:

```
python3 00_RUN_all_bundle_processing.py
```

This will reproduce all 6 stages of verification and produce identical output CSVs.

4. Verify any individual bundle’s hash:

```
sha256sum 0011.json
```

Compare the output against prev.last_hash.value in 0012.json.

5. Verify any OpenTimestamps receipt:

```
ots verify 0011.json.ots
```

6. Verify any Roughtime proof:

```
python3 verify_roughtime_bundle.py 0011.json.ots__time-stamp.json
```

7. Compare outputs against the Excel workbooks provided with this report. The BUNDLE_report_master, BUNDLE_ts_ots, and BUNDLE_ts_rt sheets should match the independently produced CSVs.

APPENDIX | SQL QUERIES

A1 — Case Registry Spot-Check

```
SELECT case_uid, case_id, defendant_name, cluster_id, cluster_name FROM case_registry  
WHERE case_id = '27-CR-19-9140';
```

Returns: case_uid = case:27-cr-19-9140, defendant_name = HEATHER ANN ANDERSON, cluster_id = 571

A2 — Parents Data and Event Summary

```
SELECT case_uid, case_id, case_status, case_type FROM parents WHERE case_id = '27-CR-19-9140';
```

```
SELECT COUNT(*) as event_count, MIN(event_json__date) as earliest,  
MAX(event_json__date) as latest FROM parents_case_events WHERE case_uid = 'case:27-cr-19-9140';
```

Returns: event_count = 55, earliest = 2019-04-22, latest = 2025-08-27

A3 — Corpus Statistics (Continuity Check)

```
SELECT (SELECT COUNT(*) FROM children) as total_children, (SELECT COUNT(*) FROM  
parents) as total_parents, (SELECT COUNT(*) FROM case_registry) as total_registry,  
(SELECT COUNT(*) FROM parents_case_events) as total_events;
```

Returns: children = 4,251; parents = 2,903; registry = 9,146; events = 155,357

APPENDIX | PYTHON PIPELINE REFERENCE

B0 — 00_RUN_all_bundle_processing.py

Purpose: Orchestrator that runs stages 1–6 sequentially; halts on any failure.

Key Logic: Uses subprocess.run() with check=True to enforce fail-fast behavior.

I/O: Input: all stage scripts in same directory. Output: all stage outputs.

B1 — 01_PROCESS_combine_json.py

Purpose: Merges sequential bundle JSONs into a single combined.json array.

Key Logic: Validates JSON parsing for each file. Pretty-prints with 2-space indent and sorted keys.

I/O: Input: 0000.json–NNNN.json. Output: combined.json.

B2 — 02_PROCESS_file_hash.py

Purpose: Computes SHA-256 for every bundle file and derives display_sha256 for PNGs.

Key Logic: Uses 8KB read chunks for memory efficiency. Walks backwards from each PNG index to find the nearest previous JSON hash.

I/O: Input: all bundle files. Output: BUNDLE_file_sha256.csv.

B3 — 03_PROCESS_png_ocr.py

Purpose: OCR extraction from hash overlay bars with 6-character substring matching.

Key Logic: Crop 18px → grayscale → autocontrast → sharpen → threshold(128) → invert-if-dark → pad to 26px. Tesseract with hex whitelist (--psm 7).

I/O: Input: PNGs + file hash CSV. Output: BUNDLE_file_sha256_ocr.csv + ocr_png/ directory.

B4 — 04_PROCESS_build_bundle_report.py

Purpose: Central cross-referencing engine producing master verification report.

Key Logic: Checks: chain hash match, image hash match, display hash match, OCR match. Computes SMPTE timecodes at configurable FPS.

I/O: Input: OCR CSV + combined.json + start_time.json. Output: BUNDLE_report_master.csv + overview.csv + downloads.csv.

B5 — 05_PROCESS_build_streams_report.py

Purpose: Extracts and flattens all stream data with timecodes.

Key Logic: Parses ISO8601 timestamps, converts CST to UTC, computes offsets from session start.

I/O: Input: combined.json + start_time.json. Output: BUNDLE_streams_http.csv + http_events.csv + net.csv.

B6 — 06_PROCESS_verify_bundle_timestamps.py

Purpose: Full OTS + Roughtime verification for all artifacts.

Key Logic: SHA-256 verification, ots info parsing, Blockstream Esplora API queries, full Roughtime cryptographic verification (Ed25519 + Merkle + nonce binding).

I/O: Input: all .ots + __time-stamp.json files. Output: BUNDLE_ts_master.csv + ts_ots.csv + ts_rt.csv.

B7 — verify_roughtime_bundle.py

Purpose: Standalone Roughtime protocol verifier.

Key Logic: Parses Roughtime wire format. Verifies Ed25519 CERT + SREP with context strings. SHA-512 Merkle tree (hash_leaf = SHA-512(0x00 + leaf), hash_node = SHA-512(0x01 + L + R)). NONC = SHA-512(b'RTNONC' + bytes.fromhex(SHA256)).

I/O: Input: any __time-stamp.json file. Output: verification result to stdout.

APPENDIX | CRYPTOGRAPHIC VERIFICATION COMMANDS

The following commands can be run by any third party to independently verify specific artifacts from this session:

Hash-Chain Verification

```
# Compute SHA-256 of a bundle file sha256sum 0011.json # Expected:
27bbc834d4a9cae9fb6fc4d2eb54d94cc47f4b86da912407715c902e79508905 # Verify it matches
prev.last_hash.value in the next bundle python3 -c "import json;
b=json.load(open('0012.json')); print(b['prev']['last_hash']['value'])" # Expected:
27bbc834d4a9cae9fb6fc4d2eb54d94cc47f4b86da912407715c902e79508905
```

OpenTimestamps Verification

```
# View OTS receipt details ots info 0011.json.ots # Verify receipt against Bitcoin
blockchain ots verify 0011.json.ots
```

Roughtime Verification

```
# Verify a Roughtime proof bundle python3 verify_roughtime_bundle.py 0011.json.ots__time-
stamp.json # Verify with explicit SHA-256 binding python3 verify_roughtime_bundle.py
0011.json.ots__time-stamp.json \ --bind-hash $(sha256sum 0011.json.ots | cut -d' ' -f1)
```

Nonce Derivation (Manual Check)

```
# Derive Roughtime nonce from OTS file SHA-256 python3 -c " import hashlib, binascii
ots_sha256 = '$(sha256sum 0011.json.ots | cut -d\' \' -f1)' nonce =
hashlib.sha512(b'RTNONC' + binascii.unhexlify(ots_sha256)).hexdigest() print('NONC =',
nonce) "
```

Full Pipeline Reproduction

```
# Clone the repository git clone https://github.com/Matt1Up/One-Way-Video.git # Navigate
to the bundles directory cd /path/to/session/bundles/ # Run the complete verification
pipeline python3 /path/to/00_RUN_all_bundle_processing.py # Compare outputs against the
provided Excel workbooks
```

Source: *Matthew Guertin v. State of Minnesota* | 27-CR-23-1886 | Evidence-Capture Session Controller | MattGuertin.com