VerifID: Design and Implementation of a Web-Based Document Authenticity Verification System Using SHA-256 Fingerprints and Institution-Branded QR Codes

Hanriyawan Adnan Mooduto [a,*]

[a] *Department of Information Technology, Politeknik Negeri Padang, Limau Manis, Padang, 25164, Indonesia*
Corresponding author: [*]mooduto@pnp.ac.id

***Abstract*— Forged and silently edited digital documents undermine trust in the certificates, letters, and approvals that institutions circulate as PDF files, while certificate-based digital signatures remain costly for small units to operate. This study designs, implements, and evaluates VerifID, a web-based system that lets an institution issue PDF documents whose authenticity any recipient can check without an account. The system was built with PHP and MariaDB on the server and with qrcodejs, pdf-lib, and html5-qrcode in the browser. Issuance takes two stages: the signer uploads the original PDF together with the signer's identity, the server issues a 256-bit random token, the browser stamps an institution-branded QR code carrying the verification link onto a page, and the server stores the final file together with its SHA-256 fingerprint. Verification recomputes the fingerprint of the received file and compares it with the registered one. The system was evaluated on a test bench with 43 black-box scenarios, ten tampering scenarios, 1,900 single-bit avalanche trials, timing measurements on 19 documents, and QR decoding tests. All 43 scenarios passed. Every modification, including one flipped bit and a content-preserving re-save, was reported as altered, whereas byte-identical copies were accepted. One-bit changes altered 49.99% of the digest bits on average. Hashing cost 3.70 ms per megabyte, so a 9 MB file was verified in under 50 ms. Stamped QR codes decoded reliably at 150 dpi and above. Further research should add issuer-held signing keys, trusted timestamps, and usability studies with institutional users.**

***Keywords*— Document authentication; QR code; SHA-256; hash function; document integrity; web application.**

I. INTRODUCTION

Institutions now issue most of their certificates, assignment letters, approvals, and transcripts as PDF files, and recipients pass these files on by e-mail and messaging applications. A PDF file is easy to edit, and an edited copy looks exactly like a genuine one on screen. Credential fraud is a recurring problem in higher education, where a forged certificate can decide a recruitment or an admission, and reviewers of verification systems describe manual confirmation with the issuer as slow and hard to scale [1], [2]. The Portable Document Format itself only describes how a page is drawn; it does not tell a reader whether the file still matches what the issuer released [3].

The established answer is the digital signature, in which the signer encrypts a document digest with a private key and anyone can check the result with the matching public key [4]. Indonesian law recognizes electronic signatures as part of its regulation of electronic information and transactions [5]. In practice, a certificate-based signature requires each signer to hold a key pair and a certificate and to keep the private key safe, which is a real burden for a study program or a small administrative unit. Signature validation in PDF viewers has also proved fragile: researchers forged valid-looking signatures in most desktop viewers, changed certified documents without invalidating them, and hid replaceable content inside files before signing [6]–[8].

A lighter family of solutions prints a QR code on the document. Several studies generate a signature from the document or from its key fields and store that signature in the QR code. Wellem *et al.* used the elliptic curve digital signature algorithm with SHA-256 for final-project approval forms [9], following an earlier RSA-based prototype [10]. Other authors combined SHA-256 with RSA for graduation letters and general documents [11], [12], a modified SHA-1 with a modified Blowfish cipher [13], and digital signatures with a distributed database [14]. Shaik proposed encrypting certificate fields into the QR code so that a scanning application could reveal forgery [2]. More recent designs protect the signer's private key with a password-derived AES key or use multi-factor RSA [15], [16]. These schemes work, yet they share two practical costs: every signer needs a key pair that must be stored somewhere, and the QR code grows with the signature it has to carry. Blockchain-based registries remove the central server from the trust chain, but they add network fees, node operation, and governance questions that reviews still list as open issues [1], [17]–[19].

The QR code is also a weak place to anchor trust. A QR symbol is only a machine-readable container; anyone can produce a look-alike symbol that points to another address, and people cannot read the payload before scanning it. Surveys of QR code security describe phishing, symbol replacement, and counterfeiting as the main attacks and recommend that applications verify the content behind the code and not the code itself [20]–[23].

Cryptographic hash functions offer a direct way to detect change. SHA-256 maps a file of any length to a 256-bit digest, and no practical collision is known for it [24], [25], whereas SHA-1 has been broken in practice and should no longer protect documents [26], [27]. A sound hash function also shows the avalanche property: changing one input bit changes about half of the output bits [28]. Comparing digests is therefore a common and inexpensive check of file integrity [29].

This paper presents VerifID, a web application developed at the Department of Information Technology of Politeknik Negeri Padang, which takes a registry approach. The issuing institution keeps the SHA-256 digest of each released file;

the QR code on the document carries only a random pointer to that record. The paper makes three contributions. First, it describes a keyless issuance and verification protocol in which the digest covers the final, QR-stamped file, so the visible mark and the registered fingerprint cannot drift apart. Second, it documents an implementation that stamps an institution-branded QR code inside the browser and applies standard web security controls on the server. Third, it reports a reproducible evaluation covering functional behavior, tamper detection, avalanche behavior, processing time, and QR readability, including two negative findings that operators need to know. Section II describes the materials and method, Section III presents and discusses the results, and Section IV concludes.

II. Materials and Method

A. Research Procedure

The work followed an iterative software engineering process with four activities: requirements analysis, design, implementation, and evaluation [30]. The version evaluated here is release 2.0, a redesign of an earlier release that kept the original workflow and data model and rebuilt the interface, the shared security layer, and the in-browser stamping step. The materials were the application source code, the structure of its production database, and the archive of final documents that the deployed instance had recorded.

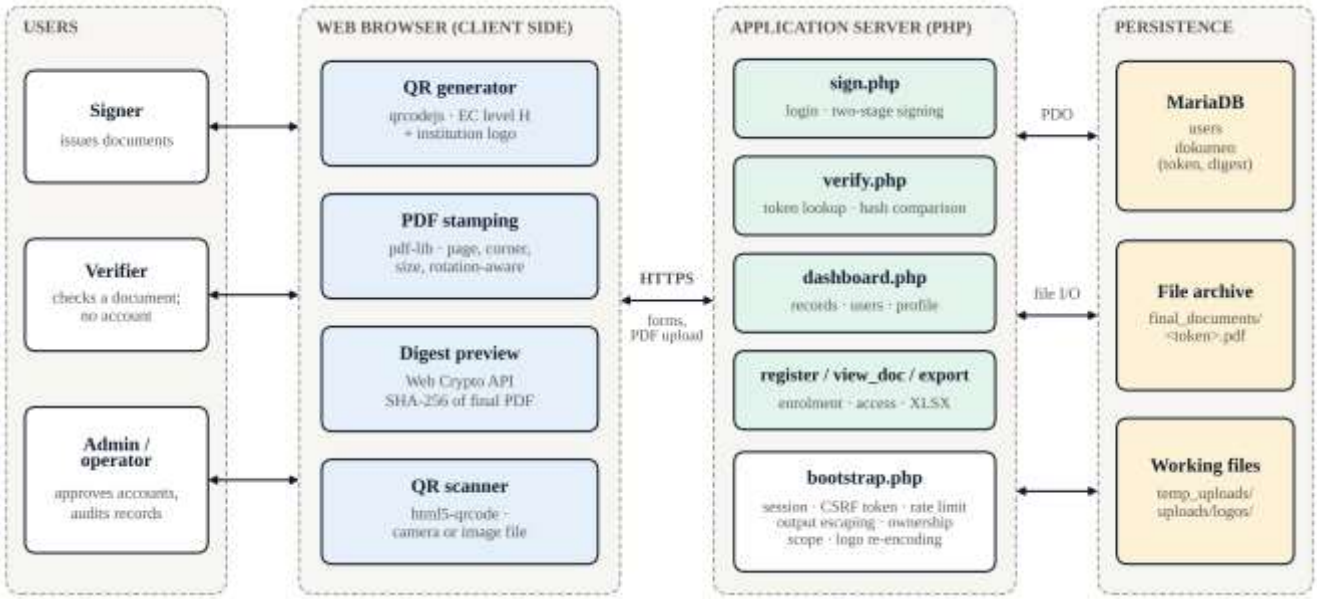

Fig. 1 Architecture of VerifID. The browser generates and stamps the QR code; the server validates uploads, computes the SHA-256 fingerprint, and keeps the registry and the archive.

B. Requirements

Three kinds of users interact with the system. A *signer* holds an account, issues documents, and manages the documents issued under that account. A *verifier* is any member of the public who receives a document; a verifier has no account. An *administrator* approves or rejects signer registrations, manages accounts and roles, and can audit every record, while an *operator* role can view every record without managing users. The functional requirements derived from these roles were:

- register a signer account that stays inactive until an administrator approves it;
- issue a document by uploading a PDF file and the signer's name, position, unit, and institution;
- produce a QR code that carries the verification link and the institution's logo, and place it on the document;
- record the fingerprint of the final file and keep a copy of that file;
- verify a received file against its record without logging in, by scanning the QR code or typing the token;
- search, view, delete, and export issuance records according to the user's role.

The non-functional requirements were that verification must run in a phone browser without an installed application, that a document of up to 10 MB must be verified within one second on the server, and that the application must resist the common web attacks listed by OWASP [31].

C. System Architecture

Fig. 1 shows the architecture. The server side is written in PHP and uses a MariaDB database through PDO; Apache serves the application over HTTPS. Four page controllers handle signing, verification, the dashboard, and registration, and two small controllers serve stored files and spreadsheet exports. A shared bootstrap module holds the session set-up, the cross-site request forgery (CSRF) token, the rate limiter, output escaping, the ownership rules, and the logo processing routine. The browser does more than display pages: it generates the QR code with qrcodejs 1.0.0, stamps it onto the PDF with pdf-lib 1.17.1, previews the SHA-256 digest of the stamped file with the Web Crypto API, and reads QR codes from the camera or from an image with html5-qrcode 2.3.8. Spreadsheet export uses PhpSpreadsheet and falls back to CSV when the library is absent.

D. Issuance and Verification Protocol

Fig. 2 summarizes the protocol. Issuance has two stages. In stage 1 the signer uploads the original PDF and the identity fields. The server checks the file type from the file content, enforces the 10 MB limit, stores the file in a temporary directory under a random name, computes the digest of the original, and draws a token t of 32 bytes from the operating system's cryptographically secure random generator. The token is written as 64 hexadecimal characters, and the verification address is the application URL followed by `verify.php?token=` and the token.

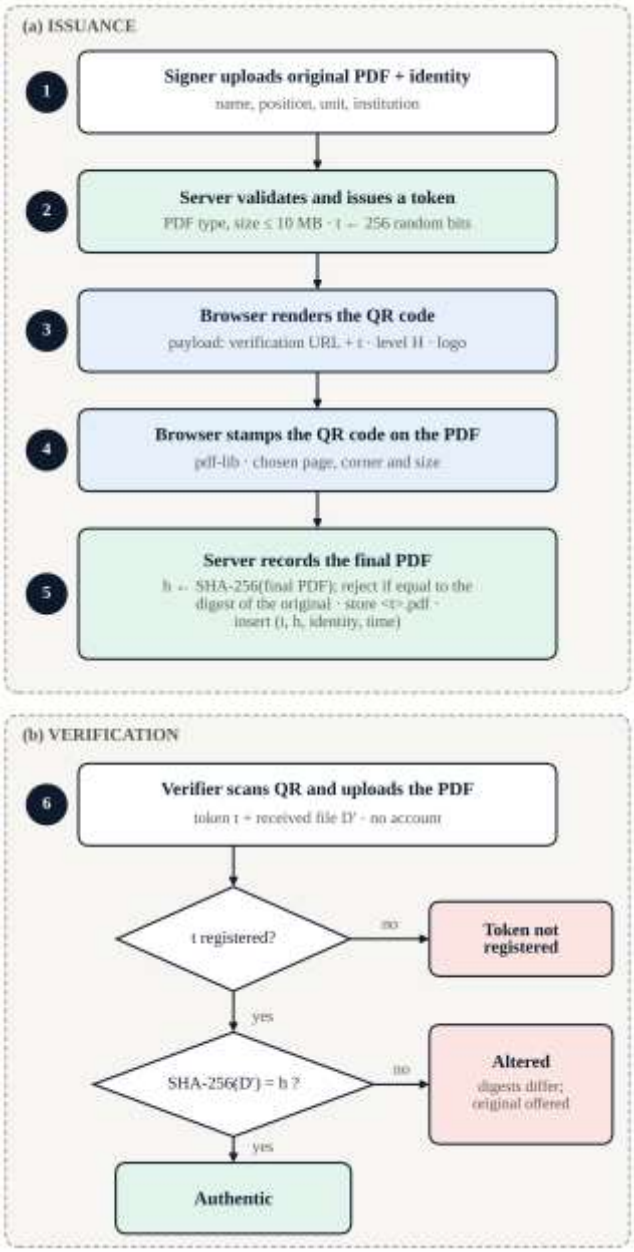

Fig. 2 Issuance (a) and verification (b) in VerifID.

In stage 2 the browser renders the QR code for that address and places it on the page that the signer selects. The signer can let the browser stamp the code or can download the QR image, place it with any PDF editor, and upload the result. In both cases the server receives the final file D_f and computes its fingerprint as

$$h_t = \text{SHA-256}(D_f) \tag{1}$$

The server refuses a final file whose digest equals the digest of the original, because such a file carries no QR code. It then stores the file as `<t>.pdf`, inserts the record (t, h_t, identity, time, account) into the registry, and deletes the temporary file. Because the digest is taken after stamping, the fingerprint covers the QR code as well as the content, and a QR code copied onto another document cannot make that document pass.

Verification needs the token and the received file D'. The registry R is searched for the token. If the token is unknown, the system reports that the document was never issued. Otherwise the verdict is

$$V(t, D') = 1 \iff \text{SHA-256}(D') = h_t \tag{2}$$

where $V = 1$ is shown as "authentic" and $V = 0$ as "altered." The two digests are compared with a constant-time string comparison. For an altered file the page marks the hexadecimal characters that differ and offers the archived original for download, so the verifier can see what the issuer actually released. Opening the QR link without uploading a file shows the issuance record only; this serves printed copies, for which a byte comparison is impossible.

E. Branded QR Code and Stamping

The verification address has 110 characters. At error correction level H, which lets a reader recover a symbol with roughly 30% of its codewords damaged, this payload needs a version 10 symbol of 57 × 57 modules [32]. The browser draws the symbol on a 640-pixel canvas with a quiet margin of 6% of the side and then covers the center with a white rounded plate whose side is 24% of the symbol side. The signer's institution logo is drawn inside the plate; an account without a logo gets the VerifID mark. The plate hides about 5.8% of the symbol area, which the error correction has to recover.

The stamping step loads the original PDF in the browser, embeds the canvas as a PNG image, and draws it on the first, last, or a chosen page at one of four corners, 28 points from the page edges. Three sizes are offered: 60, 78, and 100 points (2.1, 2.8, and 3.5 cm). The routine reads the page rotation and the crop box so that the code lands in the same visual corner on rotated pages. An optional two-line caption under the code tells the reader to scan it and prints the host name of the service, which gives a human-readable check against a look-alike address.

F. Data Model

The registry consists of two tables. Table I lists the fields of the `dokumen` table, which holds one row per issued document. The `users` table holds the account name, the password hash, the full name, the e-mail address, the role (admin, operator, or user), the status (pending, active, or rejected), and the path of the logo file. Ownership is recorded by account identifier, with the account's display name kept for records created before that column existed.

TABLE I
FIELDS OF THE ISSUANCE REGISTRY (TABLE DOKUMEN)

Field	Type	Purpose
token	varchar(64)	Random 256-bit pointer carried by the QR code
hash_dokumen	varchar(64)	SHA-256 fingerprint of the final PDF
nama_file	varchar(255)	Location of the archived final PDF
penanda_tangan, jabatan	varchar(100)	Signer's name and position
unit_kerja, instansi	varchar(255)	Signer's unit and institution
ditandatangani_pada	datetime	Time of issuance
user_id, ditandatangani_oleh	int; varchar(100)	Issuing account (identifier and display name)

G. Security Controls

Injection, broken access control, and authentication failures remain the most frequent weaknesses in web applications [31], [33], [34]. Table II maps the threats considered during design to the controls in the code. Passwords are stored with PHP's default password hashing, which currently is bcrypt, an adaptive function whose cost can be raised over time [35]; the login routine rehashes a password automatically when the default changes, which leaves a path to memory-hard functions such as Argon2 [36], [37]. Failed logins are limited per network address, in line with the throttling that NIST recommends for online guessing [38].

TABLE II
THREATS AND IMPLEMENTED CONTROLS

Threat	Control in VerifID
SQL injection	Prepared statements with native parameter binding for every query; token format checked before lookup
Cross-site scripting	All dynamic output passed through HTML escaping; `X-Content-Type-Options: nosniff`
Cross-site request forgery	256-bit session token required on every state-changing form; `SameSite=Lax` cookies
Password guessing	Five attempts per 15 minutes per address; uniform error message; dummy hash check for unknown users
Session theft or fixation	`HttpOnly` and `Secure` cookies; session identifier regenerated at login; account status rechecked on each request
Malicious upload	Type detected from file content; 10 MB limit; random server-side names; logos decoded and re-encoded to PNG; script execution disabled in upload folders
Unauthorized access	Role and ownership checks on listing, viewing, deleting, and exporting; new accounts need approval
Clickjacking, data leaks	`X-Frame-Options: SAMEORIGIN`; database errors logged, never shown; directory listing off

H. Evaluation Design

The evaluation ran on a test bench that executed the unmodified application code under PHP 8.3.6 with its built-in web server on a two-core Intel Xeon virtual machine with 7 GB of memory. The bench replaced MariaDB with SQLite through a three-line change in the connection routine and served the JavaScript libraries locally; nothing else was changed. A script drove Chrome 153 in headless mode through the real pages, so stamping and digest preview ran in an actual browser engine. Seven synthetic PDF files of 2 KB to 9.4 MB and 1 to 120 pages, one of them with rotated pages, were issued through the full workflow, and the twelve final documents found in the production archive (0.98 to 9.60 MB, 2 to 213 pages) served as additional inputs for hashing. Five experiments were carried out.

1) Functional and security testing: Forty-three black-box scenarios were run against the HTTP interface. Each scenario stated an input, an expected response, and a pass criterion checked by the script.

2) Tamper detection: Each of the seven issued files was modified in ten ways and submitted to the verification page with its own token.

3) Avalanche behavior: For each of the 19 files, one randomly chosen bit was inverted 100 times, and the Hamming distance *HD* between the new digest and the original digest was recorded. The avalanche effect over n trials is

$$AE = (100\% / 256n)\,\Sigma_{i=1..n}\, HD(H(D), H(D_i)) \qquad (3)$$

4) Processing time: The time of PHP's file hashing function was measured 30 times per file and the median was kept. The complete verification request, including upload over the loopback interface, was measured ten times per issued file. Stamping time was read from the browser clock.

5) QR readability: The stamped page of each issued file was rasterized at 72, 100, 150, and 300 dpi and decoded with the ZXing-C++ library. A second test rebuilt the symbol for 100 random tokens at the four error correction levels, varied the side of the logo plate from 0% to 52% of the symbol side, reduced each image to the pixel size of a 2.8 cm code scanned at 300 dpi, added blur and noise, and recorded the share of symbols that decoded to the exact address.

III. RESULTS AND DISCUSSION

A. Implementation

The application was implemented as designed, and a production instance is online. Fig. 3 shows stage 2 of issuance: the signer picks the page, the corner, and the size, sees the position on a page mock-up, and presses one button to stamp the code. The lower part of the figure is a QR code produced for an account that had uploaded the logo of Politeknik Negeri Padang. After the stamped file is saved, a confirmation page lists the token, the fingerprint, and the verification link.



Fig. 3 Stamping options in stage 2 of issuance (top) and a QR code branded with the institution logo (bottom).

Fig. 4 shows the two verdicts on the verification page. The authentic verdict states that the fingerprint of the uploaded file equals the registered one. The altered verdict lists both fingerprints and highlights the characters that differ, in this example 61 of 64 after a single bit of the file was inverted. Both pages also show who issued the document and when, and both can be printed as a verification report.

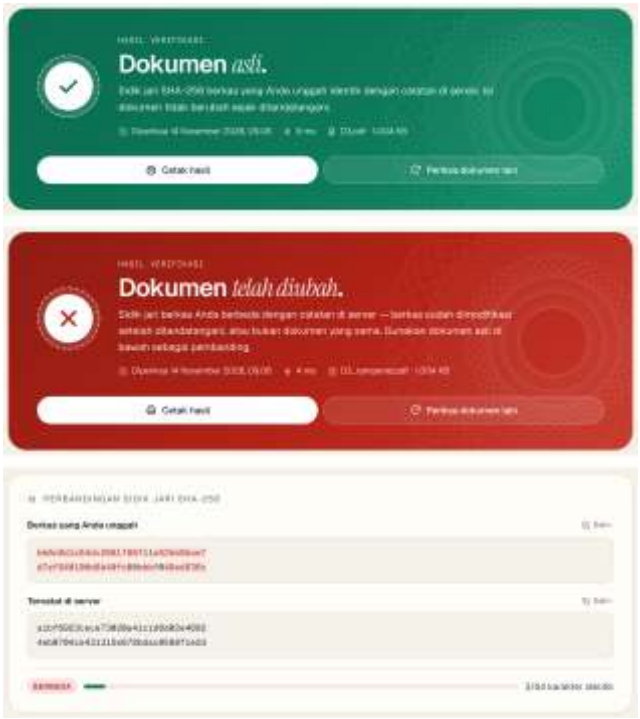

Fig. 4 Verification results: authentic (top) and altered, with the fingerprint comparison (middle and bottom).

B. Functional and Security Testing

All 43 scenarios produced the expected response (Table III). In the issuance group, the seven documents issued through the browser each received a distinct 64-character token, and for every document the digest previewed in the browser, the digest stored in the registry, and the digest of the archived file were identical. Uploading the unstamped original as the final file was refused, as were non-PDF files, files above 10 MB, and requests without the CSRF token. In the access control group, a second signer could neither view nor delete the first signer's document, and a non-administrator who posted an administrative action was refused. After an owner deleted a record, its token was reported as not registered, which is the intended way to revoke a document.

TABLE III
SUMMARY OF BLACK-BOX FUNCTIONAL AND SECURITY TESTING

Group	Scenarios exercised	Passed
Login and session	Valid login; wrong password; account awaiting approval; missing CSRF token; SQL-injection string as user name; lock after five failures; correct password during the lock; dashboard without a session	8/8
Registration	New signer stored as pending; duplicate user name; weak password; PHP script submitted as a logo image	4/4
Issuance	Non-PDF file; file above 10 MB; empty identity field; stage 2 without a signing session; unstamped original submitted as final; missing CSRF token; full browser workflow for seven documents; token format and uniqueness	8/8
Verification	Registered file; one bit inverted; pre-stamp original; token of another document; unregistered token; malformed token; full link pasted as token; SQL-injection string as token; non-PDF file; oversized file; QR link opened without a file; QR link with unregistered token	12/12
Output encoding	Script tag stored in an identity field and rendered on the public verification page	1/1
Access control	Viewing another signer's file; operator access; unauthenticated file request; listing limited to own records; user-management tab and administrative action by a non-administrator; deleting another signer's record; export; revocation by the owner; approval of a pending account	10/10
Total		**43/43**

C. Tamper Detection

Table IV gives the verdicts for the ten modification scenarios. The two benign cases, a byte-identical copy and the same bytes under another file name, were accepted for all seven documents, which confirms that the verdict depends on content only. Every other case was reported as altered: 56 of 56 modified files were detected, and no modified file was accepted.

TABLE IV
VERDICTS FOR TEN MODIFICATION SCENARIOS (SEVEN DOCUMENTS EACH)

Scenario	Authentic	Altered
Byte-identical copy	7/7	0/7
Same bytes, different file name	7/7	0/7
One bit flipped at a random offset	0/7	7/7
One byte appended at end of file	0/7	7/7
Last byte removed (truncation)	0/7	7/7
Document title metadata edited	0/7	7/7
One page removed (blank page added for the 1-page file)	0/7	7/7
Blank page appended	0/7	7/7
First page rotated 180 degrees	0/7	7/7
Re-saved by another PDF library, content untouched	0/7	7/7

The last row deserves attention. Re-saving a file with another PDF library changed no visible content, yet the verdict was "altered" for every document, because the library rewrote the object structure and so changed the bytes. A fingerprint over the file bytes cannot tell a harmless re-encoding from a malicious edit. The system therefore gives a strict guarantee, that the file is bit-for-bit the one the issuer released, and operators must tell recipients to forward the original file and not a copy that was printed to PDF, compressed by a messaging application, or re-saved by a viewer. The verification page softens this limitation by offering the archived original whenever the digests differ.

D. Avalanche Behavior

Fig. 5 shows the distribution of the Hamming distance over 1,900 single-bit changes. The mean distance was 127.97 of 256 bits, an avalanche effect of 49.99%, with a standard deviation of 7.74 bits and a range of 107 to 154 bits. These values agree with the binomial distribution expected from an ideal hash function, whose mean is 128 bits and whose standard deviation is 8 bits, and with the near-50% figures reported for the SHA-2 family [28]. In the notation that the verification page displays, a one-bit change altered 60.0 of the 64 hexadecimal characters on average and never fewer than 53. A modified file therefore produces a visibly different fingerprint; no partial match can be mistaken for a genuine one.

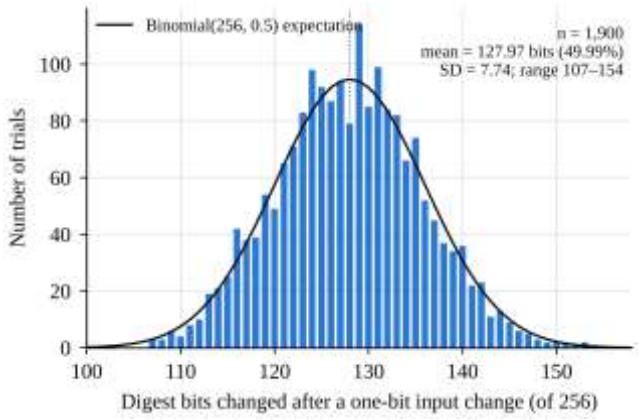

Fig. 5 Hamming distance between digests after a one-bit change in the file (1,900 trials on 19 documents).

E. Processing Time

Hashing time grew linearly with file size (Fig. 6). A least-squares line through the 19 files has a slope of 3.70 ms per megabyte, which corresponds to about 270 MB/s, with a coefficient of determination of 0.9999:

$$T_{hash}(s) \approx 3.70\ s \ \text{ms}, \quad \text{with } s \text{ in MB} \tag{4}$$

The largest archived document, 9.60 MB with 213 pages, was hashed in 35.6 ms. Table V lists the measurements for the issued documents. A complete verification request took 1.4 ms for the smallest file and 48.7 ms for the 9 MB file on the loopback interface, far below the one-second requirement; hashing accounted for most of the server time, and the registry lookup was negligible. Stamping in the browser took 162 to 259 ms and did not grow with file size, because pdf-lib appends one image to one page. On a real network the upload of the file will dominate the response time, so the 10 MB limit is the main lever on perceived speed.

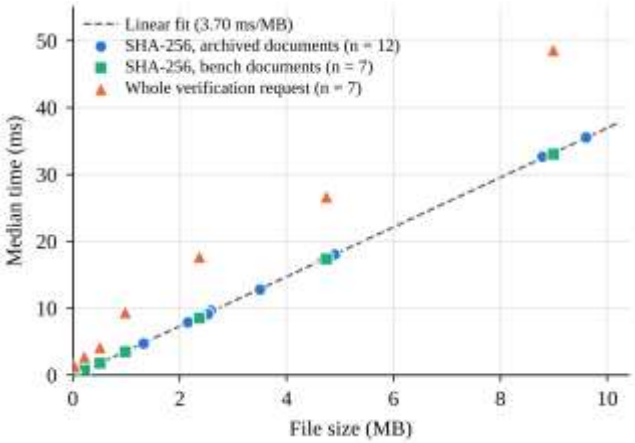

Fig. 6 Median SHA-256 time and whole-request verification time against file size.

TABLE V
PROCESSING TIME FOR THE SEVEN ISSUED DOCUMENTS (MEDIANS)

File	Pages	Size (MB)	Hash (ms)	Request (ms)	Stamp (ms)
D1	1	0.02	0.09	1.4	259
D2	3	0.50	1.79	4.2	204
D3	10	0.98	3.50	9.5	199
D4	25	2.37	8.55	17.8	250
D5	60	4.74	17.40	26.7	256
D6	120	8.99	33.04	48.7	257
D7	2	0.21	0.76	2.8	162

F. QR Code Readability

Table VI reports whether the stamped code could be decoded from the rendered page. At 150 and 300 dpi the code was read on all seven documents, for every size, corner, and page tested, including the document with rotated pages. At 100 dpi only 3 of 7 were read and at 72 dpi only 2 of 7. The cause is geometric: a 2.8 cm code of 57 modules has a module pitch of 0.42 mm, which is 2.5 pixels at 150 dpi and only 1.2 pixels at 72 dpi, too few for a reader to separate neighboring modules. Printing at normal quality or

viewing the page at ordinary zoom gives ample resolution, but a low-resolution screenshot or fax of the page may not scan. Users who expect such copies should choose the 3.5 cm size.

TABLE VI
DECODING OF THE STAMPED QR CODE AT FOUR RENDERING RESOLUTIONS

File	Side (cm)	Corner, page	72 dpi	100 dpi	150 dpi	300 dpi
D1	2.8	BR, first	yes	no	yes	yes
D2	2.1	BR, first	no	yes	yes	yes
D3	2.8	BL, last	no	no	yes	yes
D4	3.5	TR, first	yes	yes	yes	yes
D5	2.8	TL, last	no	no	yes	yes
D6	2.8	BR, first	no	no	yes	yes
D7	2.8	BR, first	no	yes	yes	yes
Read			**2/7**	**3/7**	**7/7**	**7/7**

Fig. 7 shows how much of the symbol the logo plate may cover. At level H every symbol still decoded with a plate side of 48% of the symbol side and none decoded at 52%. The 24% plate used by VerifID is half of that limit in side length and a quarter of it in area, so the logo leaves a wide reserve for print damage. The same plate would be unsafe at level L, where only 54% of the symbols decoded, and levels M and Q failed from 36% and 48%. Level H costs a denser symbol, version 10 instead of version 6, and this density is what limits readability at low resolution; the two findings are two sides of one design choice. The simulation used the institution logo, whereas the issued test documents carried the default VerifID mark; the plate geometry is identical in both cases.

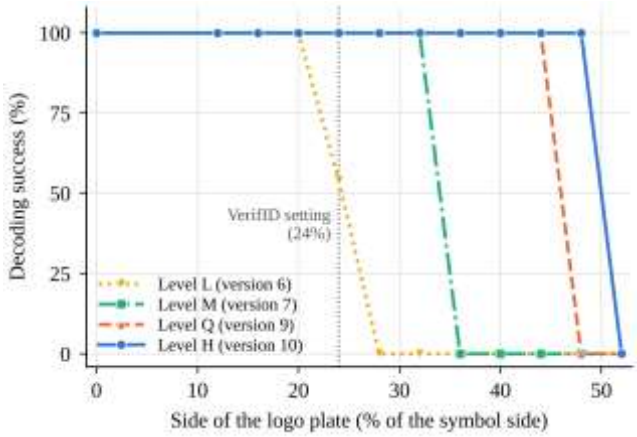

Fig. 7 Decoding success against the size of the logo plate for the four error correction levels (100 symbols per point).

G. Comparison with Related Work

Table VII places VerifID beside the QR-based schemes reviewed in Section I. The signature-based schemes bind a document to a key holder and can be checked without trusting a database, at the cost of key management for every signer. VerifID removes keys from the signer's side entirely and keeps the QR payload short and constant, which suits units that issue documents under an institutional account. The price is that trust rests on the issuing server.

TABLE VII
QR-BASED DOCUMENT AUTHENTICATION APPROACHES

Study	Mechanism and QR content	Signer-side keys
Wellem *et al.* [9]	ECDSA signature with SHA-256; the QR code stores the signature	Key pair per signer
Hutagalung *et al.* [11]	RSA signature over a SHA-256 digest; the QR code stores the signature	Key pair per signer
Quilala and Quilala [13]	Modified SHA-1 digest encrypted with modified Blowfish; the QR code stores the encrypted digest	Shared secret key
Adithia *et al.* [15]	RSA-2048 signature with SHA-256; the QR code stores the signature	Key pair per signer; private key encrypted with a password-derived AES key
VerifID (this work)	SHA-256 fingerprint of the final file kept in the issuer's registry; the QR code stores a link with a 256-bit random token	None; the signer logs in with an account

H. Limitations

Five limitations follow from the design and from the tests. First, VerifID proves that a file equals the file registered by an account; it does not produce a signature that a third party can check without the server, so it does not offer the non-repudiation of a certificate-based signature and should be presented as an authenticity registry. Second, anyone with access to the database could replace a stored digest; anchoring digests in an append-only log or adding a trusted timestamp would close that gap. Third, the token works as a bearer credential: whoever holds the QR code can read the issuance record and download the archived original. This is acceptable for documents meant to be shown to third parties, and the 256-bit token cannot be guessed, but confidential documents would need the download to be switched off or tied to an additional check. Fourth, the QR code remains exposed to substitution: a forger can print a code that leads to a look-alike site which always answers "authentic" [20], [23]. The printed host name under the code helps only if verifiers read it. Finally, the evaluation used a test bench and synthetic files; it measured server and browser processing, not network conditions, camera scanning of printed pages, or user experience.

IV. CONCLUSION

This study designed, implemented, and evaluated VerifID, a web application that lets an institution issue PDF documents carrying an institution-branded QR code and lets any recipient check them without an account. The design keeps the SHA-256 fingerprint of the final, stamped file in the issuer's registry and puts only a 256-bit random pointer in the QR code, so signers need no key pairs. On the test bench, all 43 functional and security scenarios passed, all 56 modified files were reported as altered while exact copies were accepted, single-bit changes altered 49.99% of the digest bits, hashing cost 3.70 ms per megabyte, and stamped codes decoded at 150 dpi and above with a wide error-correction reserve around the logo.

Two results matter for operation. The fingerprint is strict: a file that has been re-saved or recompressed is reported as altered even when its content is intact, so recipients must circulate the original file. The dense level-H symbol needs adequate resolution, so the largest stamp size is advisable when low-quality copies are expected. Further research should add an issuer-held signing key so that records can be checked independently of the server, attach trusted timestamps or an append-only log to the registry, test scanning of printed documents with a range of phones, and measure usability and acceptance among signers and verifiers in daily use.

ACKNOWLEDGMENT

The author thanks the Department of Information Technology, Politeknik Negeri Padang, for the facilities and support provided for the development and deployment of VerifID.

REFERENCES

[1] A. Rustemi, F. Dalipi, V. Atanasovski, and A. Risteski, "A systematic literature review on blockchain-based systems for academic certificate verification," *IEEE Access*, vol. 11, pp. 64679–64696, 2023, doi: 10.1109/ACCESS.2023.3289598.

[2] C. Shaik, "Preventing forged and fabricated academic credentials using cryptography and QR codes," *Int. J. Comput. Sci. Eng. Appl.*, vol. 11, no. 1, Feb. 2021.

[3] *Document Management—Portable Document Format—Part 2: PDF 2.0*, ISO 32000-2:2020, Int. Org. Standardization, Geneva, Switzerland, 2020.

[4] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 3rd ed. Boca Raton, FL, USA: CRC Press, 2020.

[5] Republic of Indonesia, *Law No. 1 of 2024 on the Second Amendment to Law No. 11 of 2008 on Electronic Information and Transactions*, Jakarta, Indonesia, 2024.

[6] V. Mladenov, C. Mainka, K. Meyer zu Selhausen, M. Grothe, and J. Schwenk, "1 trillion dollar refund: How to spoof PDF signatures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, London, U.K., 2019, pp. 1–14, doi: 10.1145/3319535.3339812.

[7] S. Rohlmann, V. Mladenov, C. Mainka, and J. Schwenk, "Breaking the specification: PDF certification," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, 2021, pp. 1485–1501, doi: 10.1109/SP40001.2021.00110.

[8] C. Mainka, V. Mladenov, and S. Rohlmann, "Shadow attacks: Hiding and replacing content in signed PDFs," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2021.

[9] T. Wellem, Y. Nataliani, and A. Iriani, "Academic document authentication using elliptic curve digital signature algorithm and QR code," *JOIV: Int. J. Informat. Vis.*, vol. 6, no. 3, pp. 667–675, Sep. 2022.

[10] A. Lorien and T. Wellem, "Implementation of a document authentication system based on quick response (QR) code and digital signature" (in Indonesian), *J. RESTI (Rekayasa Sist. Teknol. Inf.)*, vol. 5, no. 4, pp. 663–671, 2021.

[11] J. Hutagalung, P. S. Ramadhan, and S. J. Sihombing, "Data security using secure hashing algorithm (SHA)-256 and Rivest Shamir Adleman (RSA) in digital signature" (in Indonesian), *J. Teknol. Inf. Ilmu Komput.*, vol. 10, no. 6, Dec. 2023, doi: 10.25126/jtiik.1067319.

[12] F. Az-Zahra, R. Marwati, and R. Sispiyati, "Implementation of QR code with the secure hash algorithm (SHA)-256 and enhanced Rivest Shamir Adleman (RSA) for digital document authentication" (in Indonesian), *J. EurekaMatika*, vol. 12, no. 1, pp. 11–22, 2024.

[13] R. L. Quilala and T. F. G. Quilala, "Document verification using quick response code with modified secure hash algorithm-1 and modified blowfish algorithm," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 28, no. 1, pp. 470–479, Oct. 2022, doi: 10.11591/ijeecs.v28.i1.pp470-479.

[14] M. W. Ayub, I. Winarno, and A. Sudarsono, "QR code-based smart document implementation using distributed database and digital signature," *Indones. J. Comput. Sci.*, vol. 13, no. 1, pp. 79–92, 2024, doi: 10.33022/ijcs.v13i1.3673.

[15] M. T. Adithia, N. Elianora, and M. Veronica, "Document verification and authentication by using password based QR code signature with RSA 2048, AES encryption, and SHA-256," *J. Tek. Inform. (JUTIF)*, vol. 7, no. 2, pp. 1981–1995, Apr. 2026.

[16] B. D. Yulianto and W. Wendanto, "Enhanced QR code based digital signatures for secure document management toward sustainable green investment," *JITK (J. Ilmu Pengetah. Teknol. Komput.)*, vol. 11, no. 4, May 2026, doi: 10.33480/jitk.v11i4.7728.

[17] M. Turkanović, M. Hölbl, K. Košič, M. Heričko, and A. Kamišalić, "EduCTX: A blockchain-based higher education credit platform," *IEEE Access*, vol. 6, pp. 5112–5127, 2018, doi: 10.1109/ACCESS.2018.2789929.

[18] G. Caldarelli and J. Ellul, "Trusted academic transcripts on the blockchain: A systematic literature review," *Appl. Sci.*, vol. 11, no. 4, Feb. 2021, Art. no. 1842, doi: 10.3390/app11041842.

[19] C. Delgado-von-Eitzen, L. Anido-Rifón, and M. J. Fernández-Iglesias, "Blockchain applications in education: A systematic literature review," *Appl. Sci.*, vol. 11, no. 24, Dec. 2021, Art. no. 11811, doi: 10.3390/app112411811.

[20] K. Krombholz, P. Frühwirt, P. Kieseberg, I. Kapsalis, M. Huber, and E. Weippl, "QR code security: A survey of attacks and challenges for usable security," in *Human Aspects of Information Security, Privacy, and Trust* (Lecture Notes in Computer Science, vol. 8533). Cham, Switzerland: Springer, 2014, pp. 79–90, doi: 10.1007/978-3-319-07620-1_8.

[21] H. A. M. Wahsheh and F. L. Luccio, "Security and privacy of QR code applications: A comprehensive study, general guidelines and solutions," *Information*, vol. 11, no. 4, Apr. 2020, Art. no. 217, doi: 10.3390/info11040217.

[22] R. Focardi, F. L. Luccio, and H. A. M. Wahsheh, "Usable security for QR code," *J. Inf. Secur. Appl.*, vol. 48, Oct. 2019, Art. no. 102369, doi: 10.1016/j.jisa.2019.102369.

[23] D. Njuguna and J. Ndia, "Quick response code security attacks and countermeasures: A systematic literature review," *J. Cyber Secur.*, vol. 7, pp. 1–20, 2025, doi: 10.32604/jcs.2025.059398.

[24] *Secure Hash Standard (SHS)*, FIPS PUB 180-4, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Aug. 2015, doi: 10.6028/NIST.FIPS.180-4.

[25] Z. A. Al-Odat, M. Ali, A. Abbas, and S. U. Khan, "Secure hash algorithms and the corresponding FPGA optimization techniques," *ACM Comput. Surv.*, vol. 53, no. 5, Sep. 2020, Art. no. 97, doi: 10.1145/3311724.

[26] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full SHA-1," in *Advances in Cryptology—CRYPTO 2017* (Lecture Notes in Computer Science, vol. 10401). Cham, Switzerland: Springer, 2017, pp. 570–596, doi: 10.1007/978-3-319-63688-7_19.

[27] G. Leurent and T. Peyrin, "SHA-1 is a shambles: First chosen-prefix collision on SHA-1 and application to the PGP web of trust," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 1839–1856.

[28] D. Upadhyay, N. Gaikwad, M. Zaman, and S. Sampalli, "Investigating the avalanche effect of various cryptographically secure hash functions and hash-based applications," *IEEE Access*, vol. 10, pp. 112472–112486, 2022, doi: 10.1109/ACCESS.2022.3215778.

[29] P. Boyanov, "Practical applications of hash functions MD5, SHA-1, and SHA-256 using various software tools to verify the integrity of files," *J. Sci. Appl. Res.*, vol. 27, no. 1, pp. 120–137, 2024, doi: 10.46687/jsar.v27i1.413.

[30] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. New York, NY, USA: McGraw-Hill Education, 2020.

[31] OWASP Foundation, "OWASP Top 10:2021," 2021. [Online]. Available: https://owasp.org/Top10/

[32] *Information Technology—Automatic Identification and Data Capture Techniques—QR Code Bar Code Symbology Specification*, ISO/IEC 18004:2015, Int. Org. Standardization, Geneva, Switzerland, 2015.

[33] O. Andreeva, "Top 10 web application vulnerabilities in 2021–2023," Securelist by Kaspersky, Mar. 2024. [Online]. Available: https://securelist.com

[34] J. Juremi, A. Allouche, and K. F. Ishraq, "Web application common vulnerabilities and preventions," *J. Appl. Technol. Innov.*, vol. 9, no. 2, pp. 49–54, 2025, doi: 10.65136/jati.v9i2.29.

[35] N. Provos and D. Mazières, "A future-adaptable password scheme," in *Proc. USENIX Annu. Tech. Conf. (FREENIX Track)*, Monterey, CA, USA, 1999, pp. 81–91.

[36] K. Chouhan, S. Singh, A. Kaintyura, and A. K. Mahato, "Comparative analysis of Argon2 and Bcrypt security methods to strengthen password security," *J. Sci. Technol. Res.*, vol. 13, no. 2, pp. 32–39, 2023.

[37] A. Biryukov, D. Dinu, D. Khovratovich, and S. Josefsson, *Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications*, RFC 9106, Internet Res. Task Force, Sep. 2021, doi: 10.17487/RFC9106.

[38] P. A. Grassi *et al.*, *Digital Identity Guidelines: Authentication and Lifecycle Management*, NIST Special Publication 800-63B, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Jun. 2017, doi: 10.6028/NIST.SP.800-63b.