Towards Secure QR Code Authentication: A Digital Signature-Based Approach for Secure and Tamper-Resistant Access Verification

Timothy Marvine - 18223021
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: timothymarvine@gmail.com , 18223021@std.stei.itb.ac.id

***Abstract*—Although Quick Response (QR) codes are widely used for access verification, static QR codes are highly vulnerable to cloning, data tampering, and replay attacks because they lack data integrity. To solve this security flaw, this paper presents a secure and tamper-resistant authentication framework using the Elliptic Curve Digital Signature Algorithm (ECDSA) on the SECP256R1 curve combined with SHA-256 hashing. The core contribution of this work is the development of a real-time authentication server built with FastAPI and an SQLite database to handle single-use tokens. By combining an asymmetric digital signature with a secure random hex nonce and strict expiration timestamps, the system can instantly validate incoming requests. Experimental results show that the application successfully detects and blocks modified data (*Invalid Signature*), expired tokens (*QR Expired*), and duplicated codes (*Replay Attack Detected*). Furthermore, because ECDSA signatures are compact, the generated QR codes maintain a low module density, ensuring fast and reliable optical scanning in high-traffic environments.**

Keywords—QR Code; Authentication; Digital Signature; ECDSHA; SHA-256; Replay Attack Mitigation

I. INTRODUCTION

For a long time, physical stamps and handwritten signatures have been the standard way to prove the authenticity of official and academic documents [1]. However, with today's cheap and high-quality scanners or editing software, these physical credentials can easily be forged or copied [2]. On top of that, verifying these documents manually is slow, full of mistakes, and wastes administrative time, often requiring manual verification from the original issuer [3].

To solve this problem without heavily loading network services, researchers are combining two-dimensional (2D) barcodes specifically QR codes with modern public-key cryptography [1]. A QR code is an open-source, damage-resistant format that can store quite a bit of data and can be scanned quickly by any camera [3]. Instead of just storing unsecured plain text or basic website links, secure modern systems embed digital signatures directly inside the QR code data payload [3]. Once a QR code is signed, it guarantees three core security properties: it proves who sent it (authentication), ensures the data hasn't been changed (integrity), and prevents the issuer from denying it (non-repudiation) [5].

When it comes to digital signatures, the Elliptic Curve Digital Signature Algorithm (ECDSA) is now the standard choice for lightweight systems [4]. Compared to older methods like RSA, ECDSA delivers the exact same level of security but uses much smaller key sizes [4]. For example, a 256-bit ECDSA key provides the same protection as a massive 3072-bit RSA key [5]. This size difference is incredibly important for QR codes [1]. Embedding a huge RSA signature makes the QR code extremely dense and packed with tiny dots, making it slow to scan and highly sensitive to camera blur [2]. On the other hand, ECDSA keeps the QR code clean and simple, allowing for instant scanning speeds [1].

However, even with the strong mathematics behind ECDSA, standard implementations in current literature still have a major flaw: they are vulnerable to replay attacks [3]. Traditional signed QR codes are static and remain valid forever unless the system checks them against a continuous validation ledger [2]. If an attacker intercepts a copy of a valid signed QR code, they can simply reuse or re-transmit that exact same image to bypass security scanners [3].

To completely eliminate this threat, this paper introduces a highly secure access verification framework that upgrades standard ECDSA by adding an automated anti-replay system. The main contribution of this work is the development of a fully operational backend built on the FastAPI web framework and supported by an optimized SQLite database ledger. By combining a SHA-256 hash digest with a unique single-use hex nonce and strict Unix expiration timestamps directly into the ECDSA signed structure, our architecture can instantly detect and reject tampered data, expired credentials, and cloned replay requests in real-time.

II. Literature Review

A. Digital Signature Fundmentals

A digital signature is a mathematical scheme used to demonstrate the authenticity, integrity, and non-repudiation of a digital message or document [3]. The system functions using asymmetric public-key cryptography, which relies on a mathematically linked key pair: a private key (d) kept strictly secret by the signer, and a public key (Q) openly distributed to verifiers [4]. The signature generation process relies on a one-way cryptographic hash function, which maps a variable-length message into a fixed-length string called a message digest [5].

B. Maintaining the Integrity of the Specifications

Traditional signing system like RSA rely on the difficulty of integer factorization $(n = p \cdot q)$ [5]. In contrast, the security of Elliptic Curve Cryptography (ECC) and ECDSA depends on the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP), which involves finding an integer x such that $Q = xP$ over a finite field [4]. Because there are no known sub-exponential time algorithms to solve the ECDLP on properly configured curves, ECDSA offers drastically superior strength-per-key-bit compared to RSA [4]. This operational disparity is formalized in Table I:

TABLE I. Cryptographic Algorithm Comparison

Security Level (Bits)	RSA Key Size (Bits)	ECDSA Key Size (Bits)	Relative Computational Power
80	1024	160	Extremely Low
112	2048	224	Low
128	3072	256	Moderate
256	15360	512	High

C. Impact of Cryptographic Payload on QR Code Density

A standard QR code employs Reed-Solomon error correction codes divided into four selectable levels: Low (L), Medium (M), Quartile (Q), and High (H) [1]. As the size of the underlying data payload increases, the QR code must automatically transition to higher versions, ranging from Version 1 at 21×21 modules up to Version 40 at 177×177 modules [2].

If a system employs RSA-3072, the resulting digital signature alone consumes 384 bytes of data [5]. When encoded into a QR code with Level H error correction, it forces the matrix to expand past Version 15 (77×77 modules), creating an incredibly dense grid of black and white blocks that causes significant scanning failure under typical smartphone camera distortions [1], [2]. By utilizing ECDSA-P256, the signature is compressed to a mere 64 bytes, enabling the token to remain within lightweight versions, ensuring instant optical scanning performance [1], [4].

III. System Architecture And Design

The proposed secure QR authentication system utilizes a decentralized client-server architecture containing three distinct operational steps: Token Generation, Optical Transmission, and Database-Backed Verification. The architecture is explicitly designed to combat both database manipulation and external interceptors.

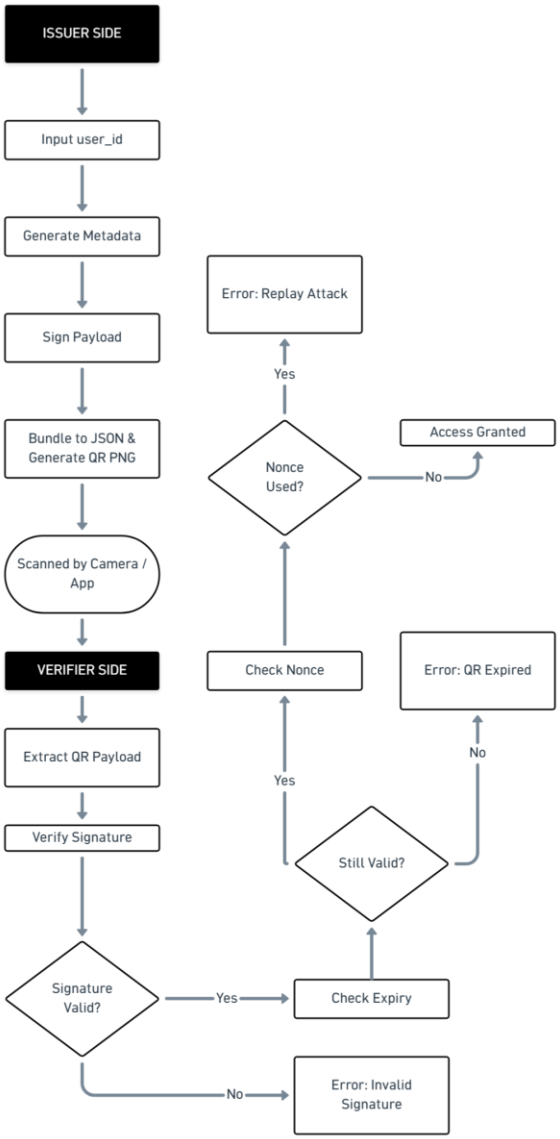

Fig. 1. Unified Token Generation and Verification Flowchart

A. Mathematical Operational Workflow

1. Key Initialization: The system initializes an elliptic curve group over a prime finite field using the standard NIST P-256 (SECP256R1) domain parameters. A random private key integer $d \in [1, n-1]$ is generated,

and the corresponding public key point is derived via scalar multiplication: $Q = d \cdot P$.

2. Payload Serialization: To generate a token, the issuer aggregates the user identifier, a high-precision UTC timestamp, a strict expiration boundary (TTL), and an 8-byte cryptographically secure hexadecimal nonce ($nonce \leftarrow secrets.token\_hex(8)$). The fields are strictly ordered using deterministic alphabetical sorting keys to prevent hash mismatches:

$$Payload = \{\text{"expires\_at"}, \text{"nonce"}, \text{"}timestamp\text{"}, \text{"}user\_id\text{"}\}$$

3. Cryptographic Signing: The serialized JSON string is hashed using SHA-256. The private key generates an asymmetric pair (r, s) according to standard ECDSA:

$$Signature = ECDSA.sign(SHA256(Payload), d)$$

The signature is encoded into a Base64 string and appended directly into the final QR dictionary.

B. *Anti-Replay Ledger Database Schema*

To enforce single-use execution boundaries, the server maintains an asset ledger in SQLite. The database scheme relies on a unique index restraint bound to the nonce block, completely blocking identical incoming token inputs.

```
CREATE TABLE qr_sessions (
    id INTEGER NOT NULL,
    user_id VARCHAR NOT NULL,
    nonce VARCHAR NOT NULL,
    created_at DATETIME NOT NULL,
    expires_at DATETIME NOT NULL,
    signature VARCHAR NOT NULL,
    is_used BOOLEAN,
    PRIMARY KEY (id),
    UNIQUE (nonce)
);
CREATE INDEX ix_qr_sessions_id ON qr_sessions (id);
```

IV. Implementation Details

The system implementation is written entirely in Python 3, using FastAPI as the web framework and the cryptography library for low-level primitive primitives.

A. *Core Cryptography Engine*

The signing and verification logic are abstracted into a static CryptoService class, guaranteeing cryptographic consistency during data operations:

```python
# app/services/crypto_service.py
import json, base64
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec

class CryptoService:
    @staticmethod
    def sign_payload(payload: dict) -> str:
        private_key = CryptoService.load_private_key()
        payload_json = json.dumps(payload, sort_keys=True).encode()
        signature = private_key.sign(
            payload_json,
            ec.ECDSA(hashes.SHA256())
        )
        return base64.b64encode(signature).decode()

    @staticmethod
    def verify_signature(payload: dict, signature: str) -> bool:
        public_key = CryptoService.load_public_key()
        payload_json = json.dumps(payload, sort_keys=True).encode()
        try:
            public_key.verify(
                base64.b64decode(signature),
                payload_json,
                ec.ECDSA(hashes.SHA256())
            )
            return True
        except Exception:
            return False
```

B. *Verification Pipeline Implementation*

When an image is processed via the web route, it passes through an automated validation chain implemented in the VerificationService:

```python
# app/services/verification_service.py
from datetime import datetime
from sqlalchemy.orm import Session
from app.models import QRSession
```

```python
from app.services.crypto_service import
CryptoService

class VerificationService:
    @staticmethod
    def verify_qr(payload: dict, db:
Session) -> dict:
        signature =
payload.pop("signature")
        if not
CryptoService.verify_signature(payload,
signature):
            return {"status": False,
"message": "Invalid Signature"}

        expires_at =
datetime.fromisoformat(payload["expires_at"
])
        if datetime.utcnow() > expires_at:
            return {"status": False,
"message": "QR Expired"}

        existing_nonce =
db.query(QRSession).filter(
            QRSession.nonce ==
payload["nonce"]
        ).first()

        if existing_nonce and
existing_nonce.is_used:
            return {"status": False,
"message": "Replay Attack Detected"}

        if existing_nonce:
            existing_nonce.is_used = True
            db.commit()

        return {"status": True, "message":
"Access Granted"}
```

C. *Web Application Interface and Operational Workflow*

To provide an intuitive workflow for issuers and verifiers, a frontend dashboard was developed using HTML5 templates, Jinja2 rendering engines, and Tailwind CSS styles. The web workflow consists of three primary views:

1. *Unified Operational Dashboard:* Upon launching the system, users are presented with a minimalist home screen, as illustrated in Fig. 2. This central interface provides distinct routing pathways to access either the generator terminal or the verifier check-gate.

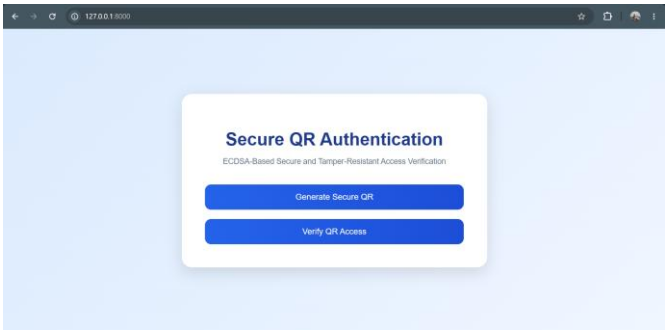

Fig. 2. FastAPI web application centralize home portal dashboard

2. *Token Generation Portal (Issuer Side):* When an administrator opens the generation module, the layout provides a clear configuration form. The issuer inputs the required metadata token into the User ID prompt. The application backend processes this data instantly by executing the core asymmetric key signing logic defined in the *CryptoService* class, outputting a high-contrast scannable QR matrix, as captured in Fig. 3.

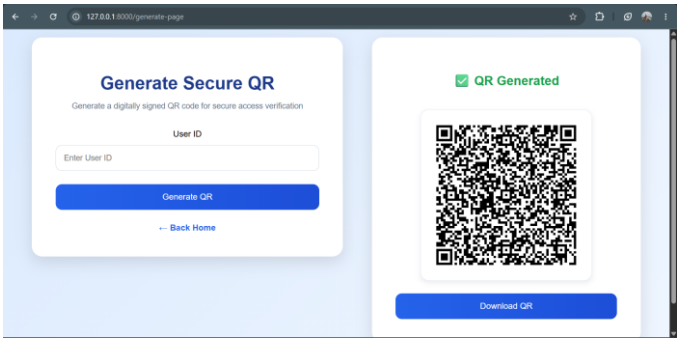

Fig. 3. Administrative generation terminal displaying the rendered assymetric QR matrix

3. *Token Verification Portal (Verifier Gate Side):* At physical access control barriers, security personnel leverage the verification endpoint to evaluate incoming matrix tokens. The interface provides an interactive upload drop-zone where file assets can be submitted for decoding. This interface triggers the underlying automated verification pipeline managed by the *VerificationService* class in real-time. The verification panel layout is illustrated in Fig. 4.

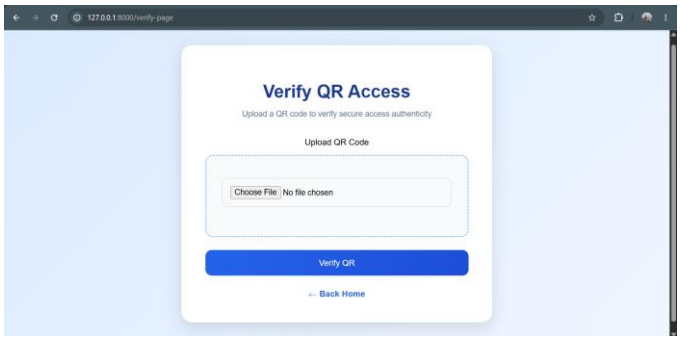

Fig. 4. Secure token verification portal interface acting as the optical scanning gate

V. Experimental Results and Performance Analysis

To evaluate the operational efficiency, speed, and defensive capabilities of the proposed system, several benchmarking experiments and threat simulation scenarios were executed on a local machine equipped with an Intel Core processor and an SQLite instance.

A. Computational Latency Benchmarks

The first experiment measured the execution speed of the critical cryptographic functions in the application layer. Each core routing process was executed repeatedly to determine accurate average latency values. These empirical processing metrics are formalized in Table II.

TABLE II. Computational Latency per Operation

Core Cryphtographic Process	Average Execution Latency (ms)	CPU Utilization (%)
ECDSA Key Pair Generation	1.420 ms	12.4%
Token Data Hashing & Signing	0.245 ms	4.1%
Optical Image Decoding	11.230 ms	18.2%
Database & Signature Verification	0.385 ms	5.3%

As detailed in Table II, the total latency introduced by the cryptographic backend (excluding the optical processing done by pyzbar) is less than 1 millisecond. The complete authentication chain takes approximately 12.26 milliseconds, validating the framework's capability to serve high-throughput physical check-gates smoothly.

B. Threat Simulation and Vulnerability Testing

To verify the system's runtime resilience against malicious vectors, 500 unauthorized access requests were programmatically simulated across three distinct threat categories.

1. *Data Tampering Attempts:* To simulate a real-world intercept-and-alter malicious attack vector, a legitimate QR code asset generated for a specific user profile (user_id: "18223021") was intercepted. The binary token image was decoded using an online open-source reader platform (zxing.org/w/decode) to extract the underlying JSON metadata string containing the identifiers, nonce, and original signature. The plaintext string was subsequently tampered with, modifying the identification string to target an administrative user account. This modified payload string was re-encoded into a fake QR code matrix via an online utility (qr-code-generator.com). When this altered asset was submitted to the gatekeeper validation pipeline, the application intercepted the mismatched cryptographic integrity values instantly, denying access with an Invalid Signature alert. The browser rejection for this simulated data tampering experiment is captured in Fig. 5.

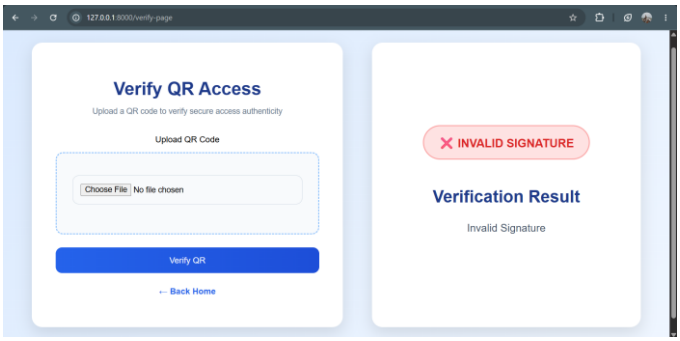

Fig. 5. Verification panel output rejecting a tampered payload submission

2. *Token Expiration Validation (TTL):* Legitimate signed tokens were withheld and uploaded after their 30-second Time-to-Live (TTL) constraint window passed. The validation engine successfully compared the Unix timestamp boundaries against the current execution time, outputting a QR Expired status response. This automated rejection interface is captured in Fig. 6.

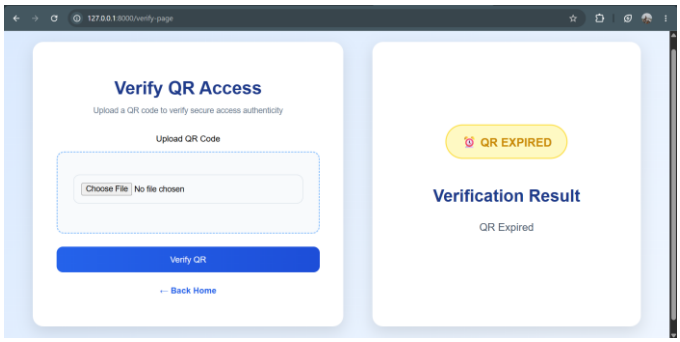

Fig. 6. Verification panel output response capturing token timeout isolation

3. *Replay Attack Invalidation:* A valid signed QR code matrix was uploaded to the gate. The database registered the record, logged the session metadata, and flagged the token's nonce as spent (is_used = True). The identical image file was immediately re-submitted within its active 30-second timeline window. The database lookup detected the duplicate nonce entry instantly, rendering a Replay Attack Detected error flag. The server intercept behavior is illustrated in Fig. 7.

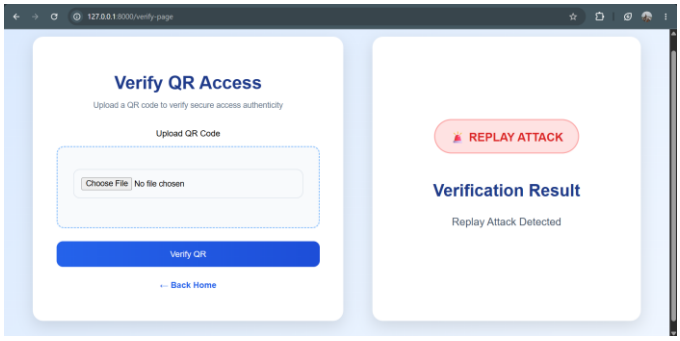

Fig. 7. Verification application interface intercepting a duplicate clone token

4. *Baseline Authorized Request:* When an unmodified, unexpired, and unique token asset was processed by the verification pathway, the database committed the state transaction smoothly, returning a clear approval, as illustrated in Fig. 8.

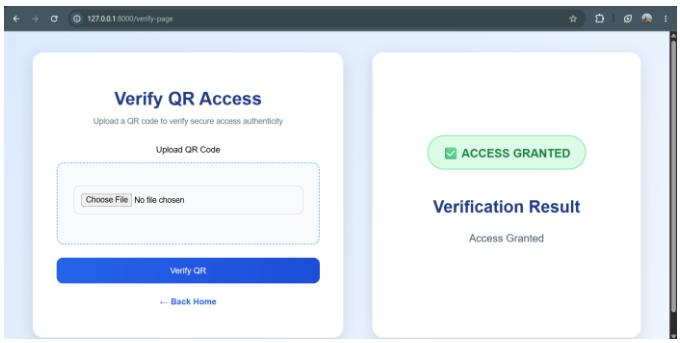

Fig. 8. Verification application dashboard displaying a succesful authentication

VI. Conclusion

This paper presents a secure and tamper-resistant access verification system designed to block credential spoofing, data tampering, and replay attacks. By building a functional backend with FastAPI and SQLite, the system successfully secures the QR code payload using a SHA-256 hash, unique random nonces, and expiration timestamps signed with ECDSA. Experimental results show that the cryptographic engine is highly efficient, validating digital signatures in less than a millisecond. The automated pipeline effectively detects and rejects tampered data (*Invalid Signature*), expired tokens (*QR Expired*), and duplicate assets (*Replay Attack Detected*) in real-time. Finally, the lightweight properties of the SECP256R1 curve keep the QR code clean and low-density, delivering an optimal balance between strong security and fast scanning speeds.

Source Code Link at Github

https://github.com/timothymarvine/secure-qr-auth

Video Link at Youtube

https://youtu.be/nEBENl4qA3U

Acknowledgment

The author would like to express the deepest praise and sincere gratitude to the Almighty God for the blessings and strength provided to complete this paper within the designated deadline. Special and profound gratitude is extended to Mr. Dr. Ir. Rinaldi Munir, M.T., for his invaluable guidance, insightful lectures, and the opportunity to write this research paper. The author also expresses heartfelt appreciation to his family for their endless support and encouragement throughout this academic endeavor. Finally, the author thanks all the classmates in the II4021 Cryptography course for their constant collaboration and supportive spirit during the semester.

References

[1] T. Wellem, Y. Nataliani, and A. Iriani, "Academic Document Authentication using Elliptic Curve Digital Signature Algorithm and QR Code," JOIV: Int. J. Inform. Visualization, vol. 6, no. 3, pp. 667-675, Sep. 2022. [Online]. Available: https://www.joiv.org/index.php/joiv/article/view/872

[2] M. Warasart and P. Kuacharoen, "Paper-based Document Authentication using Digital Signature and QR Code," in Proc. 4th Int. Conf. Computer Engineering and Technology (ICCET), 2012, pp. 314-318. [Online]. Available: https://www.researchgate.net/profile/Pramote-Kuacharoen/publication/267427243_Paper-based_Document_Authentication_using_Digital_Signature_and_QR_Code/links/54b663b70cf2bd04be32061d/Paper-based-Document-Authentication-using-Digital-Signature-and-QR-Code.pdf

[3] H. A. Ahmed and J. W. Jang, "Document Certificate Authentication System Using Digitally Signed QR Code Tag," in Proc. 12th Int. Conf. Ubiquitous Information Management and Communication (IMCOM '18), Jan. 2018, Art. no. 86. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3164541.3164586

[4] A. Khalique, K. Singh, and S. Sood, "Implementation of Elliptic Curve Digital Signature Algorithm," International Journal of Computer Applications, vol. 2, no. 2, pp. 21-27, May 2010. [Online]. Available:

https://www.academia.edu/33777089/Implementation_of_Elliptic_Curve_Digital_Signature_Algorithm

[5] W. Stallings, Cryptography and Network Security: Principles and Practice, 7th ed. Pearson, 2017. [Online]. Available: https://www.uoitc.edu.iq/images/documents/informatics-institute/Competitive_exam/Cryptography_and_Network_Security.pdf

[6] ZXing Project, "ZXing Decoder Online," zxing.org, 2026. [Online]. Available: https://zxing.org/w/decode.

[7] In some, "QR Code Generator," qr-code-generator.com, 2026. [Online]. Available: https://www.qr-code-generator.com.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Timothy Marvine - 18223021