A Comparative Analysis of HMAC-SHA256 and ECDSA as Security Mechanisms for Transfer Transactions in Digital Banking Environment

Wijaksara Aptaluhung - 18223088
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: wijaksaraaptaluhung@gmail.com , 18223088@std.stei.itb.ac.id

***Abstract* — Digital banking transactions demand robust cryptographic mechanisms to ensure integrity, authenticity, and non-repudiation of transfer operations. This paper presents a comparative analysis of two widely deployed cryptographic schemes: HMAC-SHA256 (Hash-based Message Authentication Code using SHA-256) and ECDSA (Elliptic Curve Digital Signature Algorithm using the P-256 curve). A simulated digital banking transaction environment is constructed in Python to implement and evaluate both mechanisms across multiple dimensions including execution time, key size, output size, and security properties. Experimental results demonstrate that HMAC-SHA256 achieves significantly lower computational overhead, producing MAC values approximately 283 times faster than ECDSA signatures per transaction, making it suitable for high-throughput internal systems. ECDSA, however, provides cryptographic non-repudiation and eliminates the shared-secret key distribution problem, making it more appropriate for open banking APIs and inter-institution transfers requiring regulatory compliance. A security analysis covering replay attacks, key compromise impact, and scalability further differentiates the two schemes. The paper concludes with practical guidance on when each mechanism should be applied in real-world digital banking environments, with reference to ISO 20022 and open banking standards.**

Keywords* — *HMAC-SHA256, ECDSA, digital banking, transaction security, message authentication code, digital signature, elliptic curve cryptography, non-repudiation

I. Introduction

A. Digital Banking Landscape

Digital banking has heavily transformed financial services, enabling millions of transfer transactions to occur every second globally. Yet, this unprecedented scale introduces significant security challenges: transaction data traversing insecure channels must be protected against tampering, replay attacks, and fraudulent repudiation. The consequences of a compromised transaction, whether through unauthorized modification or denial of execution by either party, can be financially inflicting and legally complex.

Cryptographic mechanisms sit at the core of secure transaction processing. Two families of algorithms are most relevant to transaction authentication: Message Authentication Codes (MAC), specifically HMAC-SHA256, and digital signature schemes, specifically the Elliptic Curve Digital Signature Algorithm (ECDSA). Both provide message integrity and authentication, but they differ in their trust model, computational characteristics, and security guarantees. HMAC-SHA256 relies on a shared secret key between communicating parties, while ECDSA uses an asymmetric key pair that allows public verification without exposing the signing key.

The choice between these mechanisms involves trade-offs. HMAC-SHA256 is computationally lightweight and well-suited for high-frequency internal transactions between trusted parties. On the other hand, ECDSA, being asymmetric, provides non-repudiation — an important open banking property — but at a higher computational cost. Understanding these trade-offs is essential for system architects designing secure banking infrastructure.

B. Problem Statement

The increasing adoption of both HMAC-SHA256 and ECDSA for transaction authentication in digital banking has created a practical challenge for system architects. While both mechanisms practice message integrity and authentication, they differ significantly in computational efficiency, trust model, and security guarantees. Existing literature often discusses these algorithms independently, but there is limited experimentally grounded guidance on selecting the most appropriate mechanism for banking transaction scenarios. Accordingly, this study seeks to address these research questions:

- How do HMAC-SHA256 and ECDSA compare in terms of computational performance, including key generation, signing (or MAC generation), and verification time, in a digital banking transaction context?

- How do HMAC-SHA256 and ECDSA differ with respect to critical security properties, including non-repudiation, resistance to replay attacks, and the impact of key compromise?
- Based on performance and security considerations, under what practical deployment scenarios should HMAC-SHA256 or ECDSA be preferred?

C. Objectives and Contributions

This study aims to investigate the suitability of HMAC-SHA256 and ECDSA for transaction authentication in digital banking systems through comprehensive performance and security analysis. The objectives of this paper are as follows:

- To experimentally evaluate and compare the computational performance of HMAC-SHA256 and ECDSA, based on the execution time of key generation, signing (or MAC generation), and verification operations.
- To analyze and compare the security characteristics of HMAC-SHA256 and ECDSA, especially on authentication, integrity, non-repudiation, replay attack resistance, and the consequences of key compromise.
- To formulate practical recommendations for selecting an appropriate authentication mechanism for digital banking transaction environments.

This study is also expected to contribute to the understanding and application of cryptographic authentication mechanisms in financial systems. The expected contributions of this paper are as follows:

- A Python-based banking transaction simulation that implements both HMAC-SHA256 and ECDSA using realistic transaction payloads for comparative evaluation.
- Experimental performance data obtained from 1,000 trial executions, providing quantitative insights into the computational costs of key generation, signing, and verification processes.
- A structured security comparison of HMAC-SHA256 and ECDSA across key authentication properties relevant to digital banking applications.
- Practical recommendations for selecting an appropriate authentication mechanism based on deployment requirements, with consideration of interoperability standards such as ISO 20022 and open banking frameworks.

II. Background and Technical Overview

A. Message Authentication Code and HMAC-SHA256

A Message Authentication Code (MAC) is a short piece of information used to confirm both the data integrity and the authenticity of a message. It requires a shared secret key known to both the sender and the receiver. Unlike hash functions alone, MACs provide authentication because only parties possessing the secret key can produce or verify a valid MAC [1].

HMAC (Hash-based Message Authentication Code), standardized in RFC 2104, constructs a MAC from a cryptographic hash function and a secret key. Given a message M and key K, HMAC is defined as:

$$HMAC(K, M) = H((K \oplus opad) \,||\, H((K \oplus ipad) \,||\, M))$$

where H is the hash function, *opad* is the outer padding constant (0x5c repeated), *ipad* is the inner padding constant (0x36 repeated), and || denotes concatenation [2]. When H = SHA-256, the construction is called HMAC-SHA256, producing a 256-bit MAC.

The verification flow is symmetric: the receiver independently computes *HMAC(K, M)* using the shared key and compares it with the received MAC. A match confirms both integrity (the message has not been not altered) and authenticity (the sender possesses the shared key). HMAC-SHA256 is widely used in TLS handshakes, JWT signing, and banking API authentication headers [3].

B. Digital Signatures and ECDSA

A digital signature is an asymmetric cryptographic construct that binds a message to the signer's private key, while allowing any party with the corresponding public key to verify it. Unlike MAC, digital signatures provide non-repudiation, meaning that the signer cannot later deny having signed a message, because only they possess the private key [4].

ECDSA (Elliptic Curve Digital Signature Algorithm), standardized in FIPS 186-4, operates over an elliptic curve group defined by an equation:

$$y^2 = x^3 + ax + b \ (mod\ p)$$

The P-256 curve (also known as secp256r1) is the most used curve, defined over a 256-bit prime field and providing approximately 128 bits of security [5].

The signing process generates a unique digital signature, binding the transaction data to the signer's private key. Given a message M, the process proceeds in a consequent order: (1) compute the hash $e = H(M)$, producing a fixed-length representation of the transaction data; (2) select a cryptographically random integer k from $[1, n-1]$, where n is the curve order; (3) compute the elliptic curve point $(x_1, y_1) = k \times G$ where G is the base point, aimed to derive an ephemeral public point; (4) compute $r = x_1 \ mod\ n$, forming the first component of the signature; (5) compute $s = k^{-1}(e + r \cdot d) \ mod\ n$ where d is the private key, aimed to bind the message has and the private key to produce the second signature component. The signature is the pair (r, s) [5].

Verification allows any party possessing the signer's public key $Q = dG$ to confirm that the signature is generated by the corresponding private key and that the message has not been altered, proceeding in the following order: (1) computing $e = H(M)$; (2) computing $w = s^{-1} \ mod\ n$, used to recover the

relationship established during signing; (3) computing $u_1 = e \cdot w \bmod n$ and $u_2 = r \cdot w \bmod n$, serving as weighting factors for the subsequent elliptic curve operations; (4) computing the elliptic curve point $(x_1, y_1) = u_1 \cdot G + u_2 \cdot Q$; (5) accepting signature if $r \equiv x_1 \pmod n$ [5]. The security of ECDSA relies on the computational hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP).

C. Fundamental Differences

The key architectural distinction between HMAC-SHA256 and ECDSA lies in their trust models. HMAC-SHA256 is a symmetric scheme, using the same secret key in both parties, meaning both the sender and receiver are equally trusted or even equally suspected if the key is compromised. Meanwhile, ECDSA is asymmetric; the private key is held exclusively by the signer, and verification uses a publicly distributed key, cleanly separating the signing and verification roles [4].

D. Related Work

Barker et al. provide NIST guidance on key management and recommend HMAC for session-level authentication in closed environments and digital signatures for legal-grade non-repudiation [6]. Pornin analyzed ECDSA's security properties and implementation considerations, noting the criticality of random nonce k in preventing key extraction attacks [7]. Research by Bos et al. on constant-time elliptic curve operations highlighted the implementation-level security risks in naive ECDSA deployments [8]. Prior comparative studies focused on RSA versus ECDSA but did not address the MAC versus signature dichotomy in the banking context [9]. Existing comparative studies primarily focus on signature algorithms such as RSA and ECDSA, leaving the trade-offs between symmetric authentication mechanisms and digital signatures in banking applications largely unexplored. This study addresses this limitation by experimentally comparing HMAC-SHA256 and ECDSA using realistic digital banking transaction scenarios.

III. System Design and Implementation

A. Simulated Banking Transaction Model

To provide a realistic evaluation context, a simulated digital banking transfer transaction model is constructed. Each transaction payload is represented as a JSON object containing several integral fields: transaction ID (UUID v4), sender account number, receiver account number, transfer amount, currency code (ISO 4217), timestamp (ISO 8601), and a sequence number used for replay protection. An example transaction structure is built as follows:

```
{"txn_id": "uuid-v4", "sender": "1234567890",
"receiver": "0987654321", "amount": 5000000,
"currency": "IDR", "timestamp": "2026-06-
01T10:30:00Z", "seq": 1042}
```

The payload is serialized to a canonical JSON string using the following helper:

```
def serialize(txn: dict) -> bytes:
    return json.dumps(txn, sort_keys=True,
        separators=(',',':')).encode('utf-8')
```

Keys are sorted alphabetically and whitespace is stripped to ensure deterministic byte sequences before the cryptographic processing is done. This is critical for reproducible MAC and signature computation across sender and receiver.

B. Implementation of HMAC-SHA26

HMAC-SHA256 is implemented using Python's standard library *hmac* and *hashlib* modules, which provide a FIPS-compliant implementation. The implementation workflow consists of three integrated phases: key generation, MAC computation, and MAC verification.

Key generation uses *os.urandom(32)* to produce a 256-bit cryptographically secure random key. MAC computation passes the serialized payload directly to the HMAC constructor, with implementation as follows:

```
key = os.urandom(32)
mac = hmac.new(key,payload,hashlib.sha256).digest()
```

Verification recomputes the MAC on the received payload and performs a constant-time comparison to prevent timing side-channel attacks, with implementation as follows:

```
def verify_mac(key, payload, received_mac):
    expected = hmac.new(key, payload,
    hashlib.sha256).digest()
    return hmac.compare_digest(expected,
    received_mac)
```

The sequence number embedded in each transaction payload provides replay attack resistance; the receiver maintains a monotonic counter and rejects any transaction whose *seq* value is not strictly greater than the last accepted value.

C. Implementation of ECDSA

ECDSA is implemented using the cryptography library (v42.x), which exposes the P-256 curve via ec.SECP256R1() and uses OpenSSL as its backend, providing constant-time scalar multiplication to mitigate timing attacks [10].

Key generation produces a private key from which the public key is derived, with implementation as follows:

```
priv_key = ec.generate_private_key (ec.SECP256R1())
public_key  = private_key.public_key()
```

Signing passes the serialized payload to the ECDSA signer, producing a DER-encoded signature, with implementation as follows:

```
signature = private_key.sign(payload,
ec.ECDSA(hashes.SHA256()))
```

Verification raises an *InvalidSignature* exception on failure, which is caught to return a boolean result, with implementation as follows:

```
def verify_ecdsa(public_key, payload, signature):
    try:
```

```
        public_key.verify(signature,          payload,
        ec.ECDSA(hashes.SHA256()))
        return True
    except InvalidSignature:
        return False
```

The public key is serialized in DER format for storage and transmission. The private key must remain exclusively with the transaction originator; in this case, the bank's core banking system or the customer's secure hardware element.

D. Implementation Tools and Environment

All implementations were executed on a machine running Ubuntu 22.04 LTS, Python 3.11.9, and the *cryptography* library v42.0.7. Timing measurements used *time.perf_counter()* with nanosecond resolution. Each benchmark was run for 1,000 iterations and mean values were recorded. The complete source code is publicly available at: https://github.com/wiijaakk/II4021-Kriptografi-Makalah.

IV. Experimental Analysis

A. Performance Metrics

The quantitative results from 1,000-run benchmarks for both schemes are summarized in Table I. All timing values represent arithmetic means. The results confirm that HMAC-SHA256 is substantially faster across all phases: key generation is approximately 900 times faster, MAC computation is approximately 283 times faster, and verification is approximately 620 times faster than their ECDSA counterparts, as shown in Table I.

TABLE I. Performance Comparison of HMAC-SHA256 vs. ECDSA (P-256)

Parameter	***HMAC-SHA256***	***ECDSA (P-256)***
Key Generation Time	~0.0003 ms	~0.06 ms
MAC / Sign Time	~0.006 ms	~0.12 ms
Verify Time	~0.006 ms	~0.15 ms
Key Size	256 bits	256 bits (priv)
Output Size	256 bits	512 bits
Non-repudiation	No	Yes
Key Distribution	Shared secret	Public key infrastructure
Replay Attack Resistance	With nonce/seq	With nonce/seq

From a computational perspective, HMAC-SHA256 consistently outperforms ECDSA across measured operations. The benchmark results indicate that key generation is approximately 900 times faster, MAC generation is approximately 283 times faster, and verification is approximately 620 times faster than the corresponding ECDSA operations. This performance gap is expected due to the fundamental computational differences between the two schemes: HMAC-SHA256 primarily relies on efficient hash computations, meanwhile ECDSA requires computationally intensive elliptic curve point multiplications. At the P-256 security level, each multiplication involves a sequence of point doubling and point addition operations, resulting in significantly higher execution costs.

In terms of key and output sizes, both schemes provide an equivalent 256-bit security level through a 256-bit secret or private key. However, the authentication output differs; HMAC-SHA256 produces a fixed 256-bit (32-byte) message authentication code, whereas an ECDSA signature typically occupies 71–72 bytes in DER encoding. Although the ECDSA signature is more than twice as large, the additional storage and transmission overhead remains relatively small for modern banking networks and is generally acceptable for inclusion in transaction headers or metadata fields.

The qualitative security properties further highlight the different deployment models of the two mechanisms. HMAC-SHA256 relies on a shared secret key, making it highly efficient for communication between trusted parties but incapable of providing non-repudiation, as both parties possess the same key. In contrast, ECDSA employs an asymmetric key pair that enables public verification while preserving the secrecy of the private signing key, providing non-repudiation. Both mechanisms can effectively resist replay attacks when integrated with freshness mechanisms such as nonces or sequence numbers. However, their key management requirements differ significantly, with HMAC requiring secure shared-secret distribution and ECDSA depending on a public key infrastructure (PKI).

B. Security Analysis

Table II compares the security properties of HMAC-SHA256 and ECDSA across dimensions relevant to digital banking transaction authentication. Both schemes provide message integrity and message authentication, and both require additional mechanisms such as nonces or sequence numbers to prevent replay attacks. However, they differ in terms of non-repudiation, key compromise impact, and scalability, reflecting their fundamentally different trust models.

TABLE II. Security Properties Comparison

Security Property	***HMAC-SHA256***	***ECDSA (P-256)***
Message Integrity	Yes	Yes
Message Authentication	Yes (shared key)	Yes (public key)
Non-repudiation	No	Yes
Forward Secrecy	No	No (native)
Key Compromise Impact	Both parties affected	Only owner affected
Scalability (n parties)	$O(n^2)$	O(n)

The most operationally significant distinction is non-repudiation. HMAC-SHA256, being a symmetric-key mechanism, cannot provide non-repudiation because both sender and receiver share the same secret key; consequently, either party could have generated the MAC. This creates an evidentiary limitation during dispute resolution, as a bank cannot cryptographically prove that a customer authorized a transaction using HMAC authentication alone.

In contrast, ECDSA provides strong non-repudiation because the private signing key is exclusively controlled by the signer. A valid digital signature over a transaction payload can only be generated by the holder of the corresponding private key, creating a cryptographically verifiable audit trail that is suitable for legal and regulatory purposes.

Regarding replay attack resistance, neither scheme provides inherent protection against replay attacks at the cryptographic layer. A previously captured MAC or digital signature remains valid if the identical payload is retransmitted. Consequently, both mechanisms must be combined with guarantees such as sequence numbers, timestamps, or nonces. In the implementation presented in this study, replay protection is achieved using embedded sequence numbers, while production banking systems typically supplement this approach with timestamp validation windows and server-side idempotency keys.

The impact of key compromise also differs significantly between the two schemes. Because HMAC-SHA256 relies heavily on a shared secret, disclosure of that key compromises the authenticity of communications between all parties using it and may expose both historical and future transactions until the key is replaced. In contrast, compromise of an ECDSA private key affects only the corresponding key owner. The compromised key can be revoked and replaced independently through the public key infrastructure, minimizing the impact on other participants in the system.

Scalability represents another important consideration for large-scale financial ecosystems. In a network consisting of n participants, HMAC-SHA256 requires $O(n^2)$ shared keys because each communicating pair must maintain a distinct secret key. By comparison, ECDSA requires only O(n) key pairs, with each participant maintaining a single public-private key pair that can be verified by all others. This characteristic makes ECDSA significantly more scalable for open banking environments involving numerous financial institutions and third-party service providers.

V. Discussion

A. When to Use HMAC-SHA256 in Banking

HMAC-SHA256 is most suitable for transaction authentication in environments where all communicating aspects operate under the same organizational trust boundary and can securely share symmetric keys. In these conditions, the algorithm offers extremely low computational overhead while also providing strong guarantees of message integrity and authentication. As demonstrated in the performance evaluation, its execution time is much lower than that of ECDSA, making it particularly advantageous for systems processing a high volume of transactions with strict latency requirements.

The characteristics this method has make HMAC-SHA256 well suited for internal banking applications, such as authentication between microservices, communication between core banking systems and payment processors over private networks, session token validation for internet banking platforms, and real-time payment infrastructures where sub-millisecond processing times are essential. In these scenarios, legal non-repudiation is unnecessary because all responsible entities belong to the same trusted organization and are governed by centralized security policies.

The primary limitation of HMAC-SHA256 is in its key management model. Because authentication depends on a shared secret, secure key distribution, storage, and rotation must be carefully managed to prevent unauthorized access. Nevertheless, this challenge is manageable within a single institution through the use of Hardware Security Modules (HSMs) or centralized Key Management Services (KMS). Banking institutions are heavily urged to implement these services to ensure the security of their internal keys.

B. When to Use ECDSA in Banking

ECDSA is more appropriate in environments where transaction authenticity must be independently verifiable and legally attributable to specific entities. Unlike HMAC-SHA256, which relies on a shared secret, ECDSA uses asymmetric cryptography, allowing signatures to be verified using a public key while keeping the private signing key confidential. This property enables non-repudiation, ensuring that a signer cannot plausibly deny having authorized a transaction.

Consequently, ECDSA is well suited for open banking ecosystems, inter-bank payment networks, and customer-initiated high-value transactions where regulatory compliance and auditability are essential. It is also widely used in blockchain-based settlement systems, where digital signatures provide the foundation for transaction authorization and ownership verification.

The adoption of ECDSA is further supported by modern financial standards. ISO 20022 includes provisions for digital signature metadata within financial messages, facilitating the integration of asymmetric authentication mechanisms into payment workflows. Likewise, OpenID Connect and OAuth 2.0 frameworks commonly used in open banking APIs support ES256 (ECDSA with the P-256 curve), enabling secure and publicly verifiable authentication. At the same time, it also maintains interoperability across multiple institutions.

C. Trade-offs: Speed vs Non-Repudiation

The comparative analysis demonstrates that the selection between HMAC-SHA256 and ECDSA should not be viewed as identifying a universally superior algorithm, instead as choosing the mechanism that best satisfies the operational and security requirements of a given application in its own specific context.

HMAC-SHA256 prioritizes computational efficiency, making it highly suitable for high-throughput environments where trusted parties exchange a large volume of messages and low latency is critical. However, its reliance on shared secret keys prevents it from providing non-repudiation and complicates key management as the number of communicating entities increases.

Conversely, ECDSA introduces greater computational overhead due to the complexity of elliptic curve arithmetic but offers significant advantages in terms of non-repudiation, independent public verification, and scalability through public key infrastructure. These characteristics make it particularly

valuable in distributed financial ecosystems involving multiple independent organizations.

As a result, many modern banking systems adopt a hybrid authentication architecture. HMAC-SHA256 is used for session-level or internal service authentication to maximize throughput, while ECDSA is reserved for transaction-level authorization requiring legal accountability or regulatory compliance. Implementing this approach will leverage the strengths of both mechanisms while also mitigating their limitations.

D. Real-World Applicability

The findings of this study are consistent with current practices in modern financial infrastructures. In Indonesia, the Bank Indonesia SNAP (Standar Nasional Open API Pembayaran) specification requires partner authentication through asymmetric digital signatures, reflecting the importance of non-repudiation and secure inter-organizational communication in open banking ecosystems. Similarly, international payment frameworks such as SWIFT gpi benefit from message-level authentication mechanisms that support cryptographically verifiable transaction authorization.

From an implementation perspective, the performance differences observed in this study should be interpreted within the context of the underlying hardware environment. While HMAC-SHA256 naturally achieves significantly lower computational latency on general-purpose processors, dedicated cryptographic hardware such as Hardware Security Modules (HSMs) can accelerate elliptic curve operations and reduce the performance gap for ECDSA. Consequently, organizations selecting between the two mechanisms should consider not only algorithmic efficiency but also infrastructure capabilities, regulatory obligations, transaction volume, and interoperability requirements.

VI. Conclusion

This paper presented an empirical comparison of HMAC-SHA256 and ECDSA as transaction authentication mechanisms for digital banking systems. Using a Python-based banking transaction simulation and 1,000 experimental runs, the study evaluated both algorithms from the perspectives of computational performance and security characteristics.

The experimental results demonstrate that HMAC-SHA256 significantly outperforms ECDSA in terms of computational efficiency, achieving substantially lower execution times for key generation, authentication generation, and verification. These findings make HMAC-SHA256 particularly well suited for high-throughput, latency-sensitive environments where communicating entities operate within a shared trust boundary. In contrast, ECDSA incurs greater computational overhead but provides critical security properties such as non-repudiation, improved scalability through asymmetric key management, and stronger resilience to key compromise. These characteristics make ECDSA a more appropriate choice for customer-facing transactions, inter-institution communication, and open banking ecosystems that require independently verifiable transaction authorization.

Based on these findings, this paper recommends that financial institutions adopt a context-aware authentication strategy — a hybridization of both mechanisms — rather than relying exclusively on a single cryptographic mechanism. HMAC-SHA256 should be prioritized for internal service-to-service communication and other high-volume transaction environments where computational efficiency and low latency are paramount. Conversely, ECDSA should be employed for transactions involving external stakeholders, regulatory compliance, or legal accountability, where non-repudiation and public verifiability are essential. For many modern banking architectures, a hybrid deployment that combines HMAC-SHA256 for internal authentication with ECDSA for transaction-level signing represents a practical balance between performance, security, and scalability.

Although this study provides experimentally grounded insights into the trade-offs between HMAC-SHA256 and ECDSA, the evaluation was conducted within a simulated software environment and focused on two widely adopted authentication mechanisms. Future research may extend this work by evaluating hardware-accelerated implementations, large-scale distributed banking systems, and emerging post-quantum digital signature schemes, such as CRYSTALS-Dilithium, to assess their feasibility for securing next-generation digital banking infrastructures against evolving cryptographic threats.

Video Link at YouTube

Video on YouTube can be accessed through this link:

https://youtu.be/Ui4r9e7eUR8

References

[1] M. Bellare, R. Canetti, and H. Krawczyk, "Keying Hash Functions for Message Authentication," in Advances in Cryptology – CRYPTO 1996, Lecture Notes in Computer Science, vol. 1109, N. Koblitz, Ed. Springer, Berlin, Heidelberg, 1996, pp. 1–15. doi: 10.1007/3-540-68697-5_1.

[2] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, Internet Engineering Task Force (IETF), Feb. 1997. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2104

[3] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, IETF, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8446

[4] W. Stallings, Cryptography and Network Security: Principles and Practice, 8th ed. Pearson Education, 2022.

[5] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)," FIPS PUB 186-4, U.S. Department of Commerce, Jul. 2013. [Online]. Available: https://csrc.nist.gov/publications/detail/fips/186/4/final

[6] E. Barker, "Recommendation for Key Management: Part 1 – General," NIST Special Publication 800-57 Part 1 Rev. 5, NIST, May 2020. doi: 10.6028/NIST.SP.800-57pt1r5.

[7] T. Pornin, "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)," RFC 6979, IETF, Aug. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6979

[8] J. W. Bos, C. Costello, P. Longa, and M. Naehrig, "Selecting Elliptic Curves for Cryptography: An Efficiency and Security Analysis," Journal of Cryptographic Engineering, vol. 6, no. 4, pp. 259–286, Nov. 2016. doi: 10.1007/s13389-015-0097-y.

[9] A. K. Lenstra and E. R. Verheul, "Selecting Cryptographic Key Sizes," Journal of Cryptology, vol. 14, no. 4, pp. 255–293, 2001. doi: 10.1007/s00145-001-0009-4.

[10] P. Architects, "pyca/cryptography: Cryptographic Recipes and Primitives for Python," Version 42.0.7, Python Cryptographic Authority, 2024. [Online]. Available: https://cryptography.io

[11] SWIFT, "SWIFT gpi: Global Payments Innovation," SWIFT, 2023. [Online]. Available: https://www.swift.com/our-solutions/swift-gpi

ORIGINALITY STATEMENT

I hereby declare that this paper is my own original work and is neither an adaptation nor a translation of another person's work, and that it does not contain any form of plagiarism.

Bandung, 19 Juni 2026

Wijaksara Aptaluhung
18223088