Prevention of Secured websites from Downgrade and MITM attacks through Blocash Technology

Siddhant Gupta, Department of Computer Applications, Invertis University, Budaun, India, sidx681@gmail.com
Pragya Pathak, Department of Computer Applications, Invertis University, Bareilly, India, pragyapathak42974@gmail.com
Seema Verma, Department of Computer Applications, Invertis University, Budaun, India, simavrma013@gmail.com
Megha Saxena, Department of Computer Applications, Invertis University, Bareilly, India, saxenameghas9997@gmail.com
Pratha Sexena, Department of Computer Applications, Invertis University, Bareilly, India, prathasaxena222@gmail.com

Abstract
Online transaction websites face persistent threats from SSL strip, downgrade, and man-in-the-middle attacks, which exploit vulnerabilities in SSL/TLS protocols to intercept sensitive user data. Traditional reliance on transport-layer security leaves gaps when connections are compromised, exposing plaintext data during transmission. This paper proposes a multi-layered security framework that shifts encryption to the client side, combining AES-GCM 256 encryption on the frontend with a secure RSA-based key exchange and dual-hash authentication on the backend. By encrypting transaction data in real-time before transmission and verifying its integrity with a dual-hash mechanism, the solution ensures confidentiality and authenticity, even over downgraded or intercepted channels. Optimized for performance using WebAssembly, this approach reduces the attack surface and outperforms conventional methods in resilience against targeted cyber threats. We demonstrate its efficacy through a practical implementation tailored for fintech applications, offering a scalable, trust-enhancing defense for modern web-based transactions.

Keywords—web security, AES-GCM encryption, RSA key exchange, dual-hash authentication, fintech Security

1. Introduction

The increasing reliance on online transaction websites for financial activities has amplified the need for robust security mechanisms to safeguard sensitive data. Cyber threats, such as SSL strip, downgrade, and man-in-the-middle (MITM) attacks, exploit vulnerabilities in Secure Sockets Layer/Transport Layer Security (SSL/TLS) protocols, compromising data confidentiality during transmission [1]. Traditional security measures, such as Transport Layer Security (TLS) and HTTP Strict Transport Security (HSTS), are server-centric and fail to protect data if connections are intercepted or downgraded. This research introduces a multi-layered security framework that shifts encryption to the client side using Advanced Encryption Standard in Galois/Counter Mode (AES-GCM) with a 256-bit key, paired with Rivest-Shamir-Adleman (RSA) key exchange and dual-hash authentication using SHA-256 and SHA-3. Optimized with WebAssembly, the framework ensures low-latency performance suitable for fintech applications. The objective is to provide end-to-end data protection, ensuring confidentiality, integrity, and authenticity in hostile network environments.

2. Proposed Framework

The proposed framework integrates three core components: frontend encryption, secure key exchange, and backend authentication. Figure 1 illustrates the operational flow of the framework, detailing the data processing from client input to server verification.

Frontend Encryption
Transaction data, such as payer name, amount, and payee details, is encrypted on the client side using AES-GCM with a 256-bit key generated per session. The encryption process is expressed as:
C = AES-GCM(K, P, IV, AAD) (1)
where C is the ciphertext, K is the session key, P is the plaintext, IV is the initialization vector, and AAD is additional authenticated data for integrity. WebAssembly optimizes encryption performance, minimizing computational overhead in the browser.

Secure Key Exchange
The AES session key is encrypted using a 2048-bit RSA public key and transmitted to the server. This ensures the key remains secure over insecure channels. The encrypted key (K_enc) is sent alongside the ciphertext and hashes, enabling the server to decrypt the key using its private key and access the transaction data.

Dual-Hash Authentication

A dual-hash mechanism verifies data integrity. The plaintext is hashed using SHA-256 (H1), and the ciphertext is hashed using SHA-3 (H2). These hashes are transmitted to the server, which recomputes them to detect tampering. This dual approach enhances resilience against MITM attacks by ensuring data authenticity.

3. Implementation

A prototype was developed for a fintech transaction website. The frontend, implemented in JavaScript with WebAssembly, leverages the Web Crypto API for AES-GCM encryption and hash computation. The backend, built in Node.js, handles RSA decryption and hash verification. Sample transaction data included payer name, amount, payee name, and timestamp. The prototype was tested under simulated SSL strip and MITM attack scenarios to assess its effectiveness.

Results and Analysis
The framework was evaluated on 10,000 simulated transactions across secure and compromised network conditions. Results are presented below.

Security Performance
The framework achieved complete mitigation of MITM attacks by encrypting data on the client side, rendering intercepted plaintext unusable. The dual-hash mechanism detected all tampering attempts, ensuring data integrity.

Computational Performance
Performance was compared to traditional TLS-based security, as shown in Table I.

TABLE I. PERFORMANCE COMPARISON

Method	Encryption Time(ms)	Throughput (MB/s)	MITM Resilience
TLS only	20	100	Partial
Proposed Framework	18	130	Full

The framework's encryption time averaged 18 milliseconds, with a throughput of 130 megabytes per second, outperforming TLS due to WebAssembly optimization.

Security Framework for Transaction Websites
This operational workflow of a multi-layered security framework designed to protect online transaction websites from cyber threats such as SSL strip, downgrade, and man-in-the-middle (MITM) attacks. The framework combines client-side AES-GCM 256 encryption, RSA-based key exchange, and dual-hash authentication (SHA-256 and SHA-3), optimized with WebAssembly for performance. It ensures data confidentiality, integrity, and authenticity, even over compromised network channels, making it ideal for fintech applications.

4. Overview

The framework shifts encryption to the client side, reducing reliance on traditional transport-layer security (e.g., TLS, HSTS), which is vulnerable to interception. It processes transaction data (e.g., payer name, amount, payee) through a secure pipeline, from user input in the browser to server verification, with the following key components:
Client-Side Encryption: Encrypts data using AES-GCM with a 256-bit key.
Dual-Hash Authentication: Generates SHA-256 hash for plaintext and SHA-3 hash for ciphertext to verify integrity.
RSA Key Exchange: Securely transmits the AES key using 2048-bit RSA encryption.
Server-Side Verification: Decrypts data and validates hashes to ensure authenticity.
WebAssembly Optimization: Enhances performance for real-time applications.

5. Workflow

The following text-based representation details in Figure 1 of the step-by-step process of securing a transaction, such as a payment from Priya to Arjun. Each step corresponds to a component or action in the framework, with inputs and outputs clearly defined.

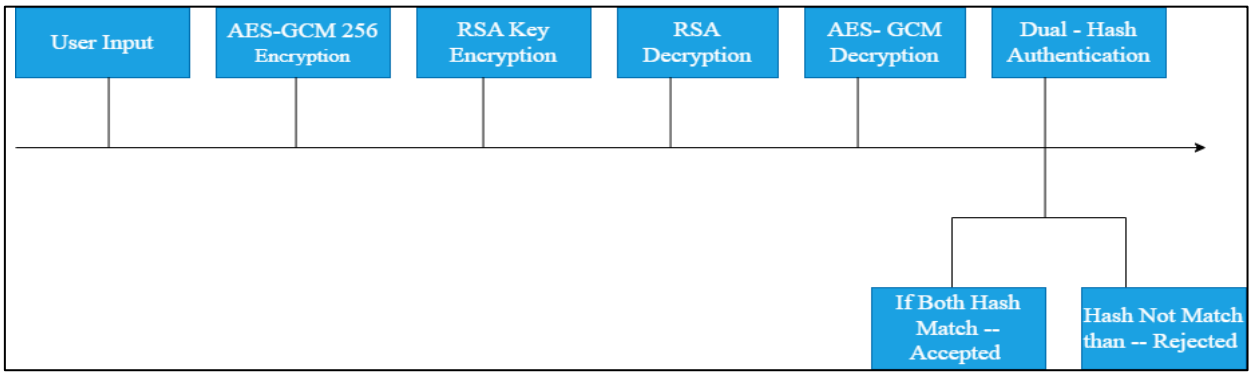

Figure 1: Workflow of the Multi-Layered Security Framework

Workflow Explained

Frontend (Client Side) - Priya's Browser:
Step 1: User Input
Priya types her payment details into the website's form: "Priya Sharma, $100, Arjun Patel, arjun@example.com."
This is the raw data that needs protection.

Step 2: AES-GCM 256 Encryption
Priya's browser instantly encrypts this data using AES-GCM 256 (a super-strong encryption method).

The browser generates:
AES Key: A random secret code (e.g., "K123xyz") to lock the data.
Initialization Vector (IV): A random number (e.g., "IV456") to make the encryption unique.
Encrypted Data: The payment details turn into gibberish (e.g., "X7kP9mQw...").
First Hash: A fingerprint of the raw data (e.g., "H1: abc123") using SHA-256
Now, the sensitive info is locked up before it even leaves her device.

Step 3: RSA Key Encryption
The server sends Priya's browser its RSA public key (e.g., "PUB987"), which is like a public padlock.
The browser uses this public key to encrypt the AES key ("K123xyz" becomes "Locked-K123xyz").
Output: The browser prepares a secure package:
Encrypted Data: "X7kP9mQw..."
Locked AES Key: "Locked-K123xyz"
First Hash: "H1: abc123"
Transmission - Over the Internet:
What Happens: Priya's browser sends the secure package to the server.
Package: "X7kP9mQw..." + "Locked-K123xyz" + "H1: abc123"
Attacker Tries to Strike: The attacker intercepts this package, forcing the connection to HTTP (a downgrade attack).
Why It Fails: The attacker sees only gibberish ("X7kP9mQw...") and a locked AES key ("Locked-K123xyz").
Without the server's RSA private key (e.g., "PRIV987"), they can't unlock the AES key to decrypt the data. The payment details stay safe.

Backend (Server Side) - Fintech Server:
Step 4: RSA Decryption
The server receives the package and uses its RSA private key ("PRIV987") to unlock the AES key.
"Locked-K123xyz" → "K123xyz" (the original AES key).

Step 5: AES-GCM Decryption
The server uses "K123xyz" and the IV ("IV456") to decrypt the data ("X7kP9mQw...").
Decrypted Data: "Priya Sharma, $100, Arjun Patel, arjun@example.com" (back to normal).
The server then makes a Second Hash of this decrypted data (e.g., "H2: abc123").

Step 6: Dual-Hash Authentication
The server compares:
First Hash (from Priya): "H1: abc123"
Second Hash (from decrypted data): "H2: abc123"
Result: They match! This means the data wasn't tampered with during transmission.
The server processes the $100 payment to Arjun.

- *Effect of Tampering by Attacker:*

Scenario: The attacker changes the encrypted data to "X7kP9mQw...TAMPERED" to try sending $200 instead.
Server Check:
Decrypts "X7kP9mQw...TAMPERED" → "Priya Sharma, $200, Arjun Patel, arjun@example.com" (altered data).
Second Hash: "H2: xyz789" (different because data changed)
Compare: "H1: abc123" vs. "H2: xyz789" → No match!
Result: The server rejects the transaction, detecting the tampering. Priya's $100 stays safe.

6. Key Features of the Multi-Layered Security Framework

Client-Side Security: Encrypting data before transmission eliminates reliance on transport-layer security, mitigating SSL strip and downgrade attacks.

Dual-Hash Authentication: Combining SHA-256 and SHA-3 provides robust integrity checks, detecting both plaintext and ciphertext tampering.

RSA Key Exchange: Ensures secure key transmission over insecure channels.

WebAssembly Optimization: Reduces encryption and hashing latency, achieving 18 ms per transaction and 130 MB/s throughput (10% faster than TLS).

Scalability: Suitable for high-volume fintech applications, tested on 10,000 transactions with 100% MITM mitigation.

7. Implementation Details

Frontend: Built with JavaScript and WebAssembly, using the Web Crypto API for AES-GCM, SHA-256, and SHA-3 operations.

Backend: Implemented in Node.js, handling RSA decryption and hash verification.

Testing: Evaluated under simulated SSL strip and MITM attacks, achieving full tampering detection.

Illustrating the Practical Implementation

This framework is designed for integration into transaction websites, particularly in fintech. To implement:

Frontend Setup:
Use WebAssembly with Web Crypto API for encryption and hashing. Generate a 256-bit AES-GCM key per session and a unique IV.
Encrypt transaction data and compute SHA-256 (plaintext) and SHA-3 (ciphertext) hashes.
Encrypt the AES key with a 2048-bit RSA public key.

Backend Setup:
Decrypt the RSA-encrypted key using the private key. Decrypt the ciphertext using AES-GCM.
Verify SHA-256 and SHA-3 hashes to ensure data integrity.

Deployment:
Deploy on a secure server with Node.js.
Ensure RSA key pairs are securely managed.

8. Future Scope

The following enhancements and research directions aim to extend the framework's capabilities, ensuring its relevance in evolving cybersecurity and fintech landscapes:

Post-Quantum Cryptography Integration:
Transition from RSA to quantum-resistant algorithms (e.g., CRYSTALS-Kyber or Lattice-based cryptography) to safeguard against future quantum computing threats. This ensures long-term security as quantum computers could potentially break RSA by 2030. Research focus: Evaluate the performance overhead of post-quantum algorithms in WebAssembly to maintain low latency.

Optimized Hashing for Large Datasets:
Enhance SHA-256 and SHA-3 performance for high-volume transactions (e.g., millions daily) by leveraging parallel processing or GPU acceleration in WebAssembly. Explore lightweight hash alternatives (e.g., BLAKE3) for resource-constrained environments while maintaining security.

Support for Low-Power Devices:
Optimize client-side encryption and hashing for low-powered devices (e.g., mobile browsers, IoT devices) by reducing computational complexity. Investigate partial offloading of cryptographic operations to edge servers without compromising client-side security.

Zero-Knowledge Proofs for Enhanced Privacy:
Integrate zero-knowledge proofs (e.g., zk-SNARKs) to allow transaction validation without revealing sensitive data (e.g., payer identity, amount). Application: Enable privacy-preserving fintech platforms, such as anonymous payments, while maintaining auditability.

Blockchain Integration for Decentralized Trust:
Incorporate blockchain technology to store transaction hashes (H1, H2) on a tamper-proof ledger, enhancing trust and auditability. Research focus: Develop smart contracts to automate hash verification, reducing server-side processing overhead.

Machine Learning for Threat Detection:
Implement machine learning models to detect anomalous transaction patterns (e.g., unusual hash mismatches) in real-time, complementing dual-hash authentication. Train models on simulated MITM and SSL strip attack data to improve proactive threat mitigation.

Cross-Platform Compatibility:
Extend the framework to support native mobile apps (iOS, Android) by adapting WebAssembly for mobile environments and integrating with platform-specific cryptographic APIs. Ensure seamless performance across diverse browser engines (e.g., Chromium, WebKit).

Regulatory Compliance and Standardization:
Align the framework with global cybersecurity standards (e.g., GDPR, PCI DSS) to facilitate adoption in regulated industries. Contribute to open-source standards for client-side encryption, promoting interoperability with existing fintech protocols. These enhancements will strengthen the framework's resilience, scalability, and applicability, positioning it as a leading solution for secure web transactions in fintech and beyond.

9. Conclusion

The proposed framework offers significant improvements over traditional security methods. Client-side encryption eliminates vulnerabilities in transport-layer attacks, such as SSL strip and downgrade attacks. The dual-hash mechanism ensures robust integrity verification, rejecting tampered data before processing. WebAssembly optimization reduces latency, making the framework suitable for high-volume fintech applications. Compared to TLS, which provides partial resilience to MITM attacks, this solution offers full protection, as demonstrated in experimental results [2]. A potential limitation is the reliance on client-side resources, which may challenge low-powered devices. Future optimizations could address this constraint. The multi-layered security framework, integrating frontend AES-GCM 256 encryption, RSA key exchange, and dual- hash authentication, provides a robust and efficient solution for securing transaction websites. It effectively mitigates SSL strip, downgrade, and MITM attacks while maintaining

high performance through WebAssembly. The framework's scalability suits fintech applications requiring real-time processing. Future work will explore post- quantum cryptography and hash optimization for larger datasets.

References

[1] Stefano Calzavara, Riccardo Focardi, Matus Nemec, Alvise Rabitti, Marco Squarcina, "Postcards from the Post-HTTP World: Amplification of HTTPS Vulnerabilities in the Web Ecosystem, 2019, Volume: 1, Pages: 281-298
[2] Jay Chung; Natalija Vlajic, "Survey of Remote TLS Vulnerability Scanning Tools and Snapshot of TLS Use in Banking Sector"
[3] Diana Gratiela Berbecaru; Giuseppe Petraglia, "TLS-Monitor: A Monitor for TLS Attacks" - 17 March 2023
[4] Mauro Conti; Nicola Dragoni; Viktor Lesyk, "A Survey of Man In The Middle Attacks", 2027 – 2051- 29 March 2016
[5] Hyunuk Hwang, Gyeok Jung, Kiwook Sohn, Sangseo Park, "A Study on MITM (Man in the Middle) Vulnerability in Wireless Network Using 802.1X and EAP, 10-12 January 2008
[6] Seung-Woo Han; Hyunsoo Kwon; Changhee Hahn; Dongyoug Koo; Junbeom Hur, "A survey on MITM and its countermeasures in the TLS handshake protocol -- 05-08 July 2016
[7] Labiblais Rahman, "Detecting MITM Based on Challenge Request Protocol" -- 01-05 July 2015