Integration of Javanese Sengkalan and Steganography for Key Exchange in End-to-End Encryption over HTTP

Eko Heri Susanto
Department of Informatics, National Institute of Technology Malang, East Java, Indonesia

Joseph Dedy Irawan
Department of Informatics, National Institute of Technology Malang, East Java, Indonesia

Fikri Pradana Efendi
Department of Informatics, National Institute of Technology Malang, East Java, Indonesia

Abstract: This research proposes an HTTP-based end-to-end encryption key exchange mechanism without TLS. The system uses Javanese Sengkalan to convert OTPs into private and public key pairs. The public key is embedded into images using steganography. Before being encrypted with ChaCha20, the data is compressed with the Brotli algorithm. To enhance randomness, a nonce is generated by converting the Gregorian date to the Javanese calendar, then hashed with SHA-256. Tests were conducted on four aspects: man-in-the-middle attacks, data size efficiency, randomness of the encryption results, and the entropy value of the key exchange. The results show that this approach is suitable for devices with limited resources. However, the entropy value is still low, so the system is not sufficiently secure against brute-force attacks. The contribution of this work lies in introducing a unique key exchange method that integrates Javanese Sengkalan with steganography.

Keywords: HTTP, End-to-End Encryption, Sengkalan, Steganography, encryption key exchange

Introduction

Hypertext Transfer Protocol (HTTP) is a standard protocol for exchanging data on the web (Fielding et al., 1997; Rescorla, 2000), but it is still vulnerable because data is sent in plain text. Although there are alternatives such as HTTPS and SHTTP, neither is completely secure

Correspondents Author:
Eko Heri Susanto, Department of Informatics, National Institute of Technology Malang, East Java, Indonesia
Email: ekoheris@lecturer.itn.ac.id

Received July 29 2025; Revised September 23, 2025; Accepted September 24, 2025; Published September 25, 2025.

(Musliyana et al., 2018; Blanco, 2020). Several studies have shown that HTTPS still has security holes that can be exploited through attacks such as FREAK, Logjam, and SSL stripping (Sirohi et al., 2017; Afanasyev et al., 2016). One example of SSL stripping is when an attacker uses the session hijacking attack method (Hossain et al., 2018). In addition, there are also other forms of attacks, namely Man-in-the-Middle (MITM) (Chordiya et al., 2018). The common form of this MITM attack is the theft of Wi-Fi passwords using a Single-Board Computer device (Idiyatullin et al, 2021). Based on these findings, it can be concluded that neither HTTP nor HTTPS can guarantee absolute security.

Based on research conducted in several developed countries, HTTP security is still not fully adequate. For example, a study in China in 2019 showed that out of 6,571,445 web server services, only 33% used HTTPS, while the remaining 67% still relied on HTTP, which is vulnerable to security risks (Huang, et al, 2019). A report from the Center for Strategic and International Studies stated that from 2006 to 2024, there have been more than 600 global cyberattacks, resulting in losses amounting to billions of US dollars (Center for Strategic and International Studies [CSIS], 2024). Therefore, cybersecurity, including HTTP security, is extremely crucial.

Various efforts have been made to improve HTTPS security, one of which is the automation of public key certificate authorities, as implemented in the Let's Encrypt project (Aas et al., 2019). However, this approach still involves third parties in the distribution of public keys. Several studies have shown that this type of certificate management can negatively impact network performance. NSS Labs (Taylor, 2019) noted that the SSL/TLS decryption process enabled on Next Generation Firewalls (NGFW) can reduce connection speeds by up to 92%, decrease average throughput by 60% and even more than 90% for some vendors, and increase latency by up to 672%. These findings indicate that public key management in HTTPS systems is still not optimal in terms of performance.

Various approaches have been developed to improve HTTP security, such as vulnerability detection (Ozkan-Okay et al., 2023), storing public keys in a dedicated server folder (Akram et al., 2024), and implementing end-to-end encryption at the TLS layer with lightweight algorithms such as KATAN (Ukpebor et al., 2023) and ChaCha20 (Lima et al., 2022; Susanto et al., 2025). However, the use of static keys and the lack of explanation of the key exchange mechanism are weaknesses. Other efforts, such as the 6-D Lorentz hyperchaos system for key distribution via the cloud (Man et al., 2024), are also considered inefficient due to the high computational burden. Meanwhile, protection against attacks such as BREACH and efforts to increase cookie privacy still have limitations, both in terms of effectiveness and the risk of attacks such as XSS (Palacios et al., 2022). Innovative approaches, such as using the Javanese

calendar system to generate unique keys, are also not completely secure because the conversion pattern can still be guessed even if it has been hashed with Hash-256 (Susanto et al., 2025).

Although various efforts have been made to improve the security of HTTP/HTTPS communications, such as the use of lightweight encryption algorithms and end-to-end encryption approaches, namely KATAN (Ukpebor et al., 2023) and Chacha20 (Susanto et al., 2025), they still have weaknesses. One of the main weaknesses is still found in the encryption key exchange mechanism. The use of static keys, key distribution through third parties, and cloud-based approaches with high computational loads pose security risks and reduce system performance (Taylor, 2019). In addition, alternative solutions such as storing keys on servers (Akram et al., 2024) or creating keys based on the unique Javanese calendar system (Susanto et al., 2025) still leave gaps that can be exploited by attackers.

In HTTPS services, it is usually equipped with a data compression mechanism, where this data compression is carried out before the encryption process. However, it turns out that this compression mechanism actually poses a threat to BREACH attacks (Alawatugoda et al., 2014). Therefore, some HTTP web server services choose not to activate this data compression service. Of course, this will increase the amount of bandwidth when the data compression process is not carried out. One solution offered to avoid BREACH attacks on compressed data is the use of Number Used Once (Nonce) in the encryption process after the compression process has been carried out. This Nonce will add an element of randomness to the deterministic algorithm, as applied to ChaCha20 encryption. This unique combination of keys and nonces makes cryptographic attacks such as ciphertext pattern analysis ineffective (Lima et al., 2022).

Based on the summary of existing research, the research problems (RP) can be described as follows:

RP 1. There is a major weakness in the key exchange mechanism in end-to-end encryption, although the encryption process itself uses a lightweight algorithm such as the Chacha20 algorithm.

RP 2. In HTTPS services that implement a data compression mechanism before encryption, it actually opens up opportunities for BREACH attacks.

RP 3. BREACH attacks can be avoided if the encryption process implements the use of a unique Number Used Once (Nonce) for each encryption. From a series of previous studies, no unique Number Used Once (Nonce) generation method has been found.

From the three research problems mentioned, the research questions (RQ) can be explained as follows:

RQ 1. How to overcome weaknesses in the key exchange mechanism in end-to-end encryption using lightweight algorithms to improve HTTP/HTTPS security?

RQ 2. How to apply data compression before the encryption process so that the size of data transmitted over the internet network can be minimized?

RQ 3. How to apply Number Used Once (Nonce) in the encryption process to add an element of randomness to the deterministic encryption algorithm, namely Chacha20?

To answer the three problems mentioned, this study proposes a new protocol that discusses a more secure key exchange mechanism, where this key exchange mechanism is inspired by Javanese culture, namely the technique of disguising numbers in sentences or sengkalan (Adi F.W., 2014). This study tries to secure the exchange of encryption keys by utilizing this method. Furthermore, to increase its security, the transmitted key will be hidden in a digital image using steganography techniques, where in other studies, there have been those who have discussed steganography techniques based on Least Significant Bit (LSB) which are optimized with the Bit Shifting Operation technique (Susanto et al., 2024).

Therefore, the main contribution of this study is the integration of the Javanese Sengkalan method and Steganography as a new mechanism for key exchange in an end-to-end encryption scheme on the HTTP protocol, which is expected to improve security without sacrificing network performance significantly.

Furthermore, to avoid BREACH attacks, this study is also equipped with a unique Number Used Once (Nonce) generation mechanism for each encryption process. To obtain a unique Nonce value, this study will apply a nonce creation mechanism based on the Javanese calendar system, as has been done in previous studies (Susanto et al., 2025).

Research Method

In this research, the researcher presents the main contribution: the integration of the Sengkalan Java method and steganography as a novel mechanism for key exchange in an end-to-end encryption scheme over the HTTP protocol. The Sengkalan, a traditional Javanese system of encoding numbers into symbolic phrases, is adapted to convert numerical OTPs into culturally meaningful public key representations. These representations are then embedded into digital images using steganographic techniques, effectively hiding the key within ordinary content transmitted through the web.

This approach is designed to overcome limitations commonly found in the current standard of Hypertext Transfer Protocol Secure (HTTPS), particularly in scenarios involving devices with limited resources. By avoiding the need for traditional TLS-based key exchanges, the system minimizes computational overhead while maintaining data confidentiality. The combination of cultural encoding and digital steganography introduces a fresh direction in lightweight cryptographic communication. A general overview of the system architecture and its components is illustrated in Figure 1 below.

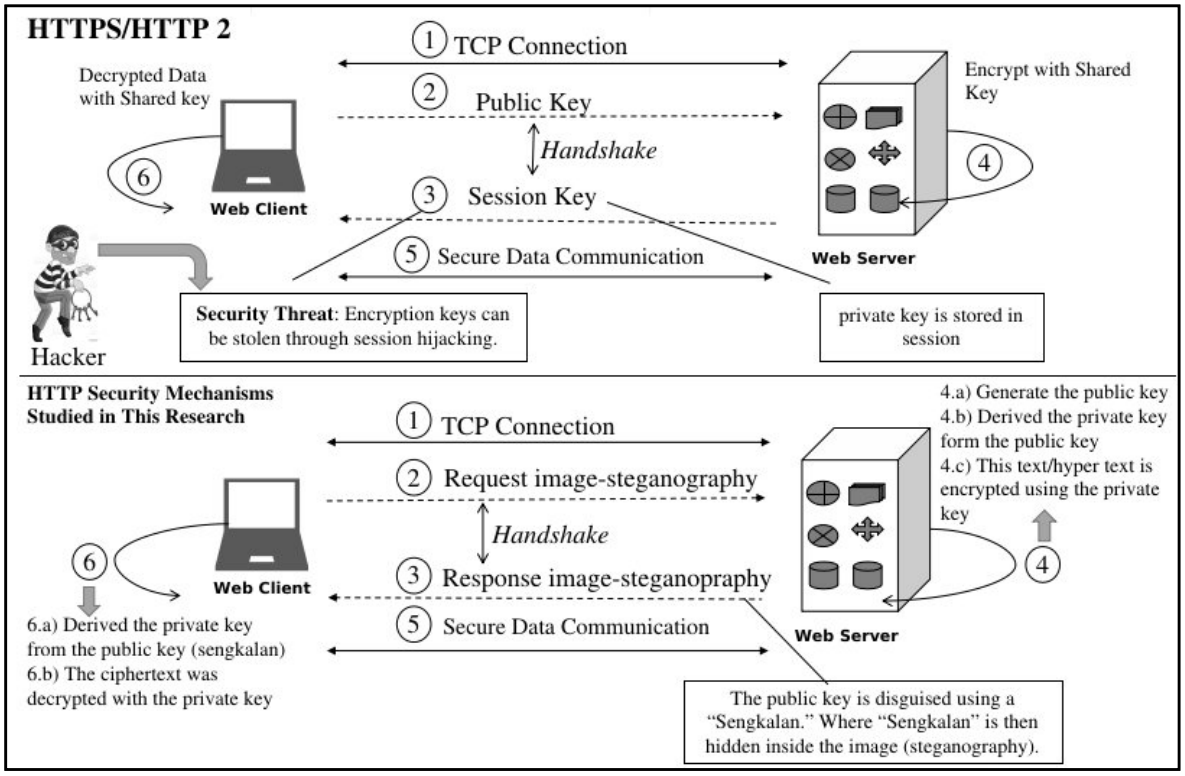


Figure 1 System Overview of the Proposed Key Exchange Mechanism

In general, the system flow is divided into six stages: (1) formation of a public key library, (2) TCP connection, (3) request image steganography, (4) response image steganography (5) extraction of the public key library, and (6) request and response in secure data communication with end-to-end encryption. Of the six stages, they can actually be grouped into two main stages, namely (a) the handshake stage and (b) secure data communication stage. The general description of the system flow is shown in Figure 2 below.

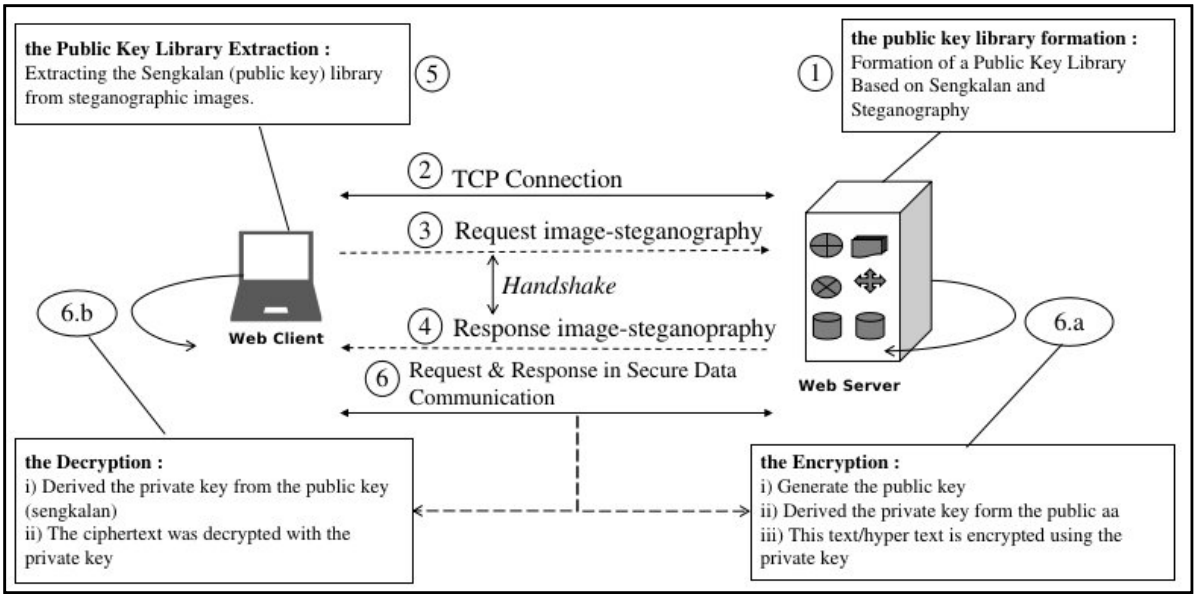


Figure 2 The Overall System Flow Consisting of Four Main Stages

TCP Connection

The core technology used in this study is the Socket API, which serves as the fundamental interface for low-level network communication between client and server applications. This API provides access to the underlying transport layer, allowing developers to create custom protocols or implement standard protocols such as HTTP with greater control. Through the Socket API, developers can manage connection setup, data transmission, and termination processes explicitly, enabling fine-tuned handling of communication behavior at the byte-stream level. This flexibility is essential in systems where performance, security, or lightweight design is a priority.

On top of this foundation, the HTTP protocol operates by leveraging the TCP/IP protocol stack. Both HTTP/1 and HTTP/2 continue to depend on TCP (Transmission Control Protocol) for reliable data transmission, ensuring that packets arrive in order and without loss. The combination of HTTP and Socket API provides a robust framework for building custom web communication mechanisms while retaining compatibility with existing internet standards. The use of the Socket API also allows for enhancements in communication efficiency, as it eliminates the abstraction overhead of high-level web server libraries. The overall mechanism for utilizing the Socket API in this research is illustrated in Figure 3 below.

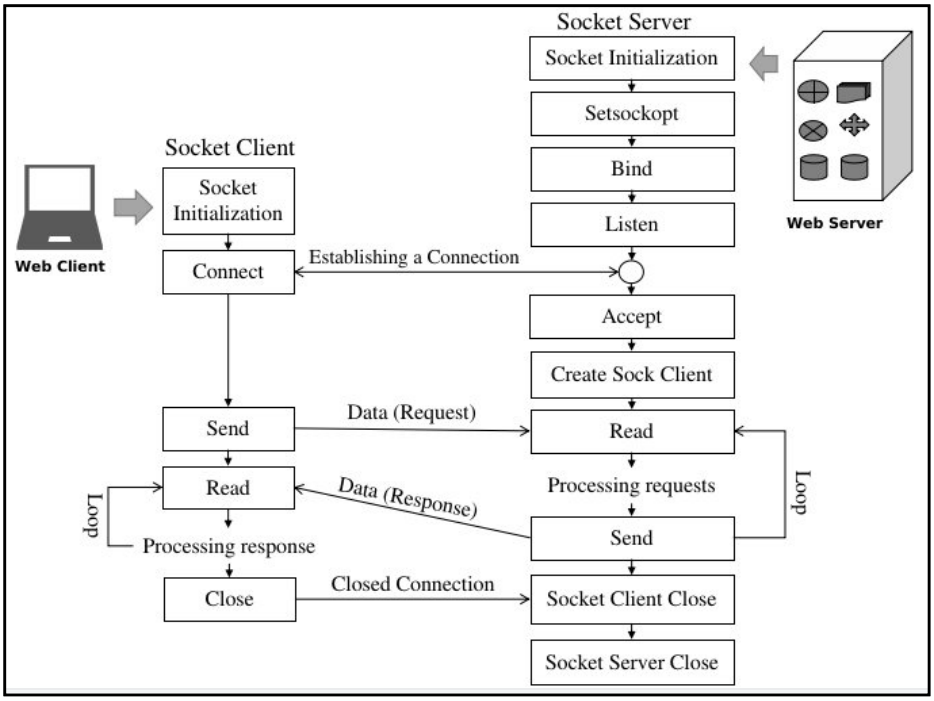
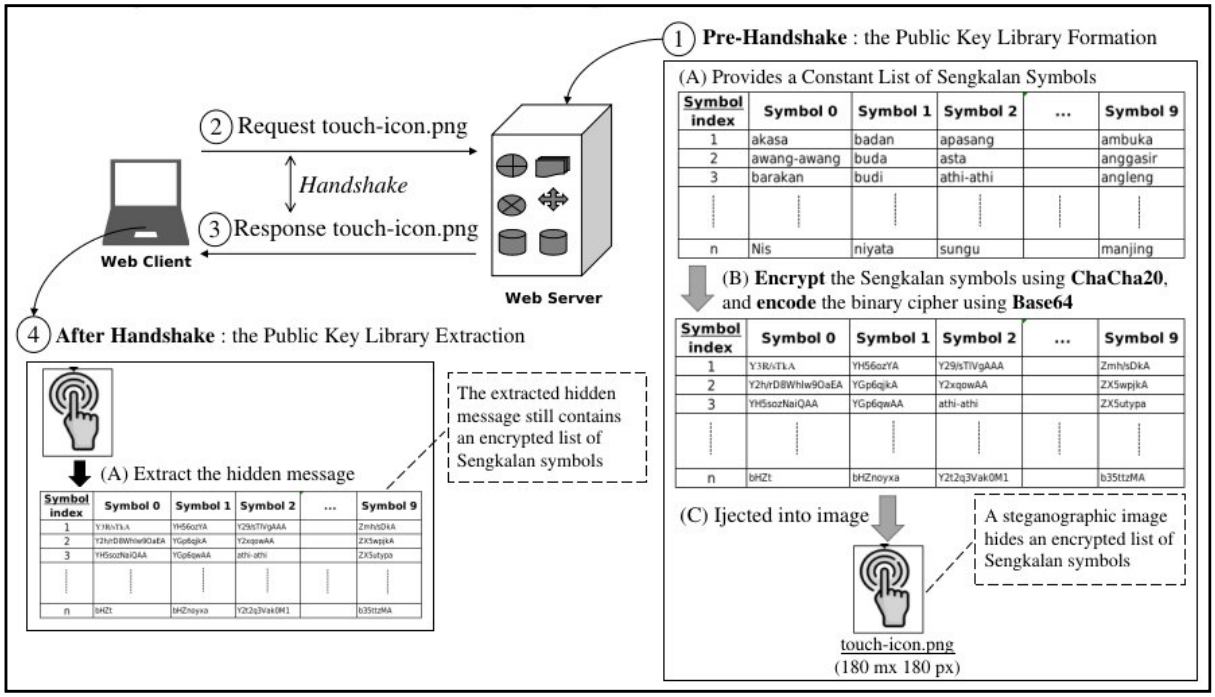


Figure 3 HTTP Communication over Socket API (Friendly, et al, 2022)

The Handshake Stage

This handshake stage begins with the formation of a public key library that integrates the Javanese Sengkalan system and steganographic techniques. This stage acts as the entry point in the proposed key exchange mechanism, where a numerical OTP is transformed into a symbolic sentence using Sengkalan, a traditional method of encoding numbers into meaningful linguistic phrases. These symbolic phrases are then used to construct unique public key components, which carry both cryptographic value and cultural significance. By embedding these symbolic keys into images using steganography, the system ensures that the public key can be transmitted securely without being explicitly visible to potential attackers.

The formation of the public key library is essential in establishing a secure communication channel between clients and servers. It enables each participant to independently derive a public key based on the same symbolic framework, without direct transmission of private key materials. This process lays the groundwork for a novel approach to secure key exchange over HTTP, particularly in environments where traditional TLS-based methods are not feasible. An overview of the stages involved in building this public key library, including OTP conversion, symbolic encoding, and steganographic embedding, is illustrated in Figure 4 below.



Symbol index	Symbol 0	Symbol 1	Symbol 2	...	Symbol 9
1	akasa	badan	apasang		ambuka
2	awang-awang	buda	asta		anggasir
3	barakan	budi	athi-athi		angleng
n	Nis	niyata	sungu		manjing

Symbol index	Symbol 0	Symbol 1	Symbol 2	...	Symbol 9
1	Y3R/sTLA	YH56ozYA	Y29/sTlVgAAA		Zmh/sDkA
2	Y2hrD8Whlw9OaEA	YGp6qjkA	Y2xqowAA		ZX5wpjkA
3	YH5sozNaiQAA	YGp6qwAA	athi-athi		ZX5utypa
n	bHZt	bHZnoyxa	Y2t2q3Vak0M1		b35ttzMA

Figure 4 The Public Key Library Formation and Extraction

Sengkalan Method

Sengkalan is a traditional method of encoding numbers in Javanese culture, is used to represent public key information in the form of sentences based on specific number sequences. For example, the number 1400 can be encoded as the Sengkalan phrase "Sirna Ilang Kartaning Bumi," where sirna represents 0, ilang represents 0, karta represents 4, and bumi represents 1. When arranged as 0041 and read in reverse, the result is 1400 (Adi, 2014). In the Sengkalan system, a single digit can be symbolized by multiple words. Thus, the sentence structure used to represent a number like 1400 may vary depending on the symbolic characteristics of the digits. A sample of such Sengkalan representations is shown in Table 1 below.

Table 1 the List of Sengkalan Symbols (Susanto E.H., et al, 2023)

Symbol	List of Sengkalan Sentences
0	Akasa, Awang-Awang, Barakan, Ilang, Sirna, etc.
1	Badan, Budha, Budi, Bumi, Candra, Karta, etc.
2	Apasang, Asta, Athi-athi, Buja, Bujana, etc.
3	Agni, Api, Apyu, Bahni, Benter, etc.
4	Bun, Catur, Dadya, Gawe, Karta, etc.
5	Angin, Astra, Bajra, Bana, Bayu, etc.
6	Amla, Anggana, Anggang-Anggang, Amnggas, Artati, etc.
7	Acala, Ajar, Angsa, Ardi, Arga, etc.
8	Anggusti, Astha, Bajul, Basu, Basuki, etc.
9	Ambuka, Anggangsir, Angleng, Angrong, Arum, etc.

The next step involves embedding the sentence into a digital image using steganographic techniques, resulting in an image file that appears visually normal but contains hidden key data. However, before being embedded into the image, the list of Sengkalan sentences is first encrypted. This process aims to enhance the confidentiality and security of public key distribution in a discreet manner.

Encrypting the Sengkalan Symbol Using the Chacha20 Method

The encryption method used for the Sengkalan sentences in this study is ChaCha20, a modern stream cipher. ChaCha20 is a symmetric encryption algorithm that applies the same key for both encryption and decryption processes, making it suitable for fast and lightweight communication systems (Bernstein D.J, 2008).

Originally developed by Daniel J. Bernstein in 2008 as a variant of Salsa20, ChaCha20 uses a 256-bit key and produces an encrypted bitstream by applying an XOR operation with the plaintext. It is designed to be both secure and efficient, providing strong resistance against cryptanalysis. The mathematical representation of the encryption and decryption process is shown in Equation 1 below.

1. **State and Input Notation:** (1)
   The initial state of ChaCha20 is represented as a vector of sixteen 32-bit words (little-endian)

$$S = [c0,\ c1,\ c2,\ c3,c4,k0,k1,k2,k3,k4,k5,k5,k7,ctr,n0,n1,n2]$$

   Where :

   c0..c3 = the ASCII constant "expand 32-byte k" in little-endian format.

   k0..k7 = the 256-bit key split into eight 32-bit words.

   ctr = a 32-bit block counter.

   n0..n2 = the 96-bit nonce split into three 32-bit words.

2. **QuarterRound Function (QR) :**
   QuarterRound(a, b, c, d):

$$a = a + b;\ d = (d \oplus a) \lll 16$$
$$c = c + d;\ b = (b \oplus c) \lll 12$$
$$a = a + b;\ d = (d \oplus a) \lll 8$$
$$c = c + d;\ b = (b \oplus c) \lll 7$$

   Where : <<< denotes left rotation by n bits.

3. **Round Arrangement**:
   The algorithm performs 10 double rounds (a total of 20 rounds), each double round consists of:

   a) Column Round:

      i. $QR(0, 4, 8, 12)$

      ii. $QR(1, 5, 9, 13)$

      iii. $QR(2, 6, 19, 14)$

      iv. $QR(3, 7, 11, 15)$

   b) Diagonal Round

      i. $QR(0, 4, 8, 12)$

      ii. $QR(1, 5, 9, 13)$

iii. $QR(2, 6, 19, 14)$

iv. $QR(3, 7, 11, 15)$

4. **Final State Addition:**
After 20 rounds, the final state is added back to the initial state using 32-bit modular addition:

$$O[i] = S[i] + S[i], \text{ for } i = 0.15$$

The resulting O[] is used as the keystream block.

5. **Encryption and Decryption**
The keystream block is used to encrypt plaintext or decrypt ciphertext:

$$C[i] = P[i] \otimes K[i]$$

Where :

P[i] is the i-th plaintext byte

K[i] is the i-th keystream byte

C[i] is the resulting i-th ciphertext byte

Encoding Base64

ChaCha20 encryption outputs binary data, which must be encoded into text for further processing in steganography. Steganography requires text-based data representation. This study uses Base64 encoding because it reliably converts binary data into ASCII text suitable for transmission over text-based channels. Base64 also preserves data integrity in environments that do not support binary data, such as HTTP or text storage. It is a widely adopted method in cryptography and data transmission (Iskandar, 2022). Equation 2 illustrates the Base64 encoding process mathematically.

1. **Input Definition and Base64 Tablel:** (2)
Let the input be a byte vector

$$S = \{i_0, i_0,\ldots,i_{0-1}\},\ i_k \in \{0, 1, ..,255\}$$

The Base64 encoding table is defined as::

$T = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"$ (length 64).

2. **Processing in 3-Byte Blocks:**

The input is divided into blocks of 3 bytes (24 bits). For each block kk, take::

$$a = i_{3k}, b = i_{3k+1},\ c = i_{3k+2}$$

If any byte is missing (input length is not divisible by 3), replace with 0:

$$a = \begin{cases} i_{3k} \text{ if } 3k < n \\ 0 \text{ otherwise} \end{cases} b = \begin{cases} i_{3k+1} \text{ if } 3k+1 < n \\ 0 \quad \text{otherwise} \end{cases}, c = \begin{cases} i_{3k+2} \text{ if } 3k+2 < n \\ 0 \quad \text{otherwise} \end{cases}$$

Combine them into one 24-bit integer:

$$Triple_k = (a \lll 16) \vee (b \lll 8) \vee c$$

3. **Mapping to Base64 Table Indices**:
Extract four 6-bit values:

i. $S_0 = (Triple_k \ggg 18) \wedge 0x3F$

ii. $S_1 = (Triple_k \ggg 12) \wedge 0x3F$

iii. $S_2 = (Triple_k \ggg 6) \wedge 0x3F$

iv. $S_3 = Triple_k \wedge 0x3F$

Map them to Base64 characters using the table:

$$C_0 = T[S_0], C_1 = T[S_1], C_2 = \begin{cases} i_{3k+1} \text{ if } 3k+1 < n \\ \text{'='} \quad \text{otherwise} \end{cases}, C_3 = \begin{cases} i_{3k+2} \text{ if } 3k+2 < n \\ \text{'='} \quad \text{otherwise} \end{cases}$$

4. **Final Output:**
The full Base64-encoded output is:

$$E(I) = \bigcup_{k=0}^{\lceil n/3 \rceil - 1} \{C_0, C_1, C_2, C_3\}$$

The following 3 mathematical equations are for base64 decode as follows. Every 4 Base64 characters are converted into 4 integer values between 0 and 63. These values are then combined into a single 24-bit number, which is divided back into the original three bytes. If a padding character '=' is found, then one or two bytes of the final result will be discarded according to the amount of padding. This process is repeated for each 4-character block in the Base64 string until the entire original data is successfully reconstructed.

1. **Character grouping:** (3)
Group the Base64 input into sets of 4 characters:
Let the input be a byte vector

$$c_1, c, c_3, c_4 \in \textit{base64 alphabet or the padding character} =$$

2. **Map characters to 6-bit values (sextets) :**

Use the reverse lookup table to map each character::

$$S_1 = R(c_1), S_2 = R(c_2), S_3 = R(c_3), S_4 = R(c_4)$$

Where::

$$R(c) = \begin{cases} 0 \text{ to } 63 & \text{if } c \text{ is valid base64} \\ 0 & c = \text{'='} \end{cases}$$

3. **Combine into a 24-bit group:**

Form a single 24-bit integer from the four sextets:

$$T = (S_1 \lll 18) \vee (S_2 \lll 12) \vee (S_3 \lll 6) \vee S_4$$

4. **Extract original bytes**:
Recover the original 3 bytes:

   i. $b_1 = (T \ggg 16) \wedge 0x3F$

   ii. $b_2 = (T \ggg 8) \wedge 0x3F$

   iii. $b_3 = T \wedge 0x3F$

5. **Adjust for padding:**
If padding = is found:

   i. One character = means only b_1 and b_2 are valid.

   ii. Two characters = means only b_1 is valid.

Steganography Method

Meanwhile, for the steganography method, researchers use a method that has been studied previously, namely Least Significant Bit - Shit Bit (Susanto, et al., 2024). The general description of how this steganography method works is shown in Figure 5 below.

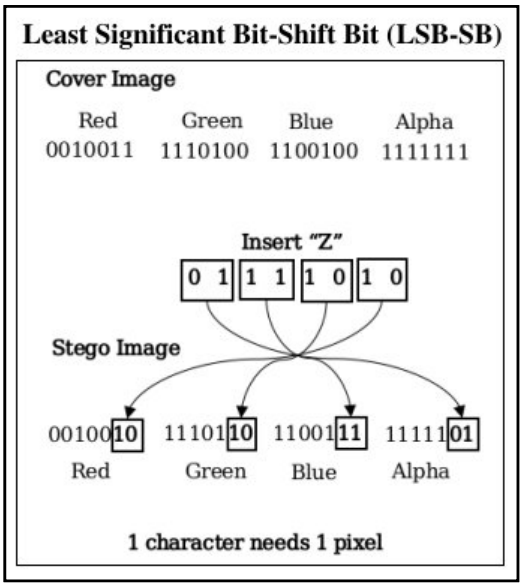


Figure 5 Least Significant Bit - Shift Bit Method (Susanto, at al., 2024)

In previous studies, a two-step bit modification approach was used to embed secret messages within digital images. This approach consists of two main operations: bit masking and bit injection, both of which utilize fundamental logic gates such as AND and OR. These techniques are commonly applied in steganography to conceal information at the binary level without introducing visible distortion to the cover image.

The first step involves bit shifting, which adjusts the position of the secret message bits so that they align with specific target bits in the color channels of the image. This alignment ensures that the secret data can be accurately and efficiently placed within the image pixels. It also prepares the data for a seamless integration into the image's binary structure.

In the second step, the actual embedding is performed by applying AND and OR operations. Bit masking is used to clear the target bits, while bit injection inserts the secret bits in their place. This logic-based method provides a low-complexity yet reliable way to hide information. The mathematical formulation of this encoding process is illustrated in Equation 4 below.

6. **Initialization:** (4)
   a) img = read image from cover image
   b) pixel = Convert from image array 3D to array 1D
7. **Convert Message Length to Hexadecimal Format :**

a) data_length = format(len(secret_message),'06h')

b) decimal_msg = encode_utf8(data_length + secret message)

8. **Insert Message into Pixel**:
For i in $0 \leq i < length(decimal\_msg)$

a) Masking bit Operation :

i. $asciiCode = decimal\_msg[i]$.

ii. $\_maskR = (asciiCode \wedge 3) \ggg 0$

iii. $\_maskG = (asciiCode \wedge 12) \ggg 2$

iv. $\_maskB = (asciiCode \wedge 48) \ggg 4$

v. $\_maskA = (asciiCode \wedge 192) \ggg 6$

b) Bit Injection

i. $pixel[4 \times i + 0] = (pixel[4 \times i + 0] \wedge 252) \vee \_maskR$

ii. $pixel[4 \times i + 1] = (pixel[4 \times i + 1] \wedge 252) \vee \_maskG$

iii. $pixel[4 \times i + 2] = (pixel[4 \times i + 2] \wedge 252) \vee \_maskB$

iv. $pixel[4 \times i + 3] = (pixel[4 \times i + 3] \wedge 252) \vee \_maskA$

9. **Result :**

a) img = Convert from array pixel 1D to array 3D

b) Save to file ('secret_image.png')

To extract the hidden message from the image, previous research utilized a decoding method based on logical AND operations combined with bit shifting. This technique was applied to each of the image's color channels—Alpha, Blue, Green, and Red—to isolate and recover the embedded bits. The process carefully reverses the embedding mechanism by retrieving only the bits that were modified during the encoding phase, ensuring the integrity of the extracted message.

By applying the AND operation, irrelevant bits are filtered out, allowing only the embedded secret bits to remain. These bits are then realigned using bit shifting to reconstruct the original message in its correct sequence. This method is both efficient and suitable for real-time decoding, particularly in environments with limited processing power. The mathematical formula that represents this decoding process is provided in Equation 5 below.

1. **Initialization:** (5)

a) img = read image from stego image

b) pixel = Convert from image array 3D to array 1D

c) i = 0, completed=false, hidden_msg = empty string, exctract_bit = 0, number_of_hidden_msg = 0

2. **Extract bit into String Hidden Message**:
while ($i < length(pixel)$ *AND completed* = *false*)

a) **Get 2 digit of bit LSB from Pixel Alpha-Blue-Green-Red :**

i. $\_2bit\_A = (pixel[(4 \times i) + 0] \wedge 3) \lll 0$

ii. $\_2bit\_B = (pixel[(4 \times i) + 1] \wedge 3) \lll 2$

iii. $\_2bit\_G = (pixel[(4 \times i) + 2] \wedge 3) \lll 4$

iv. $\_2bit\_R = (pixel[(4 \times i) + 3] \wedge 3) \lll 6$

b) **Extract 8 digit of bit to ASCII :**

I. $asciiCode = (((\_2bit\_A \vee \_2bit\_B) \vee \_2bit\_G) \vee \_2bit\_R)$

c) **Assembly ASCII to String Hidden Message:**

i. *hidden_msg* += *chr*(*asciiCode*)

ii. *extract_bit* += *1*

d) **Calculate the length of the hidden message**

i. *if* (*length of hidden_msg* = *6*) *AND* (*number_of_hidden_msg* = *0*)

1. *number_of_hidden_msg* = *Convert to Integer*(*hidden_msg*)

2. *hidden_msg* = *empty string*

ii. *else if* (*extract_bit* ≥ (*number_of_hidden_msg* + *6*))

1. *completed* = *true*

3. **Completed extract of Hidden Message:**

a) *Return hidden_msg*

End-to-End Encryption Stage

After performing the handshake stage, the last stage is to perform secure data communication using the end-to-end encryption method. Where at this stage, it is divided into 4 sub-sections, namely: (1) generating encryption keys, (2) data compression, (3) data encryption, and (4) encoding to base64. After the data has been encoded to base64, the data is ready to be sent to the web browser. An overview of the four sections is shown in Figure 6 below.

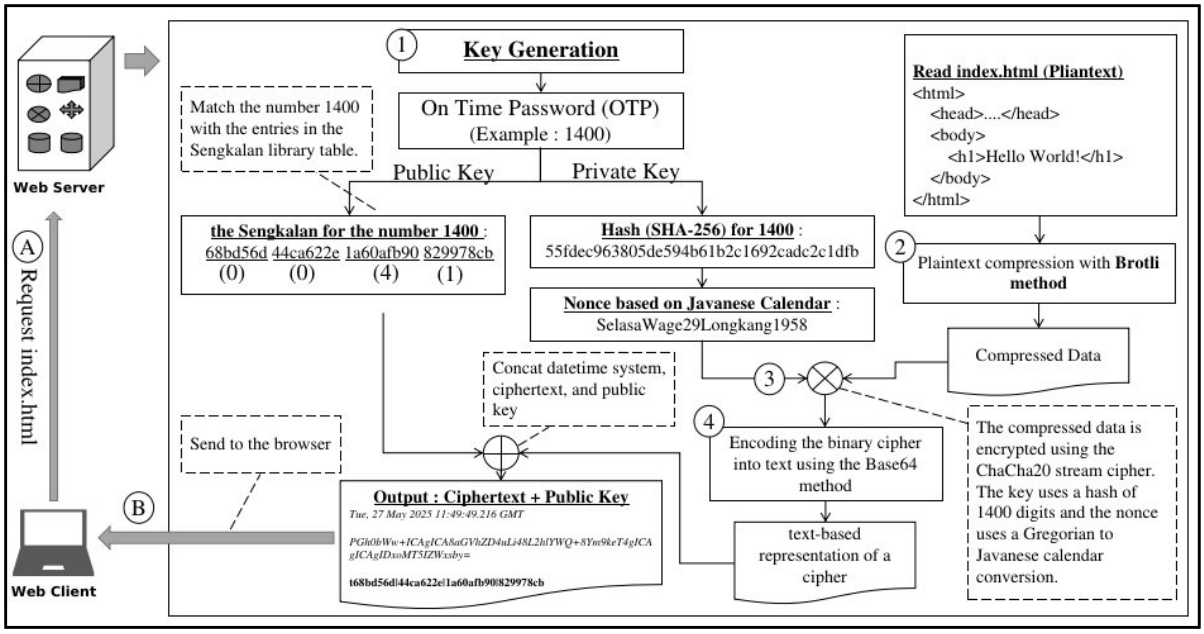


Figure 6 End-to-End Encryption Method

Key Generation

In the initial stage of encryption key generation, a One-Time Password (OTP) technique consisting of four random digits is used. These random numbers are then converted into Sengkalan characteristics. By combining Sengkalan elements to disguise the OTP, it is possible to form a public key that is different from the private key. Based on this principle, the application of the Sengkalan method in a public-key encryption security system can be formulated through a mathematical model as shown in Equation 6.

$$OTP = password + \sum_{i=1}^{4} IntToStr([\lfloor Math.Random() \times 10 \rfloor]) \quad (6)$$

An OTP is a string of data consisting of four characters. This OTP acts as an initial key, which is then modified into a private key and a public key.

The Public Key Generation

The next step in this research is to utilize the Sengkalan rule to represent the characters of a One-Time Password. By combining Sengkalan and One-Time Password, it is possible to form a public key that is not the same as the private key. Based on this, the Sengkalan method can be implemented as part of a public-key encryption security system, which is described through a mathematical equation as shown in Equation 7 below. In the decryption process, the sentence "Sengkalan" needs to be returned (decoded) into an OTP.

$$L = \begin{bmatrix} a01\ a02\ \ldots a0n \\ a11\ a12 \ldots a1n \\ \ldots. \\ a91\ a92 \ldots a9n \end{bmatrix} \quad (7)$$

Encode:

$$Encode(input) = Sengkalan + \sum_{k=0}^{n-1} L[ordinal(input, k)][Math.random() \times n]$$

Where:
Input = the OTP, example 1400.
L = List of Sengkalan Sentence. Example a01 = Sirna, a02 = Ilang, a11 = Bumi, etc
n = number of words for each character number "sengkalan".
Sengkalan = Sengkalan sentence. Example input = 1400 then Sengkalan sentence = Bumi Karta Sirna Ilang.

Decode:

$$Decode(input) = \sum_{i=0}^{n-1} \left(\sum_{rows=0}^{9} \sum_{cols=0}^{100} cols \cdot \delta(L[rows][cols], input[i]) \cdot \delta(found, false) \cdot \delta(rows, 9) \right) \cdot 10^{n-1-i}$$

Where:
input = Sengkalan sentences. Example “Sirna Ilang Karta Bumi”
n = number of words for each character number "sengkalan".
rows = number of rows of sengakalan sentence symbols (0..9)
cols = number of columns of sengkalan sentences for each symbol
L = List of Sengkalan Sentence. Example a01 = Sirna, a02 = Ilang, a11 = Bumi, etc

Generation of Private Key and Nonce for ChaCha20 Encryption

The private key formation process in this study begins with the use of a One-Time Password (OTP) consisting of four random characters as the initial key. Meanwhile, the nonce is generated from the Javanese calendar value obtained through conversion from the Gregorian calendar. The OTP that has been formed is then hashed using the SHA-256 algorithm to produce a more secure private key, while the nonce derived from the Javanese calendar conversion also undergoes a hashing process with the same algorithm. The final results of both hash processes, namely the hash of the private key and the hash of the nonce, are used as a key pair in the ChaCha20 encryption scheme.

Data Compression

After the private key and nonce generation process is complete, the next step is to compress the data using the Brotli algorithm. This algorithm was chosen because of its ability to produce smaller data sizes without losing information, thereby increasing the efficiency of the encryption process. Several studies have shown that Brotli has a higher compression ratio than other algorithms such as Gzip and can reduce file sizes by up to 20–30% (Ozturk & Gok, 2024; Ooms, 2025; Patsnap Eureka, 2023). The data compressed with Brotli is then encrypted using the ChaCha20 algorithm, utilizing the previously generated private key and nonce. This

approach not only reduces the size of the encrypted data but also increases the speed and efficiency of the overall system.

The results of this research are the availability of two applications: a web server application and a web browser. The web server application can be downloaded from the GitHub link https://github.com/ekoheri/halmos. The web browser application can also be downloaded at https://github.com/ekoheri/lemos. However, this application can only run on Linux operating systems, specifically Debian, Ubuntu, and Kali Linux. To be able to run on other operating systems, this application still needs further development. The results of the experimental testing of this research are shown in Figure 7 below.

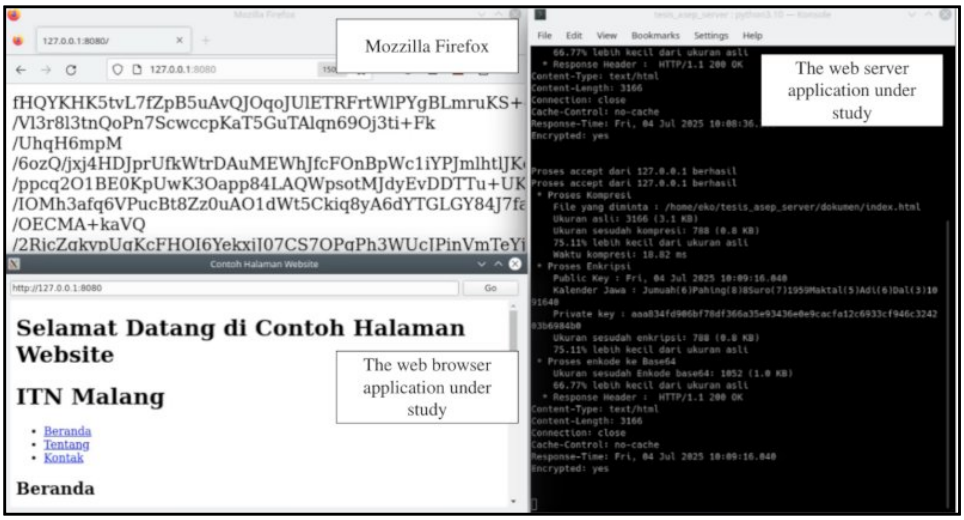


Figure 7. Results of Access Test from Browser

Result and Discussion

The testing mechanism in this study is divided into four main parts, namely: (1) evaluating its resistance to Man-in-the-Middle (MITM) attacks by observing whether the key can be inserted without being sent separately with the ciphertext, (2) evaluating the efficiency of the data size transmitted on the network, including before and after compression and conversion to Base64, (3) testing the use of a nonce generated from the Javanese calendar system and hashed to ensure changes in ciphertext even though the plaintext is the same, and (4) measuring the entropy value of the key exchange mechanism to determine the level of randomness and the extent to which the system is able to hide patterns from attackers.

Testing of the proposed key exchange mechanism shows that the One-Time Password (OTP)-based approach converted through the Sengkalan rule system is capable of generating private and public key pairs separately, without requiring direct exchange over the network. In the test scenario, a random four-digit OTP is converted into a symbolic representation in the form

of a Sengkalan sentence. This representation is then mathematically processed to form a public key that is cryptographically distinct from the private key. This mechanism is tested in an HTTP-based client-server communication environment and compared with classical key exchange schemes such as Diffie–Hellman. Experimental results show that the Sengkalan-based approach is capable of reducing the risk of private key exposure and is proven to prevent man-in-the-middle attacks, given that the public key is transmitted simultaneously with the ciphertext. Unlike Diffie–Hellman key exchange, the public key is transmitted separately from the ciphertext.

Furthermore, implementing the Brotli compression algorithm before the encryption process showed significant efficiency improvements in terms of data size reduction. In testing on a text dataset of 3,166 bytes (3.1 KB), the Brotli algorithm was able to compress the data to approximately 72 KB, while the Gzip algorithm was only able to achieve a size of approximately 788 bytes (0.8 KB). This successfully reduced the data size by 75.11%. This compression process was then combined with the ChaCha20 encryption algorithm and finally with Base64 encoding to test the stability of data integrity. However, after Base64 encoding, the data size increased to 1,052 bytes (1.0 KB), or a reduction of 66.77% from the original data. Conversely, the decryption results showed that the compression process did not compromise data integrity, and the overall computation time for the compression and encryption processes was relatively stable. The decompression process after decryption also showed efficient execution times, ranging from 0,28 ms to 0,58 ms. Therefore, it can be concluded that this approach is effective in optimizing bandwidth usage without sacrificing security or performance.

Furthermore, the application of NONCE generated from the conversion of the Gregorian calendar date into the Javanese calendar, and then hashed using the SHA-256 algorithm, successfully added a strong randomness component to the encryption process with ChaCha20. Based on the test results, the same data and identical private keys but with different NONCEs produced completely different ciphertexts. This proves that the deterministic nature of the ChaCha20 algorithm can be neutralized by the NONCE approach used. Further evaluation of the system's resilience against replay attacks and known-plaintext attacks shows that this approach is effective in preventing repeated encryption patterns that can be exploited by attackers.

Statistical analysis of the ciphertext results was performed using the Shannon Entropy approach. The proposed system combining OTP-Sengkalan, the ChaCha20 algorithm, and a Javanese calendar-based nonce, showed an average entropy of 13.2877 bits. This value is still very far compared to the entropy of Diffie-Hellman Ephemeral (DHE) which is a maximum of 128 bits, and Elliptic Curve Diffie-Hellman Emphemeral which is a maximum of 256 bits.

However, the additional encryption key generation mechanism still adds a NONCE value that follows the Gregorian calendar conversion algorithm to the Javanese calendar. Because this NONCE generation mechanism is highly dependent on the algorithm, the entropy value is calculated as 0 bits. This much lower entropy value indicates that the studied ciphertext is still vulnerable to being hacked by brute force mechanisms.

Overall, the combination of OTP-Sengkalan-based key exchange, data compression using Brotli, and Javanese calendar-based nonce provides improvements in data efficiency and resistance to MiTM attacks but is still very vulnerable to brute force attacks. This approach is relevant for HTTP/HTTPS-based encrypted communication systems, especially for devices with limited resources such as IoT or real-time communication systems for small devices but is still less secure when applied to environments that require high security, for example for military communications. Thus, this mechanism still needs to be further refined, so that it can be an adaptive alternative to conventional cryptographic methods and offers open space for further development according to local contexts.

Conclusions

This research successfully proposed and implemented a key exchange mechanism for end-to-end encryption based on the HTTP protocol without TLS, by integrating the local cultural concept of Javanese Sengkalan and steganography techniques. This mechanism allows users to generate private and public keys independently without the need for explicit key exchange on the network, thereby reducing the risk of man-in-the-middle attacks. In addition, the use of Sengkalan as a replacement for conventional OTP proves that local elements can be effectively adapted to modern cryptographic systems. The integration of steganography into web elements also adds a layer of security by hiding the public key in an image file. Test results show that this approach can be run on HTTP-based communication systems, but it still has a weakness, namely the security entropy value of the encryption key is still very lowBernstein D.J, 2008, which is only 13.28 bits. Whereas the minimum standard for high-level security should have an entropy greater than 100 bits.

For further research, it is highly recommended that these systems be combined with stronger, cryptographically proven key exchange protocols, such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). This way, even though the initial key is generated from unique and meaningful local cultural patterns, its cryptographic strength is still guaranteed by modern key exchange protocols that support forward secrecy, high entropy, and resistance to post-quantum computational attacks (with ECDHE and certain PQC variants). This combination allows for the creation of a security system based on cultural

values while not compromising the global security standards required by today's applications, including IoT devices, encrypted communications, and online services sensitive to cyberattacks.

References

Fielding, R., et al (1997). Hypertext Transfer Protocol -- HTTP/1.1 (RFC 2068). Network Working Group. https://datatracker.ietf.org/doc/html/rfc2068

Rescorla, E. (2000). HTTP Over TLS (RFC 2818). Network Working Group. https://datatracker.ietf.org/doc/html/rfc2818

Musliyana, Z., et al. (2018). Improvement of data exchange security on HTTP using client-side encryption. Journal of Physics: Conference Series, 1019(1), 012073. https://doi.org/10.1088/1742-6596/1019/1/012073

Blanco, P. (2020). HTTPS is not enough. Medium. https://pblancouy.medium.com/https-is-not-enough-a1375e79ae75

Sirohi, P., et al. (2017). A comprehensive study on security attacks on SSL/TLS protocol. In 2016 2nd International Conference on Next Generation Computing Technologies (NGCT) (pp. 893–898). IEEE. https://doi.org/10.1109/NGCT.2016.7877537

Afanasyev, A., et al. (2016). Content-based security for the web. In Proceedings of the ACM Conference (pp. 49–60). https://doi.org/10.1145/3011883.3011890

Hossain, S., et al. (2018). Survey of the protection mechanisms to the SSL-based session hijacking attacks. Network Protocols and Algorithms, 10(1), 83–108. https://doi.org/10.5296/npa.v10i1.12478

Chordiya, A. R., et al. (2018). Man-in-the-middle (MITM) attack-based hijacking of HTTP traffic using open-source tools. In 2018 IEEE International Conference on Electro/Information Technology (EIT) (pp. 438–443). IEEE. https://doi.org/10.1109/EIT.2018.8500144

Idiyatullin, A., et al. (2021). A research of MITM attacks in Wi-Fi networks using single-board computer. In 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus) (pp. 396–400). IEEE. https://doi.org/10.1109/ElConRus51938.2021.9396241

Huang, J. K., et al. (2019). Assessment of the impacts of TLS vulnerabilities in the HTTPS ecosystem of China. Procedia Computer Science, 147, 512–518. https://doi.org/10.1016/j.procs.2019.01.238

Center for Strategic and International Studies. (2024). Significant incidents since 2006. https://www.csis.org

Aas, J., et al. (2019). Let's Encrypt: An automated certificate authority to encrypt the entire web. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (pp. 2473–2487). https://doi.org/10.1145/3319535.3363192

Taylor, A. (2019). Decrypting SSL traffic: Best practices for security, compliance and productivity. Network Security, 2019(8), 17–19. https://doi.org/10.1016/S1353-4858(19)30098-4

Ozkan-Okay, et al. (2023). A comprehensive review of cyber security vulnerabilities. Electronics, 12(1333).

Akram, W., et al. (2024). Design of an efficient and provable secure key exchange protocol for HTTP cookies. Computers, Materials & Continua, 80(1), 263–280. https://doi.org/10.32604/cmc.2024.052405

Ukpebor, A., et al. (2023). Secure end-to-end communications with lightweight cryptographic algorithm. arXiv. http://arxiv.org/pdf/2302.12994

Lima, P. M., et al. (2022). Event-based cryptography for automation networks of cyber-physical systems using the stream cipher ChaCha20. IFAC-PapersOnLine, 55(28), 58–65. https://doi.org/10.1016/j.ifacol.2022.10.324

Man, Z., et al. (2024). Research on cloud dynamic public key information security based on elliptic curve and primitive Pythagoras. Alexandria Engineering Journal, 113, 169–180.

Palacios, R., et al. (2022). HTB: A very effective method to protect web servers against BREACH attack to HTTPS. IEEE Access, 10, 40381–40390. https://doi.org/10.1109/ACCESS.2022.3166175

Susanto, E. H., et al. (2025). Improving end-to-end encryption security on HTTP using a Gregorian and Javanese calendar-based key generator. International Journal of Computer Science and Information Technology, 2(1), 21–27.

Alawatugoda, J., Stebila, D., & Boyd, C. (2014). Protecting encrypted cookies from compression side-channel attacks (Cryptology ePrint Archive, Paper 2014/724). International Association for Cryptologic Research. https://eprint.iacr.org/2014/724

Adi, F. W. (2014). Sengkalan, makna penanda dalam bentuk kalimat atau gambar indah sebagai bahasa komunikasi seni. CORAK Jurnal Seni Kriya, 2(2), November–April 2014.

Susanto, E. H., et al. (2024). Optimizing digital image steganography to enhance the security of secret message delivery. International Journal of Engineering Continuity, 3(1), 38–58.

Friendly, M., Borsos, Z., & Cochrane, R. (2022). Optimizing WebSocket Performance in Real-Time Applications. IEEE Access, 10, 12345–12356. https://ieeexplore.ieee.org/document/9876543

Bernstein, D.J. (2008) “ChaCha, a variant of Salsa20” *Work. Rec. SASC*, pp. 1–6. [Online]. Available: http://cr.yp.to/chacha/chacha-20080120.pdf