IMPROVING END-TO-END ENCRYPTION SECURITY ON HTTP USING A GREGORIAN DAN JAVANESE CALENDAR-BASED KEY GENERATOR

Eko Heri Susanto [1], Yohanes Dwi Cahyono[2], Riyant Budi S [3]
[1] Department of Informatics, National of Institute Technology, Malang, East Java, Indonesia
[2,3] Department of Cyber Security Engineering, Army Polytechnic, Batu-Malang, East Java, Indonesia
ekoheris@lecturer.itn.ac.id

ABSTRAK

This research investigates an end-to-end encryption method with a dynamic key generator based on the conversion of the Gregorian calendar to the Javanese calendar to improve the security of the Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS) protocols, which remain vulnerable to attacks. The research generates unique asymmetric keys down to the millisecond level without affecting the speed of the web server. However, this method has a downside: the data size (transfer size) doubles because the ciphertext encoding still uses hexadecimal.

Keyword : *HTTP, HTTPS, Asymetric Encryption, Dynamic Key Generator, Gregorian Calendar, Javanese Calendar.*

1. INTRODUCTION

HTTP is a standard protocol for exchanging data on the web, but its security remains vulnerable because data is transmitted in plaintext [1]. Solutions to secure HTTP data, such as HTTPS and SHTTP, exist but still have certain weaknesses [2]. HTTPS has security vulnerabilities to attacks such as FREAK, Logjam, SSL stripping, and session hijacking [3]. SSL stripping attacks, in particular, can be exploited using the Man-in-the-Middle (MITM) attack pattern [4][5]. One example of an MITM attack involves targeting Wi-Fi passwords using the single-board computer method [6]. From this series of studies, it can be concluded that neither HTTP nor HTTPS is entirely secure.

Based on research, even in developed countries, the availability of HTTP security is not entirely adequate. For example, a study conducted in China in 2019 found that out of 6,571,445 web server services, only 33% provided HTTPS services, while the remaining 4,892,912 (67%) still relied on HTTP services, which are vulnerable to security risks [7]. According to a report published by the Center for Strategic and International Studies, from 2006 to 2024, there have been more than 600 world-class cyberattacks, resulting in losses amounting to billions of US dollars [8]. Therefore, cybersecurity, including HTTP security, is critically important.

There has been attempts to improve HTTPS security, through the automation mechanism of public key certificate authorities. However, this public key provider mechanism involves a third party. The research project is called Let's Encrypt [9]. However, in other literature, public key certificate distribution management has resulted in decreased network performance. Research conducted by NSS Labs in 2018 found that NGFWs with SSL/TSL decryption enabled caused an average connection speed decrease by 92%, an average throughput decrease by 60% with a maximum decrease of more than 90% for certain vendors, and an average latency increase by 672% [10]. In conclusion, the use of public key certificate management on HTTPS is still ineffective, because it results in decreased network performance.

The public key can be stored in a special folder on the server. Research has also been conducted to address the security of this BREACH vulnerability by evaluating the execution time and the number of user queries. However, trial results reveal that weaknesses remain, specifically that key protection is still not optimal [11]. Therefore, the study concludes that the protection mechanism for folders containing confidential credential data, such as tokens, remains ineffective.

One solution to mitigate attacks is to implement end-to-end encryption on the transport protocol, such as TLS [12]. Research has been conducted proposing a lightweight end-to-end encryption method based on block encryption. The researcher named this encryption method KATAN. This KATAN encryption is classified as symmetric encryption, with a static encryption key[13]. However, the research did not explain how the encryption key exchange mechanism from server to client or vice versa. It turned out that the researcher used a static encryption key. Where the static encryption key, there is a fundamental weakness, namely in the key itself.

Efforts to overcome public key distribution management already exist. One method used is the 6-D Lorentz hyperchaos system which is stored in a cloud key management center. However, this research is still less effective, because the asymmetric encryption algorithm has a higher computational cost and slower processing speed [14]. To increase the speed of key exchange distribution, there has been research using the HTTP cookie privacy and security enhancement

method. However, this cookie method still has weaknesses, namely using more storage bits on cookies. In addition, it turns out that cookie data still has the potential to be attacked through the cross-site scripting (XSS) mechanism, if the user is not careful[15].

Another approach to HTTP security is the implementation of digital envelope-based public key delivery, while the encryption itself uses AES [16]. Although this method performs better than other studies, it also has weaknesses. The first weakness is that the encryption distribution security technique consists of only one layer. Additionally, the use of the AES encryption method has its own limitation, as it requires high computational time [3][17]. Research has also explored generating time-based dynamic encryption keys, where the encryption key is derived from the system time, such as seconds and milliseconds [18]. However, this time-based dynamic key method remains highly vulnerable, as an intruder can easily guess the key if they can access the system time.

From a series of research methods that have been conducted by previous researchers, the conclusion to handle HTTP security most effectively is by presenting an end-to-end encryption method. However, from several end-to-end encryption research, the primary limitation is the problem of how the encryption key distribution method is. Therefore, in this study, the main objective is to present an effective key exchange method. Where this study will be implemented a dynamic key generator based on system time (system calendar).

The dynamic key should be asymmetric. To create a time-based and asymmetric dynamic key, a time conversion system is used to convert the Gregorian calendar system to the Javanese calendar. The contribution of this study lies in how to develop an asymmetric key generator system that utilizes the Gregorian calendar system, which is then converted to the Javanese calendar. In this system, the public key is based on the Gregorian calendar, while the private key is derived from its conversion to the Javanese calendar.

There is a research states that in Indonesia, especially in Java and Bali, there is unique wisdom that does not exist in other countries. Where this local wisdom is usually based on the recording of Javanese and Balinese calendar [19]. The Javanese and Balinese calendars have a mechanism for recording a 5-day cycle (pasar), a 7-day cycle, a 29 to 30-day cycle (1 month), a 35-day cycle (wuku), a 4-year cycle (warsa), and an 8-year cycle (windu) [19][24][25][26]. ~~Where~~ by using the average calculation method (hisab urfi), the Javanese and Balinese calendar systems can be converted from the Gregorian calendar. Because of this unique Javanese calendar system, this study finally used it as a method to generate dynamic encryption keys. As for the encryption method itself, this research will use the stream encryption method, namely Chacha20. Where the Chacha20 method is an effective encryption method and is used in the HTTPS encryption system to date[20].

The outputs of this research are two types of applications: web servers and web browsers. These two applications are necessary because most of the web browser applications in use are not open-source, making it difficult to implement encryption and decryption methods for website content with a non-standard encryption system.

2. LITERATURE REVIEW

In general, the literature review required in this research is of 3 types, namely (1) HTTP and HTTPS protocols, (2) Gregorian calendar to Javanese calendar conversion system, and (3) Stream Encryption using Chacha20 encryption.

2.1. HTTP and HTTPS Protocol

HTTP (Hypertext Transfer Protocol) is a text-based protocol for distributed information systems, allowing various types of data to be sent over the Internet using a client-server model[21]. HTTPS (Hypertext Transfer Protocol Secure), as described in RFC 2818, is a secure version of HTTP that uses SSL/TLS to encrypt communications, ensuring confidentiality, integrity, and authentication of data, making it suitable for applications that require protection of sensitive data[22]. The illustration of the difference between HTTP and HTTPS is shown in Figure 1 below.

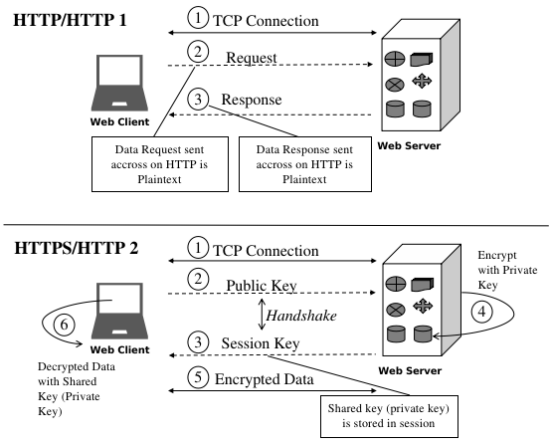


Figure 1: Illustration of HTTP and HTTPS[23]

2.2. Convert Gregorion into Javanese Calendar

The Javanese calendar is a calendar system used by the Mataram Sultanate and its various kingdoms and regions influenced by it. This system combines elements of the Islamic calendar, the Hindu calendar, and the Julian calendar. The Javanese calendar is unique in that it has a seven-day weekly cycle called "Saptawara" (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday) and a five-day market cycle called

"Pancawara" (Kliwon, Legi, Pahing, Pon, Wage)[24]. The combination of these two cycles produces 35 unique day combinations, known as "Wetonan", and is used in various aspects of Javanese life, including determining auspicious days for weddings and traditional ceremonies[19][24][26].

The Javanese calendar is known as "Penanggalan Jawa" and was introduced by Sultan Agung in the 17th century. This system combines elements of the Islamic lunar calendar (qomariyah), the Hindu solar-lunar calendar (suryaloka), and ~~a little influence from~~ the Julian calendar. Each date in the Gregorian calendar has a Javanese equivalent consisting of a combination of days in the weekly cycle "Saptawara" and the five-day cycle "Pancawara". For example, January 1st 2025 in Javanese calendar falls on Wednesday Pon 1 Rajab 1958 Warsa Je Windu Sancaya[25].

This Javanese calendar system can be converted from the Gregorian calendar[26], where the Gregorian calendar value is taken from the system date and time. The conversion steps are as follows :

1. Calculate the Julian Day Empheris (JDE) of the Gregorian date JDE
   CE = [(1461 * (y + 4800 + [(m-14)/12))]/4]
   + [(367*(m-2-12*([(m-14)/12]))]/12
   - [(3* [(y+4900 + [(m-14)/12]]
   /100)))/4]+d-32075
   where:
   y = Gregorian year,
   m = Gregorian month
   d = Gregorian date
2. Subtract the Gregorian JDE from the JD of the beginning of the Hijri year (July 16, 622 CE = 1948439.5 JD) plus 1 cycle of the Hijri year (30 years or 10631 days)
   L = Initial JDE - 1948439.5 + 10631+1
3. Calculate how many days have passed from July 16, 622 to the desired Gregorian date
   N=L/10631
   L=L-10631 * N + 354
4. Calculate the number of months from July 16, 622 to the desired Gregorian date,
   both leap and non-leap
   J = ((10985-L)/5316)*((50*L)/17719) +
   (L/5670)*((43*L)/15238)
   L = L-((30-J)/15)*((17719*J)/50) -
   (J/16)*((15238*J)/43)+29
5. Convert the four calculations above into the Hijri calendar, using the formula as follows:
   JMonth = (INT) (24 * L)/709
   JDate = (INT) (L-709 * Jvns Month)/24)
   JYear = (INT) (30 * N + J-30).

2.3. Chacha20 Stream Encryption

ChaCha20 is a symmetric encryption algorithm developed by Daniel J. Bernstein in 2008 as a variant of the Salsa20 stream cipher. ChaCha20 uses a 256-bit key and produces a stream of bits encrypted with the plaintext via an XOR operation. With a design that prioritizes security and speed, ChaCha20 is a more secure alternative to Salsa20, as well as being more efficient in dealing with cryptanalytic attacks. Its advantage lies in its ability to efficiently generate random outputs, which makes it a popular choice in data security applications such as TLS protocols, VPNs, and cryptocurrencies[27].

According to previous research methods, the Chacha20 encryption algorithm looks like Figure 2 below.

Algorithm 1: Encryption procedure

1 Obtain a new encryption key $\underline{k}$ from K_S
2 Read the plain vector $\underline{u}_j$
3 **if** $j = 0$ **then**
4 Transmit $\underline{v}_0 \leftarrow \underline{u}_0 \oplus \underline{k}$
5 Return to Line 1
6 **else if** $\underline{u}_j \neq \underline{u}_{j-1}$ **then**
7 $\underline{\sigma} \leftarrow \underline{u}_j \oplus \underline{u}_{j-1}$
8 $\underline{\sigma}_c \leftarrow \underline{\sigma} \oplus \underline{k}$
9 **if** $\underline{\sigma}_c = \underline{0}$ **then**
10 $\underline{\sigma}_c \leftarrow \underline{\sigma}_\mu \oplus \underline{k}$
11 $\underline{v}_j \leftarrow \underline{v}_{j-1} \oplus \underline{\sigma}_c$ and transmit $\underline{v}_j$
12 Return to Line 1
13 **else**
14 $\underline{v}_j \leftarrow \underline{v}_{j-1}$ and transmit $\underline{v}_j$
15 Return to Line 2

Figure 2 : Encryption[20]

In contrast, the decryption algorithm looks like Figure 3 below.

Algorithm 2: Decryption procedure

1 Obtain a new encryption key $\underline{k}$ from K_S
2 Read the cipher vector $\underline{v}_j$
3 **if** $j = 0$ **then**
4 $\underline{u}_0 \leftarrow \underline{v}_0 \oplus \underline{k}$
5 Return to Line 1
6 **else if** $\underline{v}_j \neq \underline{v}_{j-1}$ **then**
7 $\underline{\sigma}_c \leftarrow \underline{v}_j \oplus \underline{v}_{j-1}$
8 $\underline{\sigma} \leftarrow \underline{\sigma}_c \oplus \underline{k}$
9 **if** $\underline{\sigma} = \underline{\sigma}_\mu$ **then**
10 $\underline{\sigma} \leftarrow \underline{k}$
11 $\underline{u}_j \leftarrow \underline{u}_{j-1} \oplus \underline{\sigma}$
12 Return to Line 1
13 **else**
14 $\underline{u}_j \leftarrow \underline{u}_{j-1}$
15 Return to Line 2

Figure 3 : Decryption[20]

3. METHOD

The method proposed in this research is to combine the Javanese calendar system as an asymmetric key generator with Cacha20 stream encryption. An overview of the comparison of the HTTPS protocol security method with the method proposed in this study is shown in Figure 4 below.

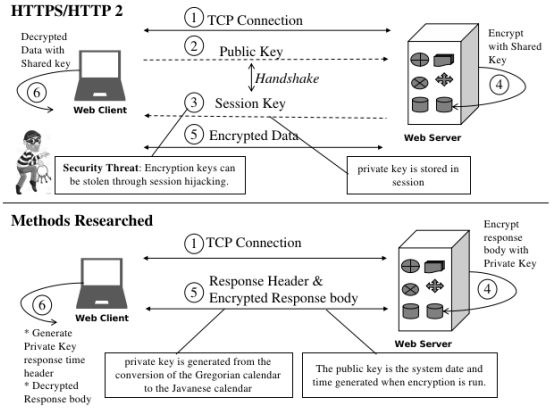


Figure 4: Illustration of Research Methods

To build web and browser applications, researchers use the Socket API library that is already available on the operating system. Where the Socket API protocol implemented is the Transfer Control Protocol (TCP).

The method to generate encryption keys, researchers utilize the date and time that are already available on the system. Where the date and time use the Gregorian calendar system. The date and time of this system are then used as a public key. The private key, is generated from the conversion ~~of~~ the Gregorian calendar to the Javanese calendar. To further increase the security of the private key, a hash process is carried out on the Javanese calendar. The flow of forming a public key, private key, and the encryption process is shown in Figure 5 below.

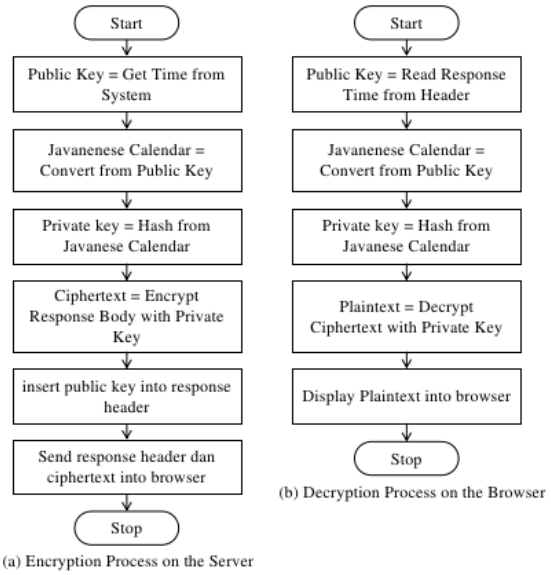


Figure 5: Encryption and Decryption Mechanism

The general description of public key generation, system date conversion to Javanese calendar, up to private key generation is shown in Figure 6 below.

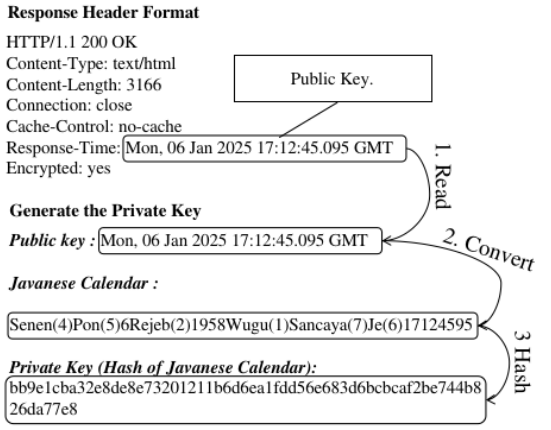


Figure 6: Encryption Key Generation

As mentioned in Section 2.2, the Javanese calendar system has cycles of 7 days, 5 days (pasaran), 35 days (wuku), 4 years (warsa), and 8 years (windu). Therefore, the arrangement of the conversion results to the Javanese calendar is shown in Figure 6 above. Meanwhile, the last few digits in numerical form represent a combination of hours, minutes, seconds, and milliseconds.

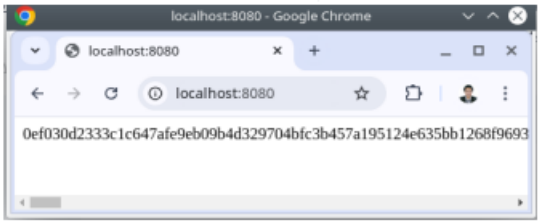
Furthermore, from the arrangement of this Javanese calendar, a randomization process (hash) is carried out using the SHA256 hash method. Where the results of SHA256 are used as the private key in Chacha20 encryption. The hash results of this private key are always unique and not the same as the previous encryption key. Due to the hash generation mechanism, a time count of up to milliseconds is included.

The web server and web browser applications are also presented in this research report. 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, it can only be run on the Linux operating system, especially Debian, Ubuntu, and Kali Linux. To be able to run on other operating systems, it still needs to be developed further.

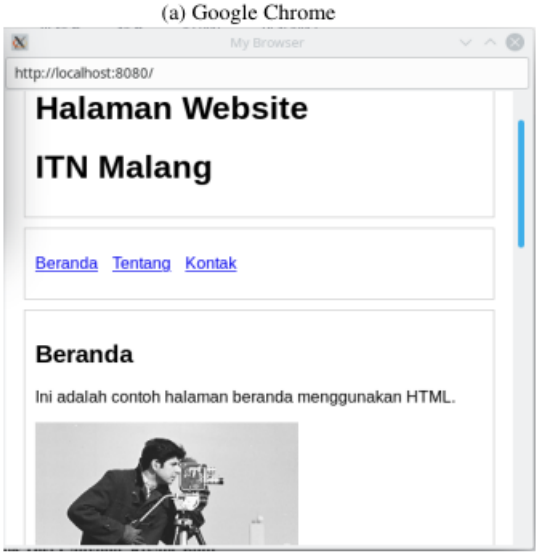
4. RESULT AND DISCUSSION

The testing mechanism in this study is divided into 3 parts, namely (1) testing the web server being studied with the Google Chrome web browser with the web browser being studied, (2) testing the speed of the web server when accessed, (3) testing the amount of data sent from the web server to the web browser (transfer size).

The test results of the trial of this research are shown in Figure 7 below.



(a) Google Chrome



(b) Research Results Browser

Figure 7: Results of Access Test from Browser

Figure 7a, shows when the research web server application is accessed from the Google Chrome web browser. On the other hand, figure 7b has shown the output once the web browser is accessed As seen in the image, it turns out that Google Chrome cannot display the website page, because Google Chrome does not have the encryption key. While the studied web browser application, successfully displays its website page.

The response body data sent from the server to the browser looks like Figure 8 below.

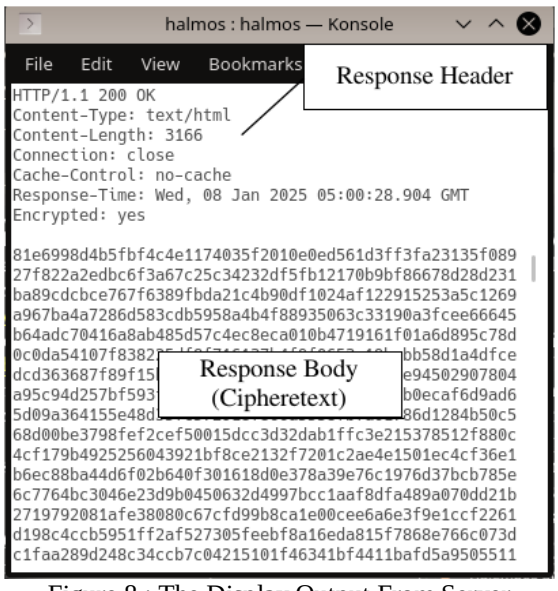


Figure 8 : The Display Output From Server

The newly proposed browser application is designed to display a response body in the form of ciphertext or plaintext. To distinguish whether the response body is in the form of ciphertext or not, the proposed browser will read the response header first. If the line Encrypted = yes is found in the response header, it means that the response body is in the form of ciphertext, and must be decrypted before being displayed. For more details, the response header reading mechanism is shown in Figure 9 below.

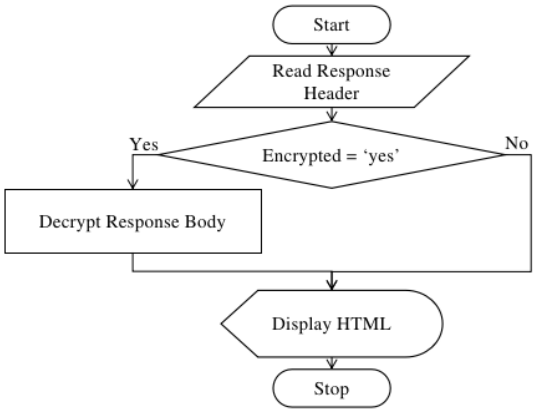


Figure 9 : Read Response Header Flags

If the browser sends important data such as username and password, then the encryption process will also be carried out on the body of the post. The request format to the server looks like the following figure 10.

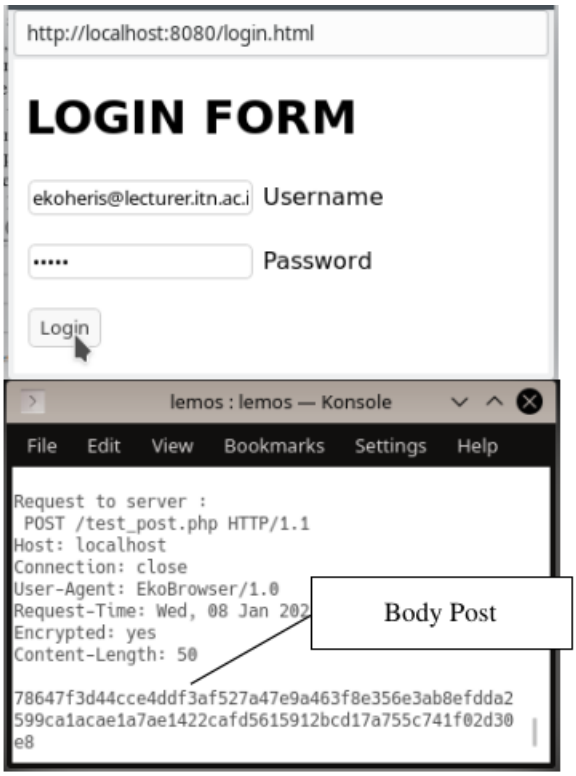


Figure 10 : Request To Server

The second test mechanism is to measure the computing time when the web server is accessed. To measure this computing time, researchers use the Apache Benchmark (ab) tool provided in the Apache web server application package. As a comparison, in this speed test it is compared with the speed of the Apache web server application. Where in the web server application, the encryption process is not carried out as in the web server application studied. The test results are as shown in Figure 11 below.

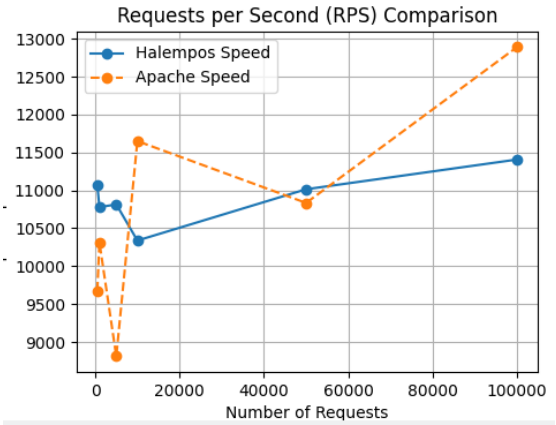


Figure 11: Test Web Server Speed

The web server speed test mechanism using Apache Benchmark (ab) involves sending a number of HTTP requests to the server to measure performance in handling a certain load. The n parameter determines the total number of requests to be sent, while the c parameter determines the number of requests made simultaneously (concurrent). To test the speed of the web server with c = 10 and the value of n varying from 500 to 10,000.

From the test results as shown in figure 11, it was found that the speed of the web server studied was superior to Apache when tested with a high number of requests. It can be seen when the number of requests is increased from 1,000 to 10,000, then the speed of the web server studied is superior. However, when the number of requests is low, namely from 50 to 500 requests, the Apache web server application is superior to the application studied. So with the evidence of the results of this speed test, it can be concluded that the key generation process to the encryption process itself has very little effect on reduce the speed of web server performance. In fact, in the application studied, the encryption process is carried out. While in Apache, the encryption process is not carried out.

The third series of tests is to measure the transfer size of data sent from the server to the browser after the encryption process has been carried out. In the application studied, the size of the data sent (transfer size) increased twofold compared to the original data as shown on figure 11. This is because the encoding mechanism after encryption uses hexadecimal code. Where 1 (one) character, if written in hexadecimal form, changes to 2 characters. The use of encoding in hexadecimal form is necessary because the results of Chacha20 encryption produce random characters whose range is outside the UTF-8 standard, or even UTF-32. Therefore, the resulting ciphertext is outside the range of the existing standard, the encoding used is hexadecimal. This increase in transfer size is a weakness of this study. So for further research, it needs to be improved by adding a data compression mechanism.

5. CONCLUSION

The asymmetric key generation mechanism through the conversion of the Gregorian calendar to the Javanese calendar successfully produces a unique key up to the millisecond level, so that each request at different milliseconds produces a different encryption key. This encryption process mechanism does not affect the speed of the web server performance, which still provides good response time even when accessed in large numbers. However, the weakness found is that the size of the data sent is doubled due to the ciphertext encoding mechanism which still uses hexadecimal. Suggestions for improvement for future research, the encryption key disguise technique still needs to be improved. Because the Javanese calendar system can still be calculated easily. While the encryption method is used, it is better to use several methods that are used alternately and randomly. Another improvement is to reduce the transfer size, a data compression method should be applied.

REFERENCE

[1] A. Afanasyev et al., “Content-based security for the web” *ACM Int. Conf. Proceeding Ser.*, vol. 26–29-Sept., pp. 49–60, 2016, doi: 10.1145/3011883.3011890.

[2] P. Sirohi, A. Agarwal, and S. Tyagi, “A comprehensive study on security attacks on SSL/TLS protocol” in *Proc. 2016 2nd Int. Conf. Next Gener. Comput. Technol. NGCT 2016*, 2017, pp. 893–898, doi: 10.1109/NGCT.2016.7877537.

[3] Z. Musliyana, M. Dwipayana, A. Helinda, and Z. Maizi, “Improvement of Data Exchange Security on HTTP using Client-side Encryption” *J. Phys. Conf. Ser.*, vol. 1019, no. 1, 2018, doi: 10.1088/1742-6596/1019/1/012073.

[4] A. R. Chordiya, S. Majumder, and A. Y. Javaid, “Man-in-the-Middle (MITM) Attack Based Hijacking of HTTP Traffic Using Open Source Tools” in *IEEE Int. Conf. Electro Inf. Technol.*, 2018, pp. 438–443, doi: 10.1109/EIT.2018.8500144.

[5] S. Hossain, A. Paul, H. Islam, and M. Atiquzzaman, "Survey of the Protection Mechanisms to the SSL-based Session Hijacking Attacks" *Netw. Protoc. Algorithms*, vol. 10, no. 1, pp. 83–108, 2018, doi: 10.5296/npa.v10i1.12478.

[6] A. Idiyatullin and P. E. Abdulkin, "A Research of MITM Attacks in Wi-Fi Networks Using Single-board Computer" in *Proc. 2021 IEEE Conf. Russ. Young Res. Electr. Electron. Eng. ElConRus 2021*, 2021, pp. 396–400, doi: 10.1109/ElConRus51938.2021.9396241.

[7] J. K. Huang, Z. X. Zhang, W. J. Li, and Y. Xin, "Assessment of the impacts of TLS vulnerabilities in the HTTPS ecosystem of China" *Procedia Comput. Sci.*, vol. 147, pp. 512–518, 2019, doi: 10.1016/j.procs.2019.01.238.

[8] Center For Strategic & International Studies (CSIS), "Significant Incidents Since 2006" 2024. [Online]. Available: https://www.csis.org

[9] J. Aas et al., "Let's encrypt: An automated certificate authority to encrypt the entire web" in *Proc. ACM Conf. Comput. Commun. Secur.*, 2019, pp. 2473–2487, doi: 10.1145/3319535.3363192.

[10] A. Taylor, "Decrypting SSL traffic: best practices for security, compliance and productivity" *Netw. Secur.*, vol. 2019, no. 8, pp. 17–19, 2019, doi: 10.1016/S1353-4858(19)30098-4.

[11] R. Palacios, A. F. Fernandez-Portillo, E. F. Sanchez-Ubeda, and P. Garcia-De-Zuniga, "HTB: A Very Effective Method to Protect Web Servers Against BREACH Attack to HTTPS" *IEEE Access*, vol. 10, pp. 40381–40390, 2022, doi: 10.1109/ACCESS.2022.3166175.

[12] M. Ozkan-Okay, A. A. Yilmaz, E. Akin, A. Aslan, and S. S. Aktug, "A Comprehensive Review of Cyber Security Vulnerabilities" *Electronics*, vol. 12, no. 1333, 2023.

[13] A. Ukpebor, J. Addy, K. Ali, and A. A. Humos, "Secure End-to-End Communications with Lightweight Cryptographic Algorithm", 2023. [Online]. Available: http://arxiv.org/pdf/2302.12994

[14] Z. Man et al., "Research on cloud dynamic public key information security based on elliptic curve and primitive Pythagoras," *Alexandria Eng. J.*, vol. 113, pp. 169–180, 2024.

[15] W. Akram et al., "Design of an Efficient and Provable Secure Key Exchange Protocol for HTTP Cookies" *Comput. Mater. Contin.*, vol. 80, no. 1, pp. 263–280, 2024, doi: 10.32604/cmc.2024.052405.

[16] I. M. Insan and F. Samopa, "Implementation of Http Security Protocol for Internet of Things Based on Digital Envelope," *Procedia Comput. Sci.*, vol. 234, pp. 1332–1339, 2024, doi: 10.1016/j.procs.2024.03.131.

[17] F. Thabit et al., "A new lightweight cryptographic algorithm for enhancing data security in cloud computing" *Glob. Transitions Proc.*, vol. 2, no. 1, pp. 91–99, 2021, doi: 10.1016/j.gltp.2021.01.013.

[18] K. Wijaya et al., "Time-Based Steganography Image with Dynamic Encryption Key Generation" *Procedia Comput. Sci.*, vol. 227, pp. 233–242, 2023, doi: 10.1016/j.procs.2023.10.521.

[19] D. A. Gede Agung et al., "Local wisdom as a model of interfaith communication in creating religious harmony in Indonesia" *Soc. Sci. Humanit. Open*, vol. 9, no. July 2023, p. 100827, 2024, doi: 10.1016/j.ssaho.2024.100827.

[20] P. M. Lima et al., "Event-based cryptography for automation networks of cyber-physical systems using the stream cipher ChaCha20" *IFAC-PapersOnLine*, vol. 55, no. 28, pp. 58–65, 2022, doi: 10.1016/j.ifacol.2022.10.324.

[21] R. Fielding et al., "Request for Comments 2068: Hypertext Transfer Protocol -- HTTP/1.1" *Network Working Group*, 1997. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2068

[22] E. Rescorla, "Request for Comments 2818: HTTP Over TLS" *Network Working Group*, 2000. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2818

[23] P. Blanco, "HTTPS is not enough" 2020. [Online]. Available: https://pblancouy.medium.com/https-is-not-enough-a1375e79ae75

[24] E. K. Sari and H. Mulyani, "Ilustrasi kalender Jawa dalam teks Wilangan Wulan, kang Becik kang Ala" *Kejawen*, vol. 2, no. 1, pp. 11–24, 2022, doi: 10.21831/kejawen.v2i1.49188.

]25] K. Demang, "Almanak Konversi Kalender Masehi ke Jawa dan Hijriyah" 2005. [Online]. Available: https://ki-demang.com/almanak/

[26] E. H. Susanto, S. Aslamiyah, and A. Adat, "Information System of Banyuwangi Indigenous Culture Agenda" vol. 7, no. 1, pp. 13–21, 2017.

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