

Implementation of AES-128 and TOTP to Enhance Single Sign-On Authentication (Case Study: SSO of the St. Pius X Parish Application, Bengkayang)

Angga Domeos Manggara¹, Yuliana², and Kristian Novando³

^{1,2,3}Information Technology, Institut Shanti Bhuana, Bengkayang, Kalimantan Barat Indonesia

Corresponding author: amanggara@shantibhuana.ac.id

ABSTRACT

Single Sign-On (SSO) is an authentication technique that utilizes a single account to access resources across various application services. This technique enhances the user authentication experience by eliminating the need to remember multiple passwords. However, SSO systems are still vulnerable to authentication attacks such as dictionary attacks and rainbow table attacks. This study focuses on the SSO system of the St. Pius X Parish Application in Bengkayang. Based on primary data, the existing authentication system lacks two-factor authentication (2FA) and still uses the MD5 hashing method for password storage. To enhance authentication in the SSO system, this research implements two-factor authentication (2FA) and the AES-128 cryptographic algorithm, by encrypting the token and utilizing Time-based One-Time Password (TOTP) for second-factor authentication. In the stage of generating a unique six-digit code, the TOTP technique provides a time-based code after the user completes the OAuth process, with a time interval of 30 seconds and a maximum of five attempts. This process is developed using the TypeScript programming language and the Next.js framework. The JWT token encryption process is executed in 11 rounds, consisting of initial rounds, main rounds, and final rounds, with key expansion carried out in each round of the AES-128 algorithm, resulting in ciphertext. The implementation of the TOTP and AES-128 methods contributes to strengthening the authentication system of the SSO application, providing protection against token hijacking attacks.

Keywords: Single Sign-On, AES-128, TOTP, 2FA, Authentication

1. INTRODUCTION

Single Sign On (SSO) is an access control method that allows users to perform a one-time login process with a single account to obtain services on the application (Senapatha, 2021). Single Sign On (SSO) addresses the problems of traditional authentication systems, namely remembering each password. The convenience of authentication using the Single Sign On (SSO) system does not eliminate the possibility of being vulnerable to attacks and exploitation on the authentication system. Testing conducted over a three-week period in March 2025 revealed vulnerabilities in the Single Sign-On (SSO) authentication system of the St. Pius X Parish application in Bengkayang, which were exploited using dictionary attack and rainbow attack methods. The research findings indicated weaknesses in the authentication system, where sensitive data was still found in plaintext form within the JSON Web Token (JWT) response. The data in plaintext includes user email, ID, and JTI. Attack attempts were also carried out using the dictionary attack method by performing brute-force attempts with a pre-collected list, and the rainbow attack test resulted in hash matches that still used MD5. The problem formulation raised in this study is how to improve the security of the Single Sign On (SSO) authentication system through the application of the AES-128 algorithm and TOTP. Based on the problems in this study, the implementation of cryptographic methods was carried out using

the AES-128 and TOTP algorithms. The implementation of the AES-128 algorithm was conducted to encrypt plaintext data from the JWT response with the addition of a salt value. The TOTP (Time-Based One-Time Password) method provides a dual-layer 2FA (Two-Factor Authentication) which generates a unique one-time code at a 30-second time interval that is set by default according to RFC 6238 to overcome brute-force attacks and enhance user convenience (Sawong et al., 2011). The implementation of the AES-128 algorithm in encrypting token data refers to the official AES document FIPS 197 (Federal Information Processing Standards Publication) (Rahman et al., 2014), while the literature on the TOTP method implementation is based on RFC 6238.

A previous study conducted by (Azhari et al., 2022) implemented encryption for document data protection, resulting in more optimal documents. (Wibawa et al., 2024) showed that TOTP implementation can reduce phishing risks by up to 90% and increase security compared to traditional authentication systems by up to 75%. In this study, the AES-128 algorithm was implemented to encrypt JWT token data using a dynamic key for each encryption round, and the TOTP implementation with a 30-second time interval was developed using the Next.js framework with the TypeScript programming language. Testing was conducted using the dictionary attack method by attempting various OTP codes from a precompiled OTP list, targeting the OTP input endpoint to evaluate the system's ability to enforce request limits (i.e., rate limiting system). The expected outcome of this test was for the system to respond with HTTP status code 429 (Too Many Requests) and 403 (Temporarily Blocked Request), requiring a 60-second delay before allowing further requests (Sunandar, 2024). In addition, a rainbow attack was carried out to verify that the use of salt in encryption effectively prevents decryption and protects against encryption cracking attempts (Saputra & Umam, 2023). The expected outcome was that encryption with salt would not allow any recovery of plaintext data. Communication traffic between the client and server was analyzed using Wireshark to capture transmitted data, and a Man-In-The-Middle (MITM) attack simulation was conducted using MITMPROXY to analyze HTTPS traffic (Auliafitri et al., 2024). This research was conducted as a reference for improving authentication systems using Single Sign On (SSO) by encrypting plaintext data and building an authentication system with dual-layer security that produces unique one-time codes within a specified time interval.

2. METHODS

The research was conducted on the existing Single Sign-On (SSO) application of St. Pius X Bengkayang Parish. The improvement process consisted of ethical hacking-based security testing of the SSO, problem identification, method implementation, and re-testing of the security. The SSO enhancement process involved the development of a Time-Based OTP with a 30-second interval, the creation of an OTP code generation function, and the delivery of the OTP code via Gmail. The implementation of SSO and the TOTP method used JWT (JSON Web Token) to verify users. Tokens containing users' sensitive data, which were still in plaintext form, were encrypted using AES-128. This study conducted both primary data collection and literature review. The primary data of this study were obtained through field observation at St. Pius X Bengkayang Parish and interviews with the parish office (Rukhmana, 2021). The literature review of this research referred to articles and journals related to SSO enhancement, the implementation of the AES-128 method, and TOTP. The process of improving the SSO system can be seen in the figure.

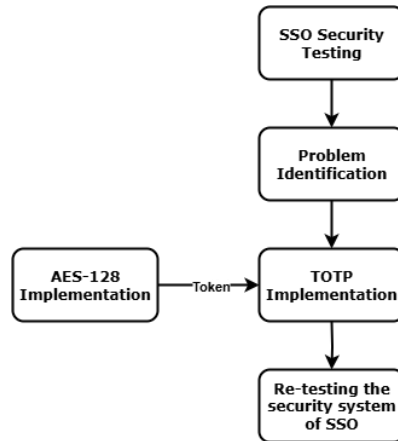


Figure 1. Research Process of SSO Enhancement

2.1. Time-Based One-Time Password Method (TOTP)

The Implementation of Time-based One-Time Password (TOTP) to Enhance Authentication Using Two-Factor Authentication (2FA) is carried out by generating a unique one-time code that is valid only within a 30-second time interval. The process of creating the workflow for implementing the TOTP method begins with the creation of middleware; the function of the middleware is to prevent Users from accessing the dashboard page before completing the OTP process. The TOTP process is carried out by new Users who do not yet have an account on the application and do not have a session. Upon successful verification in the OAuth process, the system in real-time generates an OTP and sends the OTP code via email. The OTP code that is not used within the time interval calculated during the OTP process will expire and cannot be reused, and the OTP code that has been sent can only be used once. The OTP input process performed by the User will be verified by the system to check for a match and to observe the time of OTP input. A valid OTP input process will result in an update to the User's data in the database, where "is2FVerified" is set to "true." The OTP code generation is a function built to produce a unique code. The function for sending the OTP code is performed via Gmail. OTP verification ensures that the User inputs the OTP code as sent to the email and does not exceed the 30-second time interval. During verification, if there is an incorrect code input, a function allows up to 5 attempts.

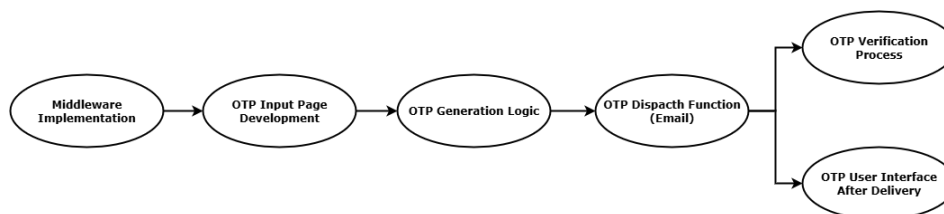
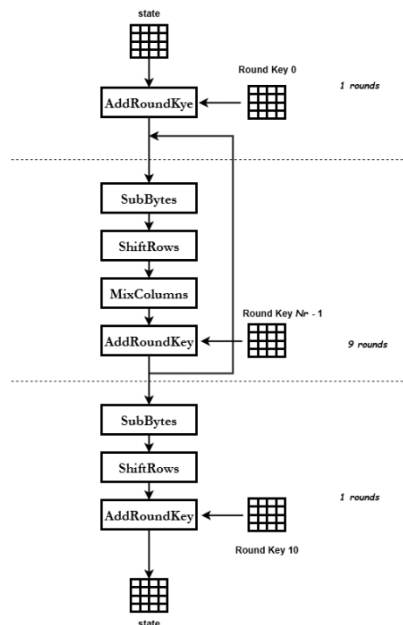


Figure 2. TOTP Implementation

2.2. AES-128 Method

The implementation of the AES-128 method is used to encrypt the JWT token response, which is still in plaintext form and contains sensitive data related to User information such as email. Encryption performed on the JWT token response provides protection in case of token

theft occurring within the authentication system; the attacker would need to decrypt the token to discover its actual contents. Therefore, the implementation of AES-128 helps enhance the Single Sign On (SSO) authentication system. The AES-128 encryption process is carried out in 11 rounds, divided into 3 stages: the initial round, main round, and final round (Cristy & Riandari, 2021). The stages in AES-128 consist of multiple rounds, each with specific processes. The initial round is the first stage, which involves converting plaintext values into ASCII (American Standard Code for Information Interchange) format stored in a 4x4 two-dimensional state matrix, containing 16 bytes, followed by an XOR (Exclusive OR) operation. The main round consists of 4 transformation processes: SubBytes, ShiftRows, MixColumn, and AddRoundKey. The main round includes 9 rounds of transformation. The final round involves a single transformation round consisting of SubBytes, ShiftRows, and AddRoundKey. In the AddRoundKey process, an XOR operation is performed between the state and the round key. The round key is a dynamic key for each round, generated during the key expansion process. The implementation of AES-128 is applied to the JWT response result, which is tested on the application's endpoint. The AES-128 implementation in this research uses a sample User on the application and uses plaintext data "namdatyu21@gmail.com" with a cipher key of "1234567890abcdef".



2.2.1. Key Expansion Formation

Key expansion is the process of generating dynamic keys used in each round of encryption. The key expansion consists of 16 bytes derived from the cipher key provided by the user. The round key consists of 11 rounds (10 round keys + 1 cipher key). The AES-128 algorithm uses a key length divided into 4 words and a block size of 4 words (Rahman et al., 2014). The division of words can be seen in the table.

Table 1. AES-128 Technical Parameters

	Key Length (<i>NK Words</i>)	Block Size (<i>NB Words</i>)	Number of Rounds (<i>NR</i>)
AES-128	4	4	10

The key length used in AES-128 is 4 words, which equals 32 bits ($32 \times 4 = 128$ bits). The word expansion to obtain the round keys results in a total of 44 words ($4 \text{ words} \times 11 \text{ rounds}$). This research uses a given cipher key to perform encryption on the email contained in the JWT response. The key expansion process uses the following formula.

$$\text{if } i \bmod 4 = 0: w[i] = w[i - 4] \oplus (\text{subWord}(\text{rotWord}(w[i - 1]))) \oplus \text{Rcon}[i/4]$$

$$\text{if } i \bmod 4 \neq 0: w[i] = w[i - 4] \oplus w[i - 1]$$

In the word expansion process, a total of 44 words (4×11) are generated, starting from $w[0]$ to $w[43]$. The formation of each round key is divided into 4 words, and for every word positioned at a multiple of 4, the following steps are performed: RotWord (left shift by 1 byte), SubWord (substitution using the S-box table), and Rcon (XOR operation with a round constant), followed by an XOR operation with the value of the word 4 positions before. For words that are not at a multiple of 4, the process involves a simple XOR between the previous word ($w[i-1]$) and the word 4 positions before ($w[i-4]$).

2.2.2. Initial Round

The Initial Round process converts the plaintext value into ASCII values which are stored in the form of a state. The transformation of the plaintext value `namdatyu21@gmail` is as follows.

[6e 61 32 6d 61 74 31 61 6d 64 79 75 40 69 67 6c]

The plaintext value `namdatyu21@gmail` consists of 20 bytes. Padding is formed using PKCS#7 (Public Key Cryptography Standards #7). AES-128 operates on a 16-byte state (Muzaki, 2023); the PKCS#7 padding in this research uses 2 states.

[2e 0c 0c 0c 63 0c 0c 0c 6f 6d 0c 0c 0c 0c 0c 0c]

Padding is applied because the plaintext value has a size of 20 bytes, thus requiring a 32-byte block to form 2 states. The remaining 12 bytes needed to complete the block are added by inserting the value 0c (12 in hexadecimal form), which is stored in the second padding state. The AddRoundKey process in the initial round requires the cipher key to be stored in the form of a state to perform the XOR operation.

2.2.3. Main Round

A. SubBytes

SubBytes performs byte substitution using the S-box table. The substitution results are stored back into the 4×4 state matrix. The byte substitution result is based on the state

obtained previously in the initial round, which is then transformed using the following S-box table.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 3. AES S BOX

Source: (Nanda et al., 2024)

B. ShiftRows

ShiftRows performs a circular left shift process, applied to rows 1, 2, and 3. The byte shifting is carried out according to the respective row: row 1 is shifted by 1 byte to the left, row 2 by 2 bytes, and row 3 by 3 bytes. The result of the byte shifting can be seen in the following matrix.

Row 0 (no shifting) $\leftarrow s_{0,0} s_{0,1} s_{0,2} s_{0,3}$

Row 1 (1-byte shift operation) $\leftarrow s_{1,0} s_{1,1} s_{1,2} s_{1,3}$

Row 2 (2-byte shift operation) $\leftarrow s_{2,0}$

Row 3 (3-byte shift operation) $\leftarrow s_{3,0} s_{2,1} s_{3,1} s_{2,2} s_{2,3} s_{3,2} s_{3,3}$

C. MixColumn

The MixColumns transformation performs a multiplication operation between the state and the AES-128 constant matrix. The multiplication used is not ordinary multiplication but rather polynomial multiplication modulo operations in $GF(2^8)$ (Rahman et al., 2014). The constant values defined by AES-128 are as follows.

[02 03 01 01 01 02 03 01 01 03 01 01 02 03 01 02]

- Multiplication by 1

$$GF_{MUL(1,a)} = a$$

- Multiplication by 2

$$GF_{MUL(2,a)} = \{a \ll 1, \text{jika nilai MSB} = 0 (a \ll 1) \oplus 0x1B \text{ jika nilai MSB} = 1$$

- Multiplication by 3

$$GF_{MUL(3,a)} = GF_{MUL(2,a)} \oplus a$$

D. AddRoundKey

The AddRoundKey transformation performs an XOR operation between the state obtained from the previous MixColumns transformation and the round key or key expansion.

$$[Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk \ Rk] \oplus [St \ St \ St \ St \ St \ St \ St \ St \ St \ St \ St \ St \ St \ St \ St \ St]$$

Note:

Rk = Round Key

St = State

The formation of KeyExpansion or dynamic key generation in this research uses a program code written in the TypeScript programming language.

```
import { S_BOX } from "../constants";

export const Rcon: number[][] = [
  [0x00, 0x00, 0x00, 0x00],
  [0x01, 0x00, 0x00, 0x00],
  [0x02, 0x00, 0x00, 0x00],
  [0x04, 0x00, 0x00, 0x00],
  [0x08, 0x00, 0x00, 0x00],
  [0x10, 0x00, 0x00, 0x00],
  [0x20, 0x00, 0x00, 0x00],
  [0x40, 0x00, 0x00, 0x00],
  [0x80, 0x00, 0x00, 0x00],
  [0x1b, 0x00, 0x00, 0x00],
  [0x36, 0x00, 0x00, 0x00],
];

export function rotWord(word: number[]): number[] {
  return [word[1], word[2], word[3], word[0]];
}

export function subWord(word: number[]): number[] {
  return word.map((byte) => S_BOX[byte]);
}

export function xorWords(a: number[], b: number[]): number[] {
  return a.map((byte, i) => byte ^ b[i]);
}

export function stringToBytes(str: string): number[] {
  return Array.from(str).map((c) => c.charCodeAt(0));
}

export function bytesToHex(bytes: number[]): string {
  return bytes.map((b) => b.toString(16).padStart(2, "0")).join("");
}

export function hexToBytes(hex: string): number[] {
  return hex.match(/.{1,2}/g)?.map((byte) => parseInt(byte, 16));
}

export function cloneMatrix(matrix: number[][]): number[][] {
  return matrix.map((row) => [...row]);
}
```

Figure 4 key expansion code function

2.3 Final Rounds

The final round transformation consists of only three transformations, namely the subBytes transformation, shiftRows, and AddRoundKey.

3. RESULTS AND DISCUSSIONS

Based on the results of the method implementation in improving the Single Sign-On (SSO) system on the Paroki St. Pius X Bengkayang application, the authentication process has a dual layer, namely the Oauth process and the OTP validation process with a 30-second interval, and includes an encryption process on the JWT token.

3.1 Result

3.1.1 Results of TOTP Implementation

The authentication result after the User successfully passes the OAuth process and the user data is successfully stored in the database, the middleware will perform a verification and redirect the User to the OTP code input page. The OTP code input page can be seen in the figure.

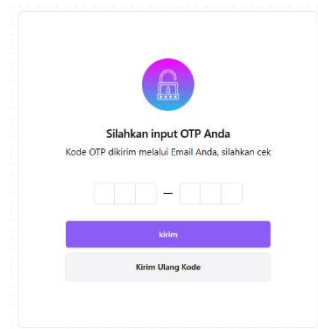


Figure 5. OTP input page

The OAuth process successfully passed by the User, the system will in real-time send an OTP code via Gmail based on the User who performed the authentication process. The result of the OTP code delivery to the User's Gmail is as follows.

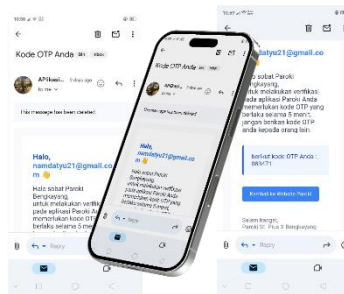


Figure 6. Result of OTP email delivery

Validation will be carried out based on the time interval of OTP code input with reference to the OTP code being sent and Gmail; validation is also carried out to check for errors in OTP code input. The following are the validation results if there is an error and the OTP code input time interval exceeds the specified limit.

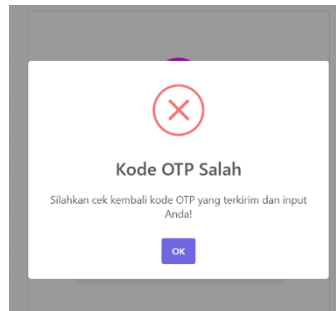


Figure 7. Invalid OTP

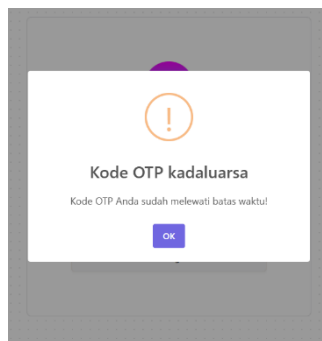


Figure 8. Expired OTP

3.1.2 The Result of AES-128 Implementation

The implementation in this study is carried out on the endpoint of the application as the application of the AES-128 method to perform token encryption. This study also uses the Postman application to conduct testing of the AES-128 implementation in performing encryption. The implementation results on the application are accurate when compared with the manual calculation results of AES-128 to ensure that the encryption implementation complies with the predetermined algorithm standards. The implementation results can be seen in the figure.

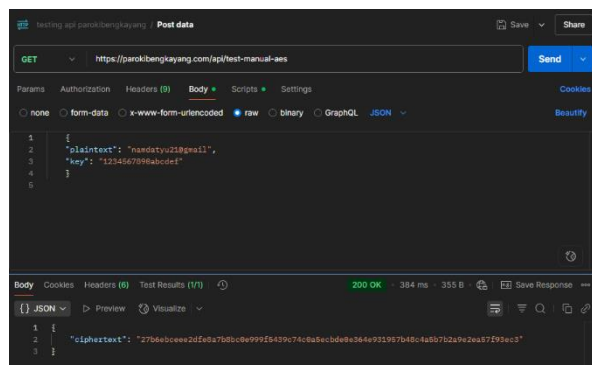


Figure 9. AES-128 implementation results

Based on the results of manual calculations and system implementation, the results are the same. The following is the result of AES-128 encryption on the token response for the email namdatyu21@gmail.com with the cipher key 1234567890abcdef.

Table 2. Final Result Table of AES-128 Encryption

State 1	27b6ebceee2dfe8a7b8bc0e999f5439c
State 2 (padding)	74c0a5ecbde0e364e931957b48c4a5b7
Final Result	27b6ebceee2dfe8a7b8bc0e999f5439c74c0a5ecbde0e364e931957b48c4a5b7

The final result of the encryption has a value of 32 hexadecimal bytes, with the addition of an 8-byte salt value, resulting in a final encryption output of 40 bytes. The salt value is a random value, and the final result of the encryption with the addition of the salt value is as follows:

Table 3. Final AES-128 Encryption Table with Salt Value

Final Result	27b6ebceee2dfe8a7b8bc0e999f5439c74c0a5ecbde0e364e931957b48c4a5b70ce70db135d5184f
-----------------	--

3.2 Discussion

Based on the results of the implementation of the TOTP and AES-128 methods used to improve authentication using the Single Sign-On (SSO) model, testing was carried out on the authentication system using the dictionary attack and rainbow attack methods. The test results show that the authentication successfully achieved improvements and is resistant to dictionary attack and rainbow attack methods. The following are the data results of the attack methods used.

3.2.1 Dictionary Attack Testing

Based on the results of the dictionary attack testing, the OTP code consistently changed at 30-second intervals and was supported by a rate limiting system that allowed a maximum of five attempts. The OTP system successfully prevented OTP code hijacking, brute-force attacks, and potential DDoS attacks targeting the OTP input page. In the test involving 100 OTP code attempts, no valid OTP codes were discovered. The system blocked 95 of the requests due to rate limiting, while 5 attempts returned incorrect OTP responses. The total duration of the testing process was recorded at 14.46 seconds. A detailed statistical summary of the test results is presented in the following table.

Table 4. Rate Limiting Testing

Number of Trial OTP	Testing Success	Requests are subject to rate limiting	Wrong OTP	Trial Time
100	0	95	5	14.46 detik

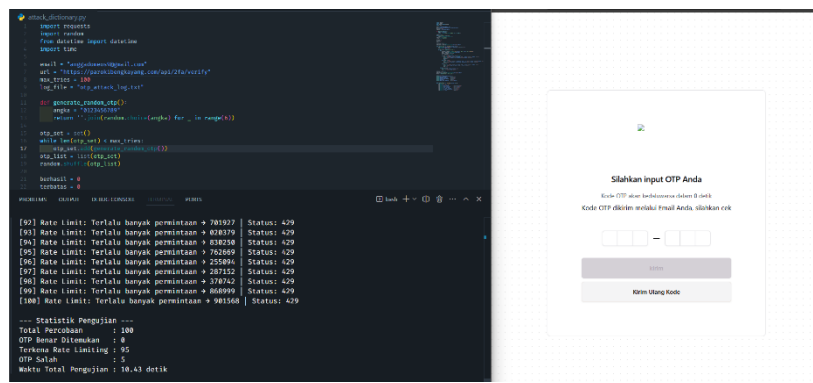


Figure 10. Detail Rate Limiting Testing

Testing was conducted on the OTP User Interface (UI), where the request results on the UI page returned a 429 response, indicating Too Many Requests. Attempts exceeding the five-attempt limit did not trigger the OTP code submission process to the server. Request logs that exceeded the attempt threshold could be observed in the application console and in the requests sent to the OTP verification endpoint.

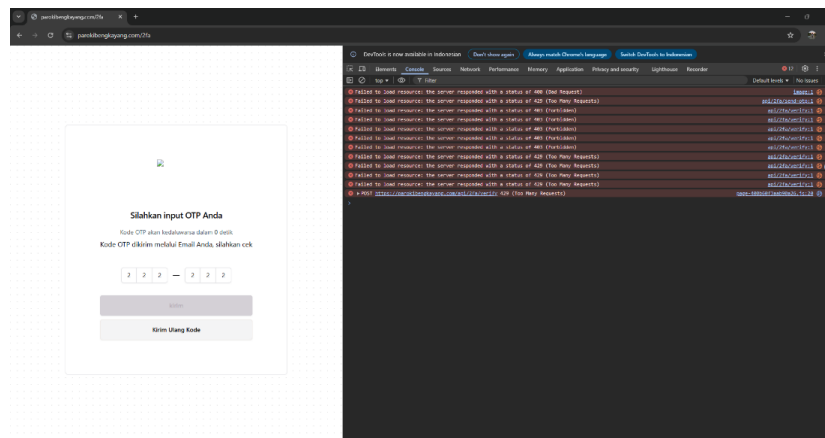


Figure 11. Console App Many Request

The results of this improvement demonstrate the system's ability to prevent brute-force attacks on the OTP input page. The implementation of rate limiting validation and the delivery of HTTP status code 429 effectively halted excessive requests at the server level, enforcing a

60-second wait time before allowing further attempts after exceeding five failed attempts. This mechanism successfully mitigates dictionary attack methods. The application of the TOTP method in this study has proven to enhance the authentication security of the Single Sign-On (SSO) system. Further dictionary attack testing was conducted using Burp Suite, simulating brute-force attempts with 1,000 OTP codes prepared in a .txt list file. The server responses indicated the issuance of status codes 429 and 403, showing that the system accurately detected dictionary attack attempts. Code 429 confirmed that the rate limiting mechanism was functioning correctly by temporarily blocking requests, enforcing a request interval, and only permitting requests in batches of five, in accordance with the OTP input configuration.

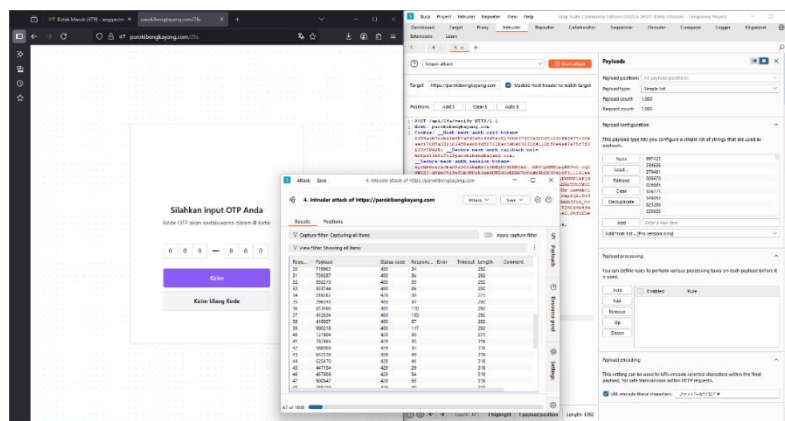


Figure 12. Result Burp Suite Testing

3.2.2 Rainbow Attack Testing

Rainbow attack testing performs a matching between encryption data and plaintext data. The testing is conducted using a Python program with a scenario of existing cipherkey data that corresponds to the AES program that was built. Based on the result of encrypting the plaintext using the AES-128 algorithm, which produces the encrypted data

“27b6ebceee2dfe8a7b8bc0e999f5439c74c0a5ecbd0e364e931957b48c4a5b7”, a salt value is added—namely a unique dynamic code consisting of 8 bytes in hexadecimal format—which is appended to the encrypted result so that the total length of the encrypted result becomes 40 bytes. The testing shows that it was not successful in obtaining the ciphertext data with the addition of the salt value; the testing could only discover the ciphertext data because the AES algorithm used and implemented in the system is a variant of AES with ECB (Electronic Codebook) mode. The following is the result of the rainbow attack testing: the final encryption result is

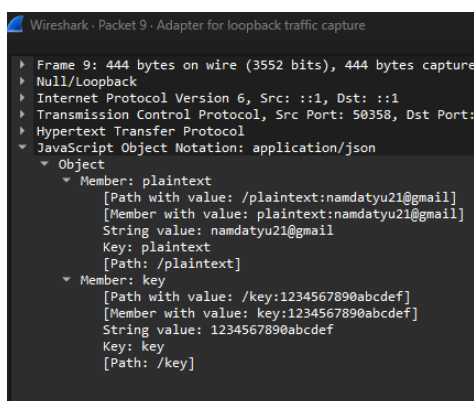
“27b6ebceee2dfe8a7b8bc0e999f5439c74c0a5ecbde0e364e931957b48c4a5b70ce70db135d5184f”.

Table 5. Rainbow Attack Method Testing Results

plaintext	enkripsi	information
admin@gmail.com	054888d3c9b45b2f9f544b6e54980e86	fail
test@gmail.com	8fdaeba859a9d970103ee15cdf663df1	fail
namdatyu21@gmail.com	27b6ebceee2dfe8a7b8bc0e999f5439c74c0a5ecbde0e364e931957b48c4a5b7	it works but not with the salt value

3.2.3 Wireshark Testing Results

Wireshark testing is used for analyzing the network traffic of client and server communication. Wireshark testing is carried out in a development environment running on the Http protocol. Based on the test results, data can be captured because it is not encrypted; this result indicates the importance of the Https protocol for data encryption in client and server communication traffic.



3.2.4 MITMPROXY Testing Results

The testing also conducted a simulation of a MITM (Man-In-the-Middle) attack performed in a production environment. The MITM attack was used to demonstrate the importance of the encryption layer on the token and two-factor authentication (2FA). The implementation of MITMPROXY testing was carried out by performing trust certificate injection. The attack can be executed if the user knowingly or unknowingly downloads the certificate. Based on the testing, the system is more vulnerable if there is no encryption and no additional layer on the authentication system. The application of AES-128 is used to encrypt the token, so if there is a successful unauthorized token theft, it will still fail due to OTP validation failure.

```
Flow Details
2025-07-01 15:52:46 GET https://parokibengkayang.com/api/
+ 200 OK application/json 97b 172

Request
Content-Type: application/json
User-Agent: PostmanRuntime/7.44.1
Accept: */*
Cache-Control: no-cache
Postman-Token: 7039b448-6ac0-4415-a53e-b2ef9bb4f152
Host: parokibengkayang.com
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 83

JSON
{
  "plaintext": "namdatyu21@gmail",
  "key": "1234567890abcdef"
}
```

3.2.5 Testing with Honeypot

The creation of a honeypot is used as a decoy system implemented on a fake OTP input endpoint. The purpose of creating the honeypot is to detect brute force attempts on the OTP code by collecting information such as IP address, attempted OTP codes, and timestamps. This information is used to analyze attack techniques and methods.

```
/home/rxu/.pm2/logs/appchurchbky-out.log last 15 lines:
0|appchurc | JWT Decoded: Tidak Ada
0|appchurc | -----fungsi middleware dipanggil-----
0|appchurc | PATH: /wp-admin/setup-config.php
0|appchurc | TOKEN (JWT): null
0|appchurc | JWT Decoded: Tidak Ada
0|appchurc | [HONEYPOT] Akses mencurigakan:
0|appchurc | IP: ::ffff:127.0.0.1
0|appchurc | OTP: 212121
0|appchurc | Time: 2025-07-02T02:21:25.104Z
0|appchurc |
0|appchurc | [HONEYPOT] Akses mencurigakan:
0|appchurc | IP: ::1
0|appchurc | OTP: unknown
0|appchurc | Time: 2025-07-02T02:22:18.605Z
0|appchurc |
0|appchurc | [HONEYPOT] Akses mencurigakan:
0|appchurc | IP: ::ffff:127.0.0.1
0|appchurc | OTP: 080808
0|appchurc | Time: 2025-07-02T02:27:52.842Z
0|appchurc |
0|appchurc | [HONEYPOT] Akses mencurigakan:
0|appchurc | IP: ::1
0|appchurc | OTP: 211716
0|appchurc | Time: 2025-07-02T02:30:19.736Z
```

4. CONCLUSION

Based on the conclusions that have been described, the suggestions provided for conducting further research related to this study are as follows:

1. The implementation of AES-128 for encryption can use a stronger AES variation, namely CBC (Cipher Block Chaining). This study uses ECB (Electronic Codebook), so from this variation, the rainbow attack method can find the input and output of the encryption.
2. The AES-128 method performs encryption of the JWT token data with a predetermined endpoint. AES-128 implemented in this system uses the AES-128 (ECB - Electronic Codebook) variation.
3. The testing results with the dictionary attack method, namely performing brute force with a range of 1 million OTP code attempts, failed 100% to find the OTP code. The TOTP process can limit or rate-limit OTP code input attempts to 5 attempts.

4. Testing with the rainbow attack method, namely matching encryption data with plaintext in a scenario where the cipher key data is found, yielded results showing that the rainbow attack succeeded in finding the plaintext data based on the encryption data. However, it did not find the encryption data that had been added with a salt value consisting of 8 bytes, which was added to the encryption data so that the encryption data was no longer 32 bytes but 40 bytes.

REFERENCES

- Auliafitri, D., RizkiSuro, E., Malik, M. R. M., & Setiawan, A. (2024). Optimalisasi Pengujian Penetrasi: Penerapan Serangan MITM (Man in the Middle Attack) menggunakan Websploit. *Journal of Internet and Software Engineering*, 1(3), 12. <https://doi.org/10.47134/pjise.v1i3.2620>
- Azhari, M., Mulyana, D. I., Perwitosari, F. J., & Ali, F. (2022). Implementasi Pengamanan Data pada Dokumen Menggunakan Algoritma Kriptografi Advanced Encryption Standard (AES). *Jurnal Pendidikan Sains Dan Komputer*, 2(01), 163–171. <https://doi.org/10.47709/jpsk.v2i01.1390>
- Cristy, N., & Riandari, F. (2021). Implementasi Metode Advanced Encryption Standard (AES 128 Bit) Untuk Mengamankan Data Keuangan. *Jurnal Ilmu Komputer Dan Sistem Informasi (JIKOMSI)*, 4(2), 75–85. <https://ejournal.sisfokomtek.org/index.php/jikom/article/view/181>
- Muzaki, D. F. (2023). *Penerapan Kriptografi dalam Pengamanan Data at Rest Level Filesystem melalui Program Encrypted File Explorer untuk Penggunaan Sehari-hari*.
- Nanda, D., Herlambang, R., Pravitasari, N., Korespondensi, P., & Pendahuluan, I. (2024). Penerapan Kriptografi AES untuk Keamanan Data Aplikasi Pemesanan Bibit Ternak pada BPSI UAT. *Remik*, 8, 29–44.
- Rahman, A. U., Miah, S. U., & Azad, S. (2014). Advanced encryption standard. *Practical Cryptography: Algorithms and Implementations Using C++*, 91–126. <https://doi.org/10.1201/b17707>
- Rukhmana, T. (2021). Jurnal Edu Research Indonesian Institute For Corporate Learning And Studies (IICLS) Page 25. *Jurnal Edu Research : Indonesian Institute For Corporate Learning And Studies (IICLS)*, 2(2), 28–33.
- Saputra, M. B., & Umam, C. (2023). Distributed MD5 Brute Force Using Message Passing Interface (MPI) on ARM Architecture. *Jurnal Transformatika*, 20(2), 105. <https://doi.org/10.26623/transformatika.v20i2.5630>
- Sawong, K. S. A., Andrias, D. R., Muniroh, L., Reddy, C., Purnawita, W., Rahayu, W. P., Nurjanah, S., & RI, K. (2011). TOTP: Time-Based One-Time Password Algorithm. *RFC - Informational*, 53(9), 167–169.
- Senapatha, I. K. D. (2021). Implementasi Single Sign-On Menggunakan Google Identity, REST dan OAuth 2.0 Berbasis Scrum. *Jurnal Teknik Informatika Dan Sistem Informasi*, 7(2), 307–320. <https://doi.org/10.28932/jutisi.v7i2.3437>

- Sunandar, A. (2024). Implementasi Penetration Testing Dan Wordlist Generator Dalam Pengujian Keamanan Jaringan Menggunakan Metode Dictionary Attack. *JATI (Jurnal Mahasiswa Teknik Informatika)*, 8(3), 3368–3372. <https://doi.org/10.36040/jati.v8i3.9685>
- Wibawa, S., Suryanto, S., & Ningsih, R. (2024). Perlindungan Data Digital Dengan Time-Based One-Time Password (TOTP). *INSANtek*, 5(1), 30–36. <https://doi.org/10.31294/insantek.v5i1.3495>