Indonesian Journal of Computing, Engineering, and Design

Journal homepage: http://ojs.sampoernauniversity.ac.id/index.php/IJOCED

JSON Web Token Leakage Avoidance using Token Split and Concatenate in RSA256

Malvin[1], Cutifa Safitri [1*]

[1] Faculty of Computing, President University, Bekasi, West Java 17530, Indonesia
Corresponding email: cutifa@president.ac.id

ABSTRACT

This research aims to protect users from JWT (JSON Web Token) leakage, which is listed plainly in the Response Header in the web browser console. The risk of malicious attackers stealing the JWT is highly dangerous since the API (Application Programming Interface) will be within the control of malicious attackers, leading to identity theft and data abuse due to the JWT leakage. As a solution, this paper proposed a method in which the JWT bearer token will be split, separately encrypted by RSA256, and concatenated into a new unique token to limit attacker accessibility towards the JWT token. The benefit of this proposed method is envisaged to achieve a more secure web application for user data protection and security optimization. The idea of this method is to modify the bearer token by splitting, encrypting, and concatenating it to be a unique token. The product of the encryption process is an unrecognizable token in the form of letters and punctuation which attackers cannot manipulate. The encrypted code will be returned to the initial location in Response Header. After testing, it is proven that modifying the bearer token by splitting and concatenating provides more security to a web application.

ARTICLE INFO

Article History:
Received 26 Oct 2022
Revised 22 Feb 2023
Accepted 02 Mar 2023
Available online 03 Apr 2023

Keywords:
Application programming interface,
Authentication,
JSON web token,
RSA256.

1. INTRODUCTION

In this Web 3.0 era, web applications have been used in every part of our lives, from e-banking, e-commerce, e-learning, and even ordering food or massage service. Web application is an inseparable part of everyday life. Besides, web applications undoubtedly, increase the efficiency and effectiveness of people of all ages. Amid all that progress, security threats have been a concerning issue. Security experts and malicious actors are engaged in a never-ending data war. Data breaches appear to be occurring more frequently overall, costing more money. In the US, approximately US$ 8.64 million was lost due to data breaches in 2020. Surprisingly, 96% of web applications have well-known flaws and oddities (Adam, Moedjahedy & Maramis, 2020). Businesses should consider security during application development to offer adequate protection against web application security threats.

One of the threats that users must pay attention to is stealing JWT. A web application that utilizes JWT as the security standard must be cautious. JWT, an open standard that enables the exchange of security-related data between a client and a server, is susceptible to theft. JWT is laid explicitly in the Response Header. A malicious user can steal the JWT and access the API without authentication. This condition is fatal since the malicious user can Create, Read, Update, Delete, and data of the API, which is identity theft and privacy violation. Imagine if this condition happens in an e-banking web application. The malicious user will access the customer's personal information and the money inside the bank account. This study aims to prevent the above incident from happening.

State of the art in this study to prevent data leaks towards user credentials is through a proposed technique that allows the JWT to be hashed in a manner so that malicious attackers who intend to steal the JWT cannot use it to access the API. The proposed method suggests using the RSA256 hashing algorithm. RSA is a frequently used asymmetric cryptosystem. The difficulty of the factorization issue determines the strength. The factors for RSA were 64, 96, 128, 160, 192, 224, and 256 bits. This study intends to employ RSA256, which is more secure, and if the keys are compromised, the user may immediately rotate them. A token's signer enables the token's recipient to confirm that the token's content hasn't been altered and ensure that the signer is the token's original creator. Although JWTs are signed, their data is still viewable. Only the non-change of the JWT's content can be verified using signatures. The JWT's signature proves that its contents have not changed since the JWT was first issued (as it would, for instance, if there were direct attacks). Signatures are generated by mixing encoded versions of a JWT's header and payload and passing them, together with the secret, as arguments into the header-defined method. Further explanations of this method will be described in the following chapters.

Several works of literature are analyzed and discussed to get the idea of authentication in a web application, hashing algorithm, and how it is applied to the web. Below is the explanation that is divided into several sub-part.

1.1. Authentication

Idrus et al. defined authentication as proof the claimant gives information to assert and monitor that they correspond to the provided identity (Idrus and Zulkarnain, 2013). Three entities are involved in the authentication process: the claimant, the provider, and the Information System. Idrus et al. believe that there are five factors to authenticate humans: something the user knows; something the user owns; something that qualifies the user; something the user can do, and where the user is.

There are various authentication methods, such as passwords, OTPs, radio frequency identification, biometrics, etc. Other than password and biometrics, Token Presence can also be utilized to supplement the authentication requirements. Token Presence can be a physical token (such as a card, etc.) and a one-time software-generated password. Several State-of-the-art and potential MFA (Multi-factor authentication) sources are developed to support the utilization of this technology in securing data and devices such as behavior detection, Beam-Forming Techniques, Electrocardiographic Recognition, Electroencephalographic Recognition, and DNA Recognition (Ometov et al., 2018).

Multiple methods can be used to achieve higher security. There are several criteria for the methods: the observed popularity, the device cost, or the cost of per-user equipment, the infrastructure cost, or the cost of the setting up of such an authentication method, the ease of use, and the general security (van der Horst & Seamons, 2007). Based on these criteria, biometrics is one of the finest ways to authenticate since the security is very safe and effective. However, the cost of setting up biometric authentication is high. In conclusion, all authentication method has its advantage and disadvantage. The selection of suitable security technique need to consider the sensitivity level of user information

1.2. Web Based Authentication

There are multiple ways to perform web-based authentication. For this research, this study has reviewed several web-based authentications such as FSUA (Frictionless and Secure User Authentication), SSH (Secure Shell), SSO (Single Sign On), Beam Auth, and JWT (JSON Web Token). Password-based authentication is insufficient since the same password is reused on many sites. Malicious web users can breach the password by brute force (repeatedly guessing the password) and credential-stuffing (trying a known credential from one website to another). On the other hand, MFA is inconvenient since it takes many steps to authenticate. Timothy Baron et al. propose using web tripwires and login rituals, which are complementary. To remain trusted by the app, one party explains what steps the user should take, while the other explains what they should not.

Web tripwires are deceptive intrusion detection mechanisms specific to every user, allowing them to customize their intrusion detection traps. On the other hand, login rituals are user actions that must be taken immediately after an initial authentication (Barron et al., 2021).

FSUA works when the user submits the username, password, OTP, and certificate. When the user is verified, FSUA will match the data with the profiler and historical data through the selected mechanism until it is successfully login. There are no overhead costs because the suggested system was created using the frictionless methodology (Olanrewaju et al., 2021).

SSH is a JWT profile for least-privilege OAuth tokens for distributed scientific computing and open-source implementation. SSH uses digitally signed, self-describing JWTs instead of opaque tokens to allow SSH-enabled services to locally verify tokens from many issuers to support large-scale distributed scientific computing environments (Park, 2019). The authorization server issues a token to a user to begin the verification process. After getting a token from the authorization server, the user can cut and paste the token into any SSH client application to respond to the host's question. However, there is a possibility of leakage of a token. As a countermeasure, the SciTokens SSH implementation is very cautious about what data is reported, only logging enough information to allow administrators and users to troubleshoot connection issues (Gao et al., 2020)

On the other hand, there is SSO, a desktop application that provides security to computers hosting COSMOS. COSMOS utilizes JWT as one of its ways to authenticate users. Users log in using Keycloak, and a JWT token is returned. The frontend application can use this token to authenticate API calls. Since the access token is signed, it can be cryptographically verified that it was generated by the Keycloak system and is unchanged. By using JWT, any information about the user can be stored in Keycloak outside of the application.

Additionally, the JWT access token is irrevocable, signifying that the token is valid until the current time reaches the token's expiration time, and it can successfully conduct API actions within the system. The benefit of using JWT here is that it provides added security to less common but more critical commanding/scripting actions (Melton, 2021; Soh & Joy, 2003).

This research has also discussed BeamAuth, a two-factor web authentication method in which a specifically created bookmark serves as the second factor. BeamAuth offers two intriguing features: any contemporary, out-of-the-box web browser can be used on the client side with simple server-side deployment, and even if the user neglects to check the necessary user interface indicators, credentials are still safe from many different sorts of phishing attacks. Any login-protected web server may deploy BeamAuth immediately with minimal effort, and it does not hinder or conflict with existing anti-phishing measures. BeamAuth is intended and implemented as a two-factor authentication method to counter various phishing attempts using only existing features of HTTP and contemporary web browsers. BeamAuth is thought to increase the difficulty of the most prevalent phishing attacks significantly. The method is best for high-value websites, especially single sign-on websites, since the user will likely make an extra minor effort during registration because it takes bookmark-bar real estate and a browser setup procedure. BeamAuth takes advantage of the URL Fragment Identifier's peculiar characteristics, including the fact that it is never communicated over the network and that altering it does not cause a page to reload (Adida, 2007; Ahmed & Lee, 2017).

Finally, JSON Web Token (JWT) is an open standard that enables the communication of security-related information between a client and a server. Each JWT contains a collection of encoded JSON objects, such as claims (Adam, Moedjahedy & Maramis, 2020). JWTs are signed with a cryptographic method to guarantee that the claims cannot be altered after issuing the token (Akana, 2022). JWT secures data that is a JSON object by digitally signing it using algorithms like HMAC (Hash-based Message Authentication Code) and encoding it as a string (Sakimura, Bradley & Jones, 2022). JWT consists of three main parts: the Header, Payload, and Signature. The Header stores the type of token and signing algorithm used on a token. The component known as the Payload is where the actual data for a given transaction is kept. The final part, called the Signature, consists of the encoded portions of the Header and Payload and a "secret" that can be any string (Pramono & Javista, 2021).

(1) The user will log in using their username and password. Then (2), the authentication service will validate the username and password and generate the user ID. (3) Subsequently, the authentication service will generate the JSON Web token by executing the JWT function called "sign," using the user ID and the private key that has been stored by default on the system as parameters. (4) The generated token will be sent to the user through the client side by attaching it to the Response Header. (5) The client will store the transported token in the local storage. On every user request, (6) the token will be sent to the server side to be verified by attaching it to the Request Header. Afterwards, (7) the server side will execute another JWT function called "verify" to verify and generate the original value, which is the user ID, using the token and public key that has been stored by default on the system as parameters. When validated (8), the user can access the selected operation. A more thorough explanation can be seen in **Figure 1.**

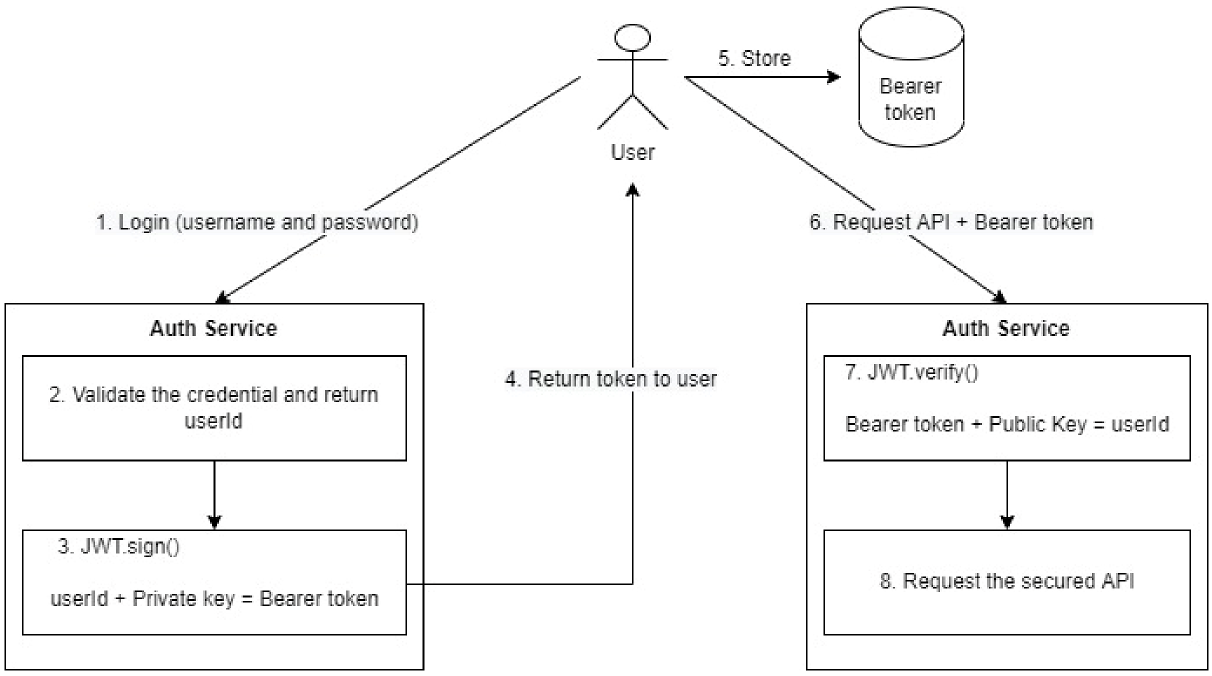
User
5. Store
Bearer token
1. Login (username and password)
6. Request API + Bearer token
Auth Service
2. Validate the credential and return userId
3. JWT.sign()
userId + Private key = Bearer token
4. Return token to user
Auth Service
7. JWT.verify()
Bearer token + Public Key = userId
8. Request the secured API

Figure 1. JSON Web Token Workflow

1.3. Hash-Based Authentication

The way hashing works in authentication is that when users create a new account and enter their password, the application code hashes the password and stores the result in the database. The hashing algorithms that have been analyzed are SHA, RSA, and AES. SHA-1 is a scheme proposed by Peyavian and Zunic which protects password transmission over insecure networks. The weakness of this scheme is that the user can randomly use an ID and password, which makes the scheme vulnerable to guessing attacks. Users can change their password over an insecure network, and unauthorized users can intercept messages and determine a potential password. Most user passwords are meaningful (found in dictionaries) or short strings (i.e., 8 bytes) for simple memory, making the guessing attack computationally feasible. The servers now calculate the authentication formula and compare the user's password to the verification table during the phase where the password is changed. If the value is the same, the server requests the user to modify the password and substitutes the new password for the old one in the verification table. In this approach, it is challenging to guess the user's password (Lee et al., 2002).

RSA is a widely used asymmetric cryptosystem whose strength is determined by the factorization issue. The factors for RSA are 64, 96, 128, 160, 192, 224, and 256 bits, and for this study, RSA256 will be used because it is more secure, and if the keys are compromised, the user can rotate them immediately. JSON Web Tokens are signed when they are created, and the token's signer allows the token's recipient to confirm that the token's content has not been altered and that the signer is the token's original creator. Although JWTs are signed, their data is still viewable. Only the non-change of the JWT's content can be verified using signatures. The JWT's signature proves that its contents have not changed since the JWT was first issued, as it would, for example, in the case of direct attacks. Signatures are generated by mixing encoded versions of a JWT's header and payload and passing them, together with the secret, as arguments into the header-defined method. (Zhang & Zhu, 2006; MacKenzie et al., 2000).

Secure hash functions, such as AES-hash, accept arbitrary bit strings as input and output strings of a defined length (in this case, 256 bits) (Dhamija, 2000). While AES-hash may parallelize to a certain extent, allowing key setups and encryptions to happen simultaneously, file hashing must be done serially, although it only needs to be done once. Secure hash modes don't require keying material, but keyed variations are easily attainable (Kogan et al., 2017). AES-hash only needs to keep one block of the hashed file in memory at a time and requires only a minimum fixed amount of memory to store its Hi values. AES-hash can operate on any bitstring to be helpful in as many different applications as possible. As a form of the compression process, all secure hash methods reduce everything to a small, fixed size; in this case, 256 bits (Sasaki, 2011) (Syamsuddin et al., 2008).

1.4. Web-based Authentication Using Hash

Merkle Hash Tree can verify query results from an untrusted database server by providing trusted information. The process involves building a Merkle tree structure for a specific site using hashes for resources and their canonical web paths. A normalized version of the resource URL, known as a canonical web path, eliminates the hostname and scheme parts and decodes escape sequences, among other things (Mainanwal, Gupta & Upadhayay, 2015). Merkle Tree proves to be feasible and practical in authenticating untrusted web-serving infrastructures. However, there is no detailed report on the Merkle Tree implementation. More implementation pseudo-code for the verification processes is required, as well as research into Merkle tree upkeep, tests, and experiences with actual implementations (Bayardo & Sorensen, 2005).

On the other hand, a technique known as the single block hash function is used to confirm a web user's identification. A single-block hash function can withstand typical attacks such as replay attacks, eavesdropping, and message alteration. The single-block hash algorithm calls the random number to the background server point when the user enters their ID and password (Mallik et al., 2019). When the authentication procedure detects the same equivalent value after the server changes the password, the server will assume that the user is legitimate and grant access to its request. The implementation showed that the single-block hash function is more effective than the widely used MD5 technique (Wang et al., 2013) (Chatterjee & Prinz, 2022).

1.5. Web Application

A web application is computer software that utilizes web technology and web browsers to operate over the Internet. Web applications use a combination of client-side scripts (JavaScript and HTML) and server-side scripts to manage information storage and retrieval (PHP and ASP). Typically, web applications are written in a language that is supported by a web browser, such as HTML or JavaScript, as these languages rely on the browser to make the program executable. The web application requires a web server to receive client requests, an application server to complete the tasks, and occasionally a database to store the data. Application servers employ the following technologies: ASP.NET, ASP, ColdFusion, PHP, and JSP.

The advantages of a web application are that it works on various devices and operating systems if the browser is compatible. There are no compatibility difficulties because everyone accesses the same version. Additionally, there are no space restrictions because they are not stored on the hard disk. In subscription-based web applications, software piracy is less of a concern (i.e., SaaS). Due to fewer corporate support and maintenance needs, and fewer demands on the end user's PC, web applications reduce expenses for both the customer and the business (Stackpath, 2022).

The increasing internet use by businesses and individuals has altered how enterprises are run. As enterprises migrate from traditional models to cloud-based and grid models, web apps are now commonly used. Web apps help businesses streamline operations, increase productivity, and save costs.

1.6. REST and SOAP API Differentiation

The most commonly used access protocols for web services are SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) APIs. SOAP is a standard communication protocol system that enables processes running on different operating systems, such as Linux and Windows, to communicate over HTTP and XML. Using SOAP-based APIs, records such as accounts, passwords, leads, and custom objects can be created, retrieved, updated, and deleted. SOAP APIs enable developers to easily control online services and receive responses without worrying about languages or platforms since all operating systems are handle HTTP and related XML.

REST is a web service architectural design that acts as a communication channel between various internet-connected devices or systems. Application programming interfaces supported by the REST architectural framework are referred to as REST APIs. REST API-compliant online services, database systems, and computer systems enable requesting systems to acquire robust access and redefine representations of web-based resources by offering a predetermined set of stateless protocols and standard procedures.

The differences between SOAP and REST are that since REST API is an architectural approach, there is no official standard for it. Conversely, because SOAP API is a protocol, it has a recognized standard. Second, compared to SOAP, which uses XML to create the Payload and produces a large-sized file, REST API deploys various standards. Therefore, it requires less bandwidth and resources. Finally, JavaScript makes REST APIs more practical and their implementation more straightforward. Although SOAP APIs work well with JavaScript, they do not support extensive implementation (Malik, 2017).

2. RESEARCH METHODOLOGY

This section is divided into two parts. Firstly, a detailed discussion of the tools used for this study is provided. Secondly, the proposed JSON web token leakage avoidance method using token split and concatenate in RSA256 will be explained.

2.1. Tools

JetBrains IntelliJ was utilized as the integrated development environment (IDE) for creating applications using various languages such as Java, Kotlin, and other JAR-based languages (Syamsuddin et al., 2008). The programming language employed in this study is Java, and the framework used for illustration is SpringBoot, an open-source, microservice-based Java web framework. As mentioned earlier, JWT will be employed for security authentication. A JWT consists of a set of encoded JSON objects, including claims. These objects are digitally signed using algorithms like HMAC (Hash-based Message Authentication Code) and encoded as a string to ensure that the claims cannot be altered after the token is issued (Venkatesha et al., 2019).

JWT provides security for data in the form of a JSON object.

Additionally, Postman was used to executing the API. Postman is a platform for creating and utilizing APIs (Ghaly & Abdullah, 2021). It uses a graphical user interface to test HTTP requests and provide the user with a range of responses that must be verified. Postman supports numerous endpoint interaction methods, including GET (to obtain information), POST (to add information), PUT (to replace information), PATCH (to update certain information), and DELETE (to delete data). A summary of these methods is provided in **Table 1**.

2.2. Proposed Method

In a web application, a vulnerable point that is particularly susceptible to attack is the JSON Web Token (JWT) bearer token. This token is written in the Response Header in an "Inspect" form, leaving it open to potential theft by malicious attackers. Suppose an attacker were to succeed in stealing the bearer token using any of the methods mentioned earlier, they could potentially access the web application through POSTMAN. This would be a significant security breach, as the attacker would be able to access the web application without needing to authenticate with a username and password. The attacker could perform various functions using POSTMAN, including Get, Post, Put, Patch, and Delete. In other words, an attacker in possession of the JWT could access the entire web application and operate it as if they were the owner of the application, allowing them to access all personal information stored within the application and creating a potentially dangerous situation.

The present study has implemented the eight JSON Web Token (JWT) architecture steps. The process commences with the user's authentication, where (1) the user logs in to the client application by providing their username and password. Subsequently, (2) the client transmits the relevant information to the server for processing. Upon receiving the data, (3) the JWT security method is invoked to generate the bearer token using a private key and the provided information.

The study proposes an additional step to the existing JWT architecture, specifically splitting the token into two parts, which are subsequently encrypted using RSA256 with a 1024-bit key size after the generation of the JWT security token (4). Once the token is processed, it will be placed in the response header and sent to the client. The token will be returned to the client (5) and placed in the response header (6). Subsequently, the token will be used to access the secured API on every request to the server, with the JWT token being transmitted on each request (7 and 8).

Table 1. The Tools that are Utilized in this Research

ITEM	NEEDED
JetBrains IntelliJ	Create computer applications using various programming languages.
Java	Write programs that direct a computer to perform specific calculations or coordinate the transfer of control between external devices.
SpringBoot	Develop applications based on the program.
JWT	Ensure secure online information transfer between two parties.
Postman	Build a platform for creating and utilizing APIs.

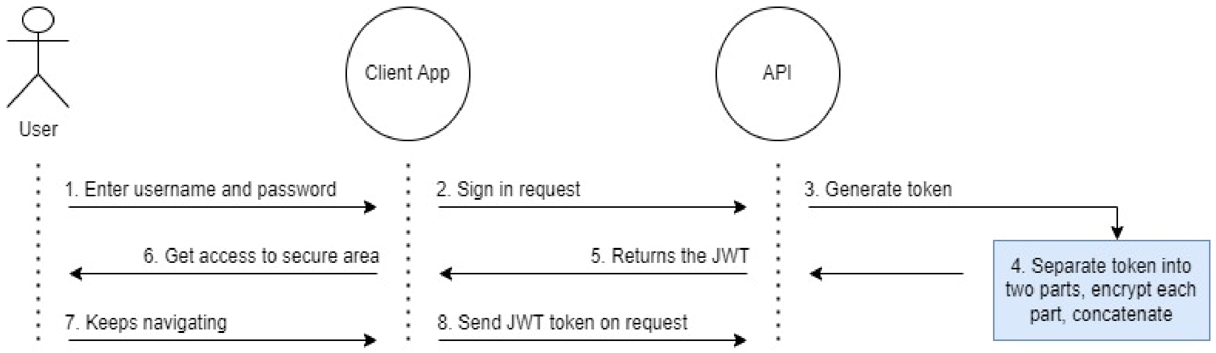
User
Client App
API
1. Enter username and password
2. Sign in request
3. Generate token
4. Separate token into two parts, encrypt each part, concatenate
5. Returns the JWT
6. Get access to secure area
7. Keeps navigating
8. Send JWT token on request

Figure 2. Existing JWT Architecture and The Proposed Method.

Figure 2 portrays the comprehensive proposed method. As the JWT security admits the generated bearer token without discerning the corresponding user utilizing it for calling the API, masking the generated bearer token becomes imperative. This action entails splitting the token, encrypting each token with RSA, and concatenating them to form a new, unique token (Romero, 2021). Implementing this action enhances the security of the original token, rendering it less susceptible to unauthorized access. Since the token encompasses confidential data, including the user's JWT private key, username, and password, unauthorized users might attempt to extract sensitive information from it, thereby rendering it vulnerable. This vulnerability is particularly pronounced in step (3) of the JWT architecture. **Figure 2** depicts the existing JWT architecture and the proposed method implemented in step (4).

Table 2 illustrates the JWT bearer token before and after encryption, with several steps taken to implement the proposed method. The process commences with the generation of the token, which is then split into two parts. Each segment is encrypted with KeyPairGenerator via RSA, utilizing a 1024-bit key size (Dhamija, 2000). The splitting of the token serves to generate a new token that cannot be decrypted to its original state, even if the encrypted token is compromised. Encrypting the two parts of the bearer token makes it more difficult for unauthorized users to decrypt it to its original state (Sasaki, 2011). Once the encryption process is complete, the processed tokens are concatenated into a new, unique token and returned to the client by being placed on the response header.

The token-splitting process is initiated by creating two variables based on the token length. The token is then sub stringed from zero to the token length divided by two. The first split token is reserved in a variable designated for this purpose. Subsequently, the token is sub stringed in the range of token length divided by two to the token length, with the resulting second split token being saved in a separate variable (MacKenzie et al., 2000). **Table 3** provides a detailed representation of the first and second split token forms, which have already been encrypted.

Table 2. Existing and Proposed Method Bearer Token

Existing Bearer Token	Proposed Method Bearer Token
eyJhbGci…NbZ25rXg (token length is 206)	CSwUp…GQugAn2M= (token length is 344)

Table 3. The encrypted first and second split token

First Token	Second Token
CSwUpT0...Z6yiQAz8= (token length is 172)	E0FR0qq...QugAn2M= (token length is 172)

After splitting the tokens, both undergo the same encoding process using Base64. The resulting tokens are then concatenated to generate a unified token, which is returned to the response header (MacKenzie et al., 2000). Further details about this process can be found in **Table 4.**

3. RESULTS AND DISCUSSION

This chapter will present an analysis of the proposed method's outcomes and elucidate the application's performance tailored specifically for this research.

The constructed REST API systems and testing algorithms were evaluated using selected testing parameters, including response time (the duration required for the server to respond to a client's request), number of bits (the key size that determines the maximum number of combinations needed to break an encryption algorithm), supported browsers (the number of browsers that can support the algorithm), and authentication time (the duration required for authentication). A higher number of supported browsers indicates better results, suggesting that the algorithm possesses greater flexibility.

3.1. Response Time

The response time is evaluated to enhance the performance of the implementation stage on the server side. During the JWT production process, the username and password parameters are utilized. However, since the maximum bytes of RSA is 126 while the number of bytes of the bearer token exceeds 126, the JWT is encrypted using RSA to mask its content. The JWT is divided into two parts, each hidden with RSA to ensure its confidentiality.

A comparison was made to test the proposed method with the existing JWT method without encryption. This comparison aimed to determine the proposed method's effectiveness compared to the current method. The testing was conducted 30 times, and the results are presented in **Table 5.** The proposed method performed slightly worse regarding response time, which is logical since greater security typically requires more time.

Table 4. The Proposed Method Algorithm

```
1   Input:
2           User credential into JWT
3   Process:
4           JWT generates token
5           Create two string variables named x and y
6           Get token length for separating the token
7           Substring token range 0 to token length/2 and set the input to variable x
8           Substring token range token length/2 to token length set the input to variable y
9           Variable x and y are encrypted by RSA and encoded it using Base64
10          Combine variables x and y using the java asset function, which is "concat".
11  Output:
12          Two variables contain half a token of each encrypted token, x, and y, which have
    been concatenated
```

Table 5. Response and Authentication Time Comparison

No	Response Time (milliseconds)		Authentication Time (milliseconds)	
	Existing Method	Proposed Method	Existing Method	Proposed Method
1	151.78	227.08	168	255
2	161.79	244.6	141	206
3	141.04	254.59	146	202
4	164.72	248.65	159	200
5	155.7	221.13	145	246
6	147.81	205.44	148	226
7	159.53	253.59	141	216
8	157.88	240.52	168	195
9	139.41	229.64	137	225
10	143.74	219.53	159	236
11	159.73	222.98	144	209
12	158.59	207.57	154	245
13	143.53	213.01	166	224
14	144.98	242.1	148	211
15	166.03	208.36	161	240
16	137.58	207.07	131	226
17	168.13	245.66	139	231
18	157.18	196.89	149	195
19	139.91	226.2	155	238
20	167.82	205.28	136	231
21	150.96	232.38	151	240
22	155.91	253.06	158	246
23	168.43	237.64	169	203
24	142.14	224.6	166	254
25	155.12	226.34	134	200
26	135.02	223.86	157	243
27	141.58	208.76	138	227
28	168.94	257.4	150	232
29	135.32	211.07	147	207
30	145.37	211.17	134	214
Avg	**152.18**	**226.87**	**149.96**	**224.10**

3.2. Number of Bits

The bit count is implemented to ensure the algorithm possesses the required key size to enable the maximum number of combinations, rendering the encryption algorithm challenging to break. The RSA encryption algorithm has several key sizes, each corresponding to the number of bits in the algorithm. The maximum number of combinations to compromise an encryption technique relies on the key length. This study employs a key size of 1024 bits in the RSA algorithm. The strength of this key is moderate; therefore, to increase its potency, the bearer token is bifurcated in this research. This measure enhances the security and safety of the algorithm.

3.3. Authenticate Time

Authentication timing is performed to evaluate the efficiency of the authentication process. This testing measures the duration and process required for RSA to encrypt the generated bearer token from the JWT authentication. **Table 5** compares the authentication time between the encrypted and non-encrypted methods. The average authentication time of the encrypted method is 224.1 milliseconds, while the average authentication time of the non-encrypted method is 149.96 milliseconds.

The authentication time test results are consistent with the response time testing. Notably, the encrypted method requires slightly more time to authenticate than the existing non-encrypted method.

4. CONCLUSION

Following a series of tests, it can be concluded that the RSA256-encrypted JWT-based web application exhibits slightly worse average performance regarding response time and authentication time. However, safety-wise, the encrypted method is more secure than the non-encrypted one since it has more bits than the existing algorithm method. Implementing the proposed method by bifurcating the bearer token into two, encrypting each of the tokens, and concatenating them into one is successful.

The minimum response time of the encrypted method is 205.28 milliseconds, while that of the non-encrypted method is 135.02 milliseconds. The maximum response time for the encrypted method is 257.4 milliseconds, and for the non-encrypted method, it is 168.94 milliseconds. Notably, the average response time is 226.87 milliseconds, which is longer than that of the non-encrypted method, which only takes 152.189 milliseconds. However, since security is more important than a slightly longer response time, the proposed method's higher number of bits represents a significant security improvement.

Regarding browser compatibility, all the methods can be deployed in every browser. Concerning authentication time, the minimum authentication time for the encrypted method is 195 milliseconds, while that of the non-encrypted method is 131 milliseconds. The maximum authentication time for the encrypted method is 255 milliseconds, while that of the non-encrypted method is 169 milliseconds. The proposed method's average authentication time is 224.1 milliseconds. Although the encrypted method takes longer, the goal of achieving more security is accomplished.

The encrypted method's number of bits is the highest, which indicates that it is more secure than the non-encrypted method. The drawback is that the encrypted method's response and authentication times may be longer. However, with a higher level of security, users can tolerate the additional milliseconds the method takes.

In summary, implementing the RSA-encrypted method to JWT increases the security of JWT-based applications. Attackers will find it difficult to access the web application because they cannot decrypt the token to its original state. For future work, smaller key sizes could be implemented to reduce the execution time's consumption.

REFERENCES

Adam, S., Moedjahedy , J., & Maramis, J. (2020). RESTful Web Service Implementation on Unklab Information System Using JSON Web Token (JWT). *2020 2nd International Conference on Cybernetics and Intelligent System (ICORIS)*.

Adida, B. (2007). BeamAuth: Two-Factor Web Authentication with a Bookmark. *Conference on Computer and Communications Security*, Alexandria.

Ahmed, A., & Lee, M. (2017). Securing User Credentials in Web Browser: Review and Suggestion. *2017 IEEE Conference on Big Data and Analytics (ICBDA)*.

Akana. (2022, July 4). *What is JWT*. Retrieved from https://www.akana.com/blog/what-is-jwt

Barron, T., So, J., & Nikiforakis, N. (2021). Click This, Not That: Extending Web Authentication with Deception. *2021 ACM Asia Conference on Computer and Communications Security.* Hong Kong.

Bayardo, R., & Sorensen, J. (2005). Merkle tree authentication of HTTP responses. *14th international conference on World Wide Web.*

Boneh, D., & Franklin, M. (2001). Efficient Generation of Shared RSA Keys. *Journal of the ACM,, vol. 48, no. 4*, 702-722

Chatterjee, A., & Prinz, A. (2022). Applying Spring Security Framework with KeyCloak-Based OAuth2 to Protect Microservice Architecture APIs: A Case Study. *Sensors, vol. 22, no. 1703*.

Dhamija, R. (2000). Hash Visualization in User Authentication. *CHI 2000*, The Hague.

Gao, Y., Basney, J., & Withers, A. (2020). *SciTokens SSH: Token-based Authentication for Remote Login to Scientific Computing Environments*. https://doi.org/10.1145/3311790.3399613

Ghaly, S., & Abdullah, M. (2021). Design and Implementation of a Secured SDN System Based on Hybrid Encrypted Algorithms. *TELKOMNIKA (Telecommunication Computing Electronics and Control),*.

Idrus, S., & Zulkarnain, S. (2013). A Review on Authentication Methods. *Australian Journal of Basic and Applied Sciences, vol. 7, no. 5*, 95-107.

Kogan, D., Manohar, N., & Boneh, D. (2017). T/Key: Second-Factor Authentication From Secure Hash Chains. *CCS 2017.* Dallas.

Lee, C., Li , L., & Hwang, M. (2002). A Remote User Authentication Scheme Using Hash Functions. *ACM SIGOPS Operating Systems Review*, (pp. 23-29).

MacKenzie, P., Patel, S., & Swaminathan, R. (2000). "Password-Authenticated Key Exchange Based on RSA,". *Springer*, 599–613.

Mainanwal, V., Gupta, M., & Upadhayay, S. (2015). Zero knowledge protocol with RSA Cryptography Algorithm forAuthentication in Web Browser Login System (Z-RSA). *2015 Fifth International Conference on Communication Systems and Network Technologies.* Gwalior.

Mallik, A., Ahsan, A., Shahadat, M., & Tsou, J. (2019). Understanding Man-in-the-middle-attack through Survey of Literature. Indonesian Journal of Computing, Engineering and Design (IJoCED), 44-56.

Melton, R. (2021). Securing a Cloud-Native C2 Architecture Using SSO and JWT. *2021 IEEE Aerospace Conference (50100).* New South Wales.

Olanrewaju, R., Khan, B., & Morshidi, M. (2021). A Frictionless and Secure User Authentication in Web-Based Premium Applications. *IEEE Access vol. 9*.

Ometov, A., Bezzateev, S., Mäkitalo, N., Andreev, S., Mikkonen, T., & Koucheryavy, Y. (2018). Multi-Factor Authentication: A Survey. *Cryptography*.

Park, J. (2019). Design and Implementation of Web Browser Secure Storagefor Web Standard Authentication Based on FIDO,". *The Tenth International Symposium on Information and Communication Technology (SoICT 2019).* Hoan Kiem.

Pramono , L., & Javista, Y. (2021). Firebase Authentication Cloud Service for RESTful API Security on Employee Presence System. *International Seminar on Research of Information Technology and Intelligent Systems (ISRITI).* Yogyakarta.

Sakimura, N., Bradley, J., & Jones, M. (2022). Retrieved from JSON Web Token: https://tools.ietf.org/html/rfc7519

Sasaki, Y. (2011). Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. *International Association for Cryptologic Research*, (pp. 378–396).

Stackpath. (2022, September 20). *WHAT IS A WEB APPLICATION?* Retrieved from Stackpath: https://www.stackpath.com/edge-academy/what-is-a-web-application/

Soh, B., & Joy, A. (2003). A Novel Web Security Evaluation Model for a One-Time-Password System. *IEEE/WIC International Conference on Web Intelligence (WI'03).* Halifax.

Syamsuddin, I., Dillon, T., Chang, E., & Han, S. (2008). , "A Survey of RFID Authentication Protocols Based on Hash-Chain Method. *International Conference on Convergence and Hybrid Information Technology*.

van der Horst, T., & Seamons, K. (2007). "Simple Authentication for the Web,". *WWW 2007.* Alberta, Canada.

Venkatesha, G., Dinesh, S., & Manjunath, M. (2019). "AES Based Algorithm for Image Encryption and Decryption. *Perspectives in CommunicationEmbedded-Systems and Signal-Processing (PiCES) – An International Journal, vol. 2, no. 11*.

Wang, S., Wang, J., & Li, Y. (2013). The Web Security Password Authentication based the Single Block Hash Function. *International Conference on Electronic Engineering and Computer Science.* Yanji.

Zhang, H., & Zhu, Y. (2006). , "Self-Updating Hash Chains and Their Implementations," p. 2006. *Springer*, 387 – 397.