Managing a Secure JSON Web Token Implementation By Handling Cryptographic Key Management for JWT Signature in REST API: : A survey

Nihal Salah

Faculty of Computers & Informatics, Zagazig University, Department of Information Technology,

nihal.radwan@hotmail.com

Abstract

JSON Web Token (JWT) is a compact and self-contained mechanism, digitally authenticated and trusted, for transmitting data between various parties. They are mainly used for implementing stateless authentication mechanisms. The Open Authorization (OAuth 2.0) implementations are using JWTs for their access tokens. OAuth 2.0 and JWT are used token frameworks or standards for authorizing access to REST APIs because of their statelessness and the signature implementation. The most important cryptographic algorithms were tested namely a symmetric algorithm HS256 (HMAC with SHA-256) and an asymmetric algorithm RS256 (RSA Signature with SHA-256) used to construct JWT for signing token based on several parameters of the speed of generating tokens, the size of tokens, time data transfer tokens and security of tokens against attacks.In this research,we propose an approach used for handling cryptographic key management for signing RS256 tokens to ensure the security of the application's architecture. JWT offer a variety of options to manage keys, the server always needs to verify the validity of the key before trusting it for verify that a JWT implementation is secure.The experimental results show It's better to use the RS256 signature method for handling cryptographic key management for signing tokens to manage a secure JWT Implementation.

Keywords: Authorization, JWT, Security, Cryptographic key management

1.Introduction

Every the Representational State Transfer (REST) [1] Application Programming Interface (API) request from the client to a server must contain all the necessary information necessary to serve the request. The server maintains neither state nor context. For authorize access to protected REST APIs.

OAuth 2.0 [2] and JWT [3] are two of the most widely used token frameworks or standards for authorizing access to REST APIs.

Received: January 24, 2021 Revised: March 24, 2020 Accepted: April 11, 2020

DOI: 10.5281/zenodo.3909958

The most important cryptographic algorithms were tested used to construct JWT for signing token as done based on the speed of generating tokens, the size of tokens, time data transfer tokens and security of tokens against attacks. The experimental results showed the use of The HS256 has performance than RS256 for framework of evaluation the performance of proposed algorithms.

Token-based authentication enables the credentials of the user's name and password to get tokens that allow them to access certain resources. After a token is available, users can reuse the same token to access resources without using a username and password again. The given token has been made by the server with a certain algorithm.

1.1Json web token

Json web token [4] is an open standard that defines a compact and self-contained way for securely transmitting information between parties. It is an authentication protocol where we allow encoded claims (tokens) to be transferred between two parties (client and server) and the token is issued upon the identification of a client, with each subsequent request we send the token.

JWT tokens are based on JSON and used in new authentication and authorization protocols like OAuth 2.0. JWT tokens can be signed using a secret (HMAC algorithm) or public / private key pair (RSA algorithm). Therefore, the focus of this research to handle key management for JWTs.

1.1.1Json web key set (JWKS) endpoints

In this scenario, the private key represents the secret and should never be shared with anyone. The second component, the public key, can be available to everyone.This comes practical because it allows to share the public keys without security concerns.Providing a JWKS endpoint [5] to serve application's public key which then can be used to verify JWTs.

JWK specification to represent the cryptographic keys used for signing RS256 tokens. This specification defines two high level data structures: JSON Web Key (JWK) and JSON Web Key Set (JWKS) [6] can be seen in Table 1.

Table 1. JSON Web Key (JWK) and JSON Web Key Set (JWKS).

Item	**Description**
JSON Web Key (JWK)	A JSON object that represents a cryptographic key. The members of the object represent properties of the key, including its value.
JSON Web Key Set (JWKS)	A JSON object that represents a set of JWKs. The JSON object must have a keys member, which is an array of JWKs.

1.1.2JWKS structure

Received: January 24, 2021 Revised: March 24, 2020 Accepted: April 11, 2020
DOI: 10.5281/zenodo.3909958

Describes the JKWS key elements for each key type, such as "RSA" depending on the key type. Below is more information about the common parameters for different key families inside a JWK Set document can be seen in Table 2.

Table 2. Parameters for different key families inside a JWK [7].

Name	**Type**	**Description**
keys [ ]	String	JWK values. One or more.
Kty	String	Identifies the cryptographic algorithm family used with the key; for example, RSA or EC. A case-sensitive string.
Use	String	Public key use: identifies the intended use of the public key This parameter indicates whether a public key is used for encrypting data or verifying the signature on data. Possible values: sig (signature) enc (encryption)
Alg	String	Identifies the algorithm intended for use with the key. A case-sensitive ASCII string.
Kid	String	Key ID: used to match a specific key. This is used, for instance, to choose among a set of keys within a JWK Set during key rollover. Distinct "kid" values are used within a JWK Set when representing different keys.

JWKS structure:

{ "keys":

[{

"kty":"RSA","e":"AQAB","use":"sig","kid":"key-id","alg":"RS256","n":"kCv9hq2XD6upQBylkA2KCdkY1tgpgqnvHR0jAmgTskYme9O4JSrZiSDgQ5wJdSS2osi--UCD_ErQ3moEs8OyGX-Ks8xbeZ9uL7Rfy2McJ0gZ-1CnlOU9JdoyEZb5qKCcazSeM8Dl6K5er5IZ7-g0XnhyalbdRw74lX1eCAkPGvEMaxFn5fRwVJElOymsk-G61Xqp070pwrygSBNPYw4CrhGIlpd5if6nfZM9QVXzVUgmFa4BCho0b4mpe8s0W4ctNlXVB_iQp020KPa8Ik2N7mAS3u3OJrapNRDo5X56vnlNMM-oeRWGayNIBfnVJca9QkCzAA5rUP7nRzssUOE0_w"

}]

2.Related work

The use of token-based authentication using JWT has already been investigated [8] which focused on the security and privacy challenges of cloud computing with specific reference to user authentication and access management for cloud SaaS applications. The suggested model uses a framework that harnesses the stateless and secure nature of JWT for client authentication and session management.

Received: January 24, 2021 Revised: March 24, 2020 Accepted: April 11, 2020
DOI: 10.5281/zenodo.3909958

The work [9] proposed an approach that combines OAuth 2.0 with JWT in REST API and test the algorithm performance on JWT and compare the performance of the two HS256 (HMAC with SHA-256) and RS256 (RSA Signature with SHA-256) algorithms in the parameters of the speed of generating tokens, the size of tokens, time data transfer tokens and security of tokens against attacks.

3.Background

This section describes some issues and solves all these problems that if an attacker compromises a user's token that does not expire, the attacker will have a valid token until signing keys aren't rotated. with a short-lived token, an attacker would have minimal time to access resources granted by the token. If token refresh process is costly, then we can extend the lifespan of tokens to reduce the number of refresh requests. If refresh process does not consume a lot of resources, then we can safely decrease the time to live. There should be a balance between resources and control. By increasing a token's lifespan, we reduce control over the token, and vice versa.

Key rotation is the process of phasing out old keys, and using new ones. It is a good idea to rotate keys anyone with keys has the ability to impersonate the service using them and to keep environment secure and maintain the security status of applications that proposes an approach used for handling cryptographic key management for signing tokens by RS256 signatures.

3.1Cryptographic key management for signing tokens by RS256 signatures

In RS256 signatures the public key used to validate tokens can be published anywhere at the same time would be problematic, an attacker wants to be able to forge JWTs, not validate them.
The application or resource servers then only need to connect to that server to fetch the public key, and check again periodically just in case it has changed, either due to an emergency or periodic key rotation. So, there is no need to bring the application server and the authentication server down at the same time, and update the keys everywhere in one go.

One of the most challenging parts of using JWT is handling the cryptographic key material. Cryptographic keys used for signing and encryption need to be rotated frequently. Performing too many operations with a key opens the application up to cryptanalysis attacks. Rotating keys implies that multiple keys might be in use at the same time. Additionally, it requires the issuer and consumer to retrieve keys dynamically.

4. Methodology

This section proposes scheme instead of installing the public key on the resource server, it's much better to have the authorization server publish the JWT validating public key in a publicly accessible URL.
The methodology used in handle key management is as follows when JWT is obtained from an issuer, often the issuer inserts a Key ID (or kid) into the JWT header. The key tells the recipient of the JWT how to find the public or secret key necessary to verify the signature on the signed JWT.

In this approach, we suppose an issuer signs a JWT with a private key. The "Key ID" identifies the matching public key to use to verify the JWT. The list of public keys is typically available at some well-known endpoint. This is the basic sequence (platform that works with JWKS) as shown Figure 1 needs to perform to work with a JWS/JWT that has a JWKS:

1. Client request an access token from the authorization server.

Received: January 24, 2021 Revised: March 24, 2020 Accepted: April 11, 2020
DOI: 10.5281/zenodo.3909958

2. The authorization server response an access token JWT to client.
3. Client makes a request with JWT for access resources.
4. The resource server requests public key from authorization server.
5. Discovery endpoint of authorization server returns information about the server's configuration in JSON format. The value of the jwks_uri parameter in the JSON represents the URL of the JWK.
6. Get the URL of the JWK set document.
7. Request to JWK set that the authorization server provides an endpoint that exposes its JWK Set endpoint that includes a public key whereby to verify signature of access tokens, resource servers can download the public key from the endpoint.
8. Return JWK set that contains public keys.
9. For Getting public key to use to verify the JWT to perform to work with requested JWT that has a JWKS:

    9.1 Examine the JWS/JWT header to find the Key ID (kid).
    9.2 Examine the JWS/JWT header to find the signing algorithm (alg), such as RS256.
    9.3 Retrieve the list of keys and IDs from the JWKS of the JWK set endpoint for a given issuer.
    9.4 Extract the public key from the list of keys with the key ID noted in the JWS/JWT header and with the matching algorithm, if the JWKS key specifies the algorithm.
    9.5 Use that public key to verify the signature on the JWS/JWT.

10. And finally, resource server retrieves the resource, now being sure that the client has the correct permissions.

Received: January 24, 2021 Revised: March 24, 2020 Accepted: April 11, 2020
DOI: 10.5281/zenodo.3909958

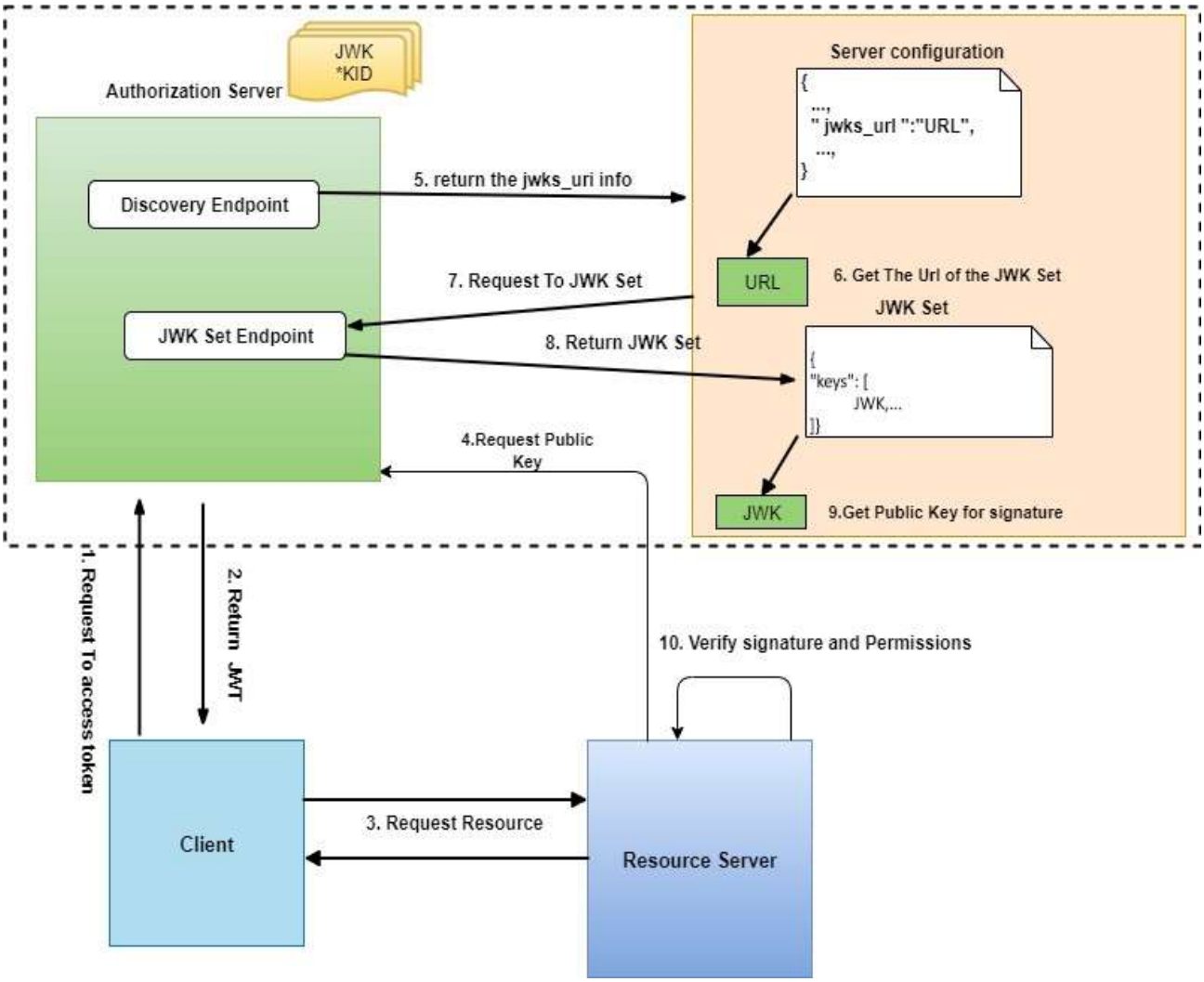

Figure 1. Platform that works with JWKS

The verify JWT policy does everything else:

If a key with a Key ID that matches the Key ID (kid) asserted in the JWT is not found in the JWKS, then the Verify JWT policy throws an error and does not validate the JWT.

If the inbound JWT does not bear a key ID (kid) in the header, then this mapping of keyid-to-verification-key is not possible.

5. Experimental results and discussion

Proposed scheme is done by making OAuth2.0 with JWT. It has been developed in JAVA and configured with Spring Security OAuth2 [10] by implementing the algorithm RSA-256 to prevent the security issues that JWT offer a variety of options to manage keys. One way is to identify a particular key using an identifier to enable the embedding of a key in the token, or the retrieval of a key from a dedicated key URL.

5.1 Evaluation of cryptographic key management for signing RS256 tokens

Received: January 24, 2021 Revised: March 24, 2020 Accepted: April 11, 2020

DOI: 10.5281/zenodo.3909958

The evaluation of cryptographic key management is performed show that JWK set URL also allows us to implement "key rotation "by expiring RSA key pairs and issuing new ones when needed. This is an additional security layer, as it would make harder to generate fake JWT using a stolen private key, since they are frequently rotated:

1. Store key in a dedicated key service.
2. Use the kid claim in the header to identify a specific key.
3. The header can also contain a URL pointing to public keys.

We have rotated without causing any issues for users and store keys securely and give out access to them.

5. Conclusions

In this research, we proposed an approach used for handling cryptographic key management for signing tokens by RS256 signatures for managing a secure JWT Implementation to ensure the security of the application's architecture. JWT offer a variety of options to manage keys, the server always needs to verify the validity of the key before trusting it for verify that a JWT implementation is secure.The experimental results show It's better to use the RS256 signature method for handling cryptographic key management for signing tokens to manage a secure JWT Implementation.

References

1. Ong, S.P., et al., *The Materials Application Programming Interface (API): A simple, flexible and efficient API for materials data based on REpresentational State Transfer (REST) principles.* Computational Materials Science, 2015. **97**: p. 209-215.
2. Hardt, D., *The OAuth 2.0 authorization framework*. 2012, RFC 6749, October.
3. Jones, M., B. Campbell, and C. Mortimore, *JSON Web Token (JWT) profile for OAuth 2.0 client authentication and authorization Grants.* May-2015.[Online]. Available: https://tools. ietf. org/html/rfc7523, 2015.
4. Peyrott, S.E., *The JWT Handbook*. 2017.
5. Jones, M. *JSON Web Key (JWK)*. May 2015; Available from: https://tools.ietf.org/html/rfc7517.
6. auth0. *JSON Web Key Set*. Available from: https://auth0.com/docs/jwks.
7. Peyrott, S., *The JWT Handbook.* Seattle, WA, United States, 2016.
8. Ethelbert, O., et al. *A JSON token-based authentication and access management schema for Cloud SaaS applications*. in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. 2017. IEEE.
9. Ehab rushdy, W.K., Nihal salah, *Framework to secure the oauth 2.0 and json web token for rest api.* Journal of Theoretical and Applied Information Technology -- Vol. 99. No. 09 -- 2021
10. Alex, B., et al., *Spring Security Reference.* URL https://docs. spring. io/spring-security/site/docs/current/reference/htmlsingle/.[utoljára megtekintve: 2017. 04. 21.], 2004.

Received: January 24, 2021 Revised: March 24, 2020 Accepted: April 11, 2020

DOI: 10.5281/zenodo.3909958