

JWT Private / Public Key Confusion

Asked 4 years, 9 months ago Modified 3 months ago

Viewed 137k times



74



I'm trying to understand the logic of using JSON web tokens with private/public keys (RS512) when signing a payload of data sent from a client (in this case, a React Native App) to my server.

I thought the whole point of private/public keys was to keep the **private** key private (on my server) and hand the **public** key to the person who's successfully logged into the app.

I thought, with each API request to my server, the authenticated user of the app would use the **public** key to create the JWT (on the client side) and the server would use the **private** key to verify the signature/payload from the API request.

It seems I have it backwards because everywhere I read, you're supposed to sign a JWT with the **private** key -- but that goes against my understanding of who's in possession of the keys.

Depending on how the keys are created, some private keys can have a passcode which is supposed to be secret! So if the private key and the secret is out in the open (in client-side code) how secure can that be?

And where does the encryption come in? If the user of the app is sending sensitive data in the API, am I supposed to encrypt the payload and sign it with the JWT on the client side and then let the server verify the JWT signature and decrypt the data?

This tutorial was helpful

<https://medium.com/@siddharthac6/json-web-token-jwt-the-right-way-of-implementing-with-node-js-65b8915d550e> but it seems backwards.

Any explanation would definitely help because all of the on-line tutorials aren't making sense.

Thank you.

jwt

Share

Improve this question

Follow

asked Mar 5, 2020 at 4:06



Marc

1,730 ● 2 ● 17 ● 25


-
- 16** JWTs are usually NOT used for encrypting/decrypting of payload. They are solemnly used for authentication and authorization. The website you are linking to is describing the process correctly. A user logs in at an authorization server and receives a JWT token which is signed with the **private** key of the authorization server. The user then attaches this very JWT to every sent request. The authorization server then verifies whether the JWT was issued by this authorization server and if it was tampered by someone by

verifying its signature with the **public** key. – [pero_hero](#) Mar 5, 2020 at 5:38

7 I think you are confused by two different terms; signing and encryption. The data is encrypted using the public key and decrypted using the private key. On the other hand, you use your private key to sign your data and use your public key to verify it. – [dubucha](#) Mar 20, 2021 at 4:18

1 ok but why does the [jwt.io](#) contain the private key also besides the public one in the 3rd part of the JWT token (aka the signature)? – [Lucian](#) Dec 28, 2021 at 20:22

Agreed, in my second example the JWT private key should not be used. This solution isn't acceptable. I was trying to create a way of refreshing the key with every API call but it's not ideal. I'm going to delete this part of the answer. – [Marc](#) Dec 29, 2021 at 22:07

@Lucian [jwt.io](#) is a tool to inspect, verify and create tokens. I guess with *contain the private key also besides the public one in the 3rd part of the JWT* you refer to the input fields in the right column. You can insert the private key there to sign a token. When you have an existing token on the left side, you just insert the public key on the right side to verify the token, but if you add a private key, the token is signed using that private key. – [jps](#) Dec 31, 2021 at 9:06 

5 Answers

Sorted by:

Highest score (default)



83

With JWT, the possession and the use of the key materials are exactly the same as in any other contexts where cipher operations occur.



For signing:



- The private key is owned by the issuer and is used to compute the signature.
- The public key can be shared with all parties that need to verify the signature.

For encryption:

- The private key is owned by the recipient and is used to decrypt the data.
- The public key can be shared with any party that wants to send sensitive data to the recipient.

Encryption is rarely used with JWT. Most of the time the HTTPS layer is sufficient and the token itself only contains a few information that are not sensitive (datetime, IDs...).

The issuer of the token (the authentication server) has a private key to generate signed tokens (JWS). These tokens are sent to the clients (an API server, a web/native application...). The clients can verify the token with the public key. The key is usually fetched using a public URI.

If you have sensitive data that shall not be disclosed to a third party (phone numbers, personal address...), then the encrypted tokens (JWE) are highly recommended. In this case, each client (i.e. recipient of a token) shall have a private key and the issuer of the token must encrypt the token using the public key of each recipient. This means that the issuer of the token can select the appropriate key for a given client.

Share Improve this answer

edited May 19 at 22:43

Follow



Tuychiev Toir

370 ● 2 ● 12

answered Mar 5, 2020 at 7:36

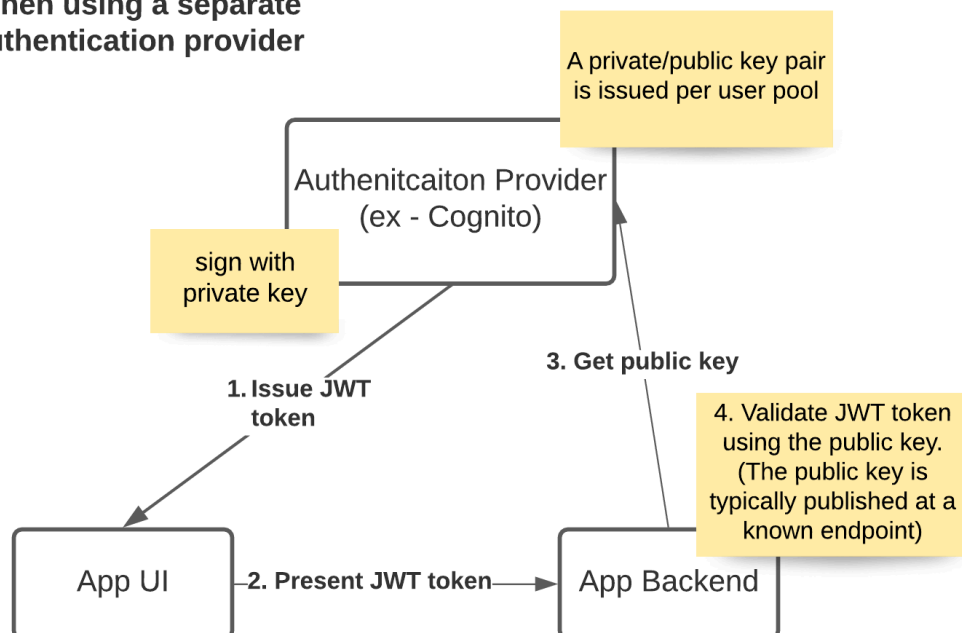


Spomky-Labs

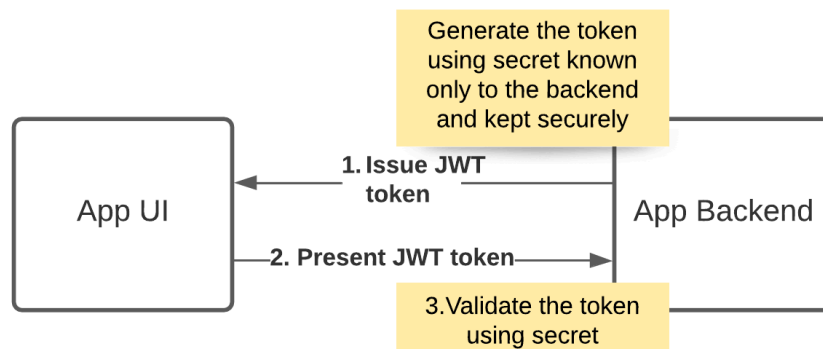
16.7k ● 5 ● 44 ● 68

Hope this figure adds some clarity.

When using a separate authentication provider



When your backend serves as auth provider



Share Improve this answer

answered Mar 12, 2022 at 15:34

Follow



Sashi

2,395 ● 20 ● 15

7 picture speaks a thousand words! – Sachin Giri Aug 14, 2022 at 19:11

How do you get the public key in a client app backend? I'm using .net core and I need to fetch public key (in builder configuration stage) from my authentication server which is different from my client app – sTx Feb 6, 2023 at 21:15

The public keys are typically exposed at a known endpoint. For a Cognito user pool, the public endpoint looks like this - "[cognito-idp.{region}.amazonaws.com/{userPoolId}/.well-known/jwks.json](#)" Here is a link with more information - [aws.amazon.com/premiumsupport/knowledge-center/...](#) – Sashi Feb 23, 2023 at 12:00



10



Solution for JWE in React Native and Node Backend

The hardest part was finding a method that works in both RN and Node because I can't just use any Node library in RN.

I'm transmitting all of the API calls over HTTPS.

Create a JWE to encrypt the token and payload simultaneously.

React Native App Code

```

import {JWK, JWE} from 'react-native-jose';

/**
 * Create JWE encrypted web token
 *
 * @param payload
 * @returns {Promise<string>}
 */
async function createJWEToken(payload = {}) {

  // This is the Public Key created at login. It
  // is stored in the App.
  // I'm hard-coding the key here just for
  // convenience but normally it
  // would be kept in a Keychain, a flat file on
  // the mobile device, or
  // in React state to refer to before making the
  // API call.

  const publicKey = `-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEApI9FLYsL
qdAeHA8qDU5rmY8YFFlc0cy2q1dijpgfop8WyHu1ULufJJXm0PV2
DZ+/qTv4gldJjyIlo/PIhehQJqSrdIim4fjuwkax9F0CuFQ9nesv
QSxUPjNzsYG0uULWSR3cI8FuV9InlSZQ7q6dEunLPRf/rZujxiAx
LNdl7qDE0sc109Yy3HBB0wUdJyyTg/GRPwklLogw9kkldz5+wMvw
qJpqqqt1KmxdoQNbeGwNzZiGiuYIdiQWjilq5a5K9e75z+Uivx+G3
LwIDAQAB
-----END PUBLIC KEY-----`;

  try {

    const makeKey = pem => JWK.asKey(pem, 'pem');
    const key = await makeKey(publicKey);

    // This returns the encrypted JWE string

    return await JWE.createEncrypt({
      zip: true
    }, payload, key);
  } catch (error) {
    console.log(error);
  }
}

```

Node Backend

```

const keygen = require('generate-rsa-keypair');
const {JWK, JWE} = require('node-jose');

/**
 * Create private/public keys for JWE
 * encrypt/decrypt
 *
 * @returns {Promise<object>}
 *
 */
async function createKeys() {

    // When user logs in, create a standard RSA key-
    // pair.
    // The public key is returned to the user when
    // he logs in.
    // The private key stays on the server to
    // decrypt the message with each API call.
    // Keys are destroyed when the user logs out.

    const keys = keygen();
    const publicKey = keys.public;
    const privateKey = keys.private;

    return {
        publicKey,
        privateKey
    };
}

/**
 * Decrypt JWE Web Token
 *
 * @param input
 * @returns {Promise<object>}
 */
async function decryptJWEToken(input) {

```

Share Improve this answer

edited Dec 29, 2021 at 22:11

Follow

answered Mar 5, 2020 at 19:52



Marc

1,730 ● 2 ● 17 ● 25



1



jwt.io does a great job of explaining that there is more than one way to sign the JWT. Users may sign and verify with a single secret, or use a public/private key pair for verifying/signing respectively.

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on signed tokens. Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

Share Improve this answer

answered Jan 28, 2022 at 13:53

Follow



MichaelG

712 ● 1 ● 10 ● 18



0



The whole point of a **bearer** token, such as how JWTs are commonly used, is their intrinsic validity, i.e. you can verify them without being the issuer. This makes sense in a microservice architecture, where you have an auth server and many services that can accept the user token without asking the auth server. So, in the end, both the private and the public key are in the server, just in different machines. No one else is gonna verify a token in real life, but a company branch can set up a service from remote with just the public key received via email

Share Improve this answer

answered Sep 19 at 8:32

Follow



Cesare Giannetti

1 ● 1