

Zero-Knowledge Authentication & Identification protocol

Augustin Poirier

November 28, 2025

Zero-knowledge end-to-end Authentication & Identification (ZKAI) is a protocol designed to enforce significantly the confidentiality of users by reducing, at the very least, the amount of time any information is available in plain-text. By applying this method, we can keep the information encrypted until its final use by the website's backend logic. The data is never stored in plain-text in the database or in the user's browser.

Objectives :

- Never store the data in plain-text in the database
- Never store the data in plain-text in the user's browser
- Avoid significantly impacting the user's experience

Registration process :

In order to complete the registration process, the user must generate their own unique cryptographic keys (a pair consisting of a Private Key and its associated Public key).

Once the keys have been generated, the user needs to encrypt all its required informations (for example : a name, an email address, etc) using their Private Key. The user then creates a dataset, containing this encrypted information with each value provided by the user encrypted individually.

This entire process (key generation and dataset creation) is often automated by a dedicated application. The user can then submit their encrypted informations, along with their Public Key, to the web application to complete the registration process.

Exemple of dataset :

```
register {
    name : encrypt(name, Public Key),
    email : encrypt(email, Public Key),
    Public Key : Public Key
}
```

The backend will store this information in the database and return to the user an ID corresponding to his account.

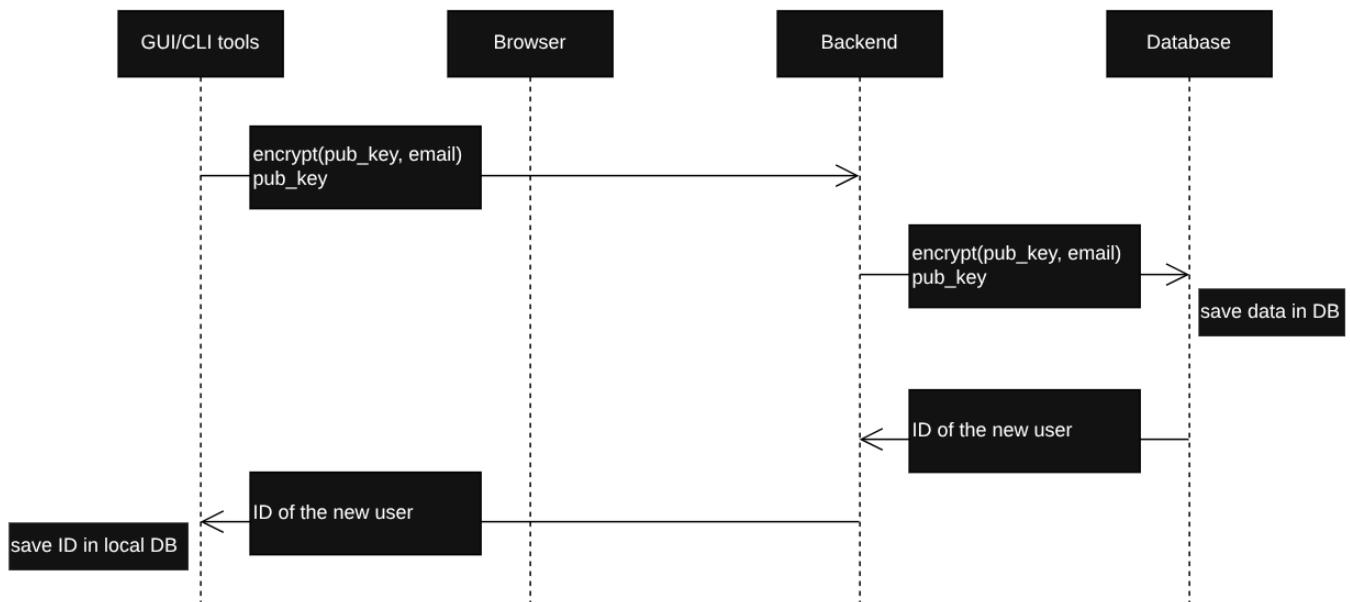


Figure 1: Registration process

Authentication process :

The user sends their ID to the website's backend, which answers with a crypto-challenge encrypted using the Public Key registered for that ID. The

user use its Private Key to solve the crypto-challenge, proving its identity. He then sends back this decrypted cryptochallenge as proof that they have the corresponding Private Key, along with a JWT. This JWT contains user informations, encrypted with their Private Key. The JWT is still unsigned at this point.

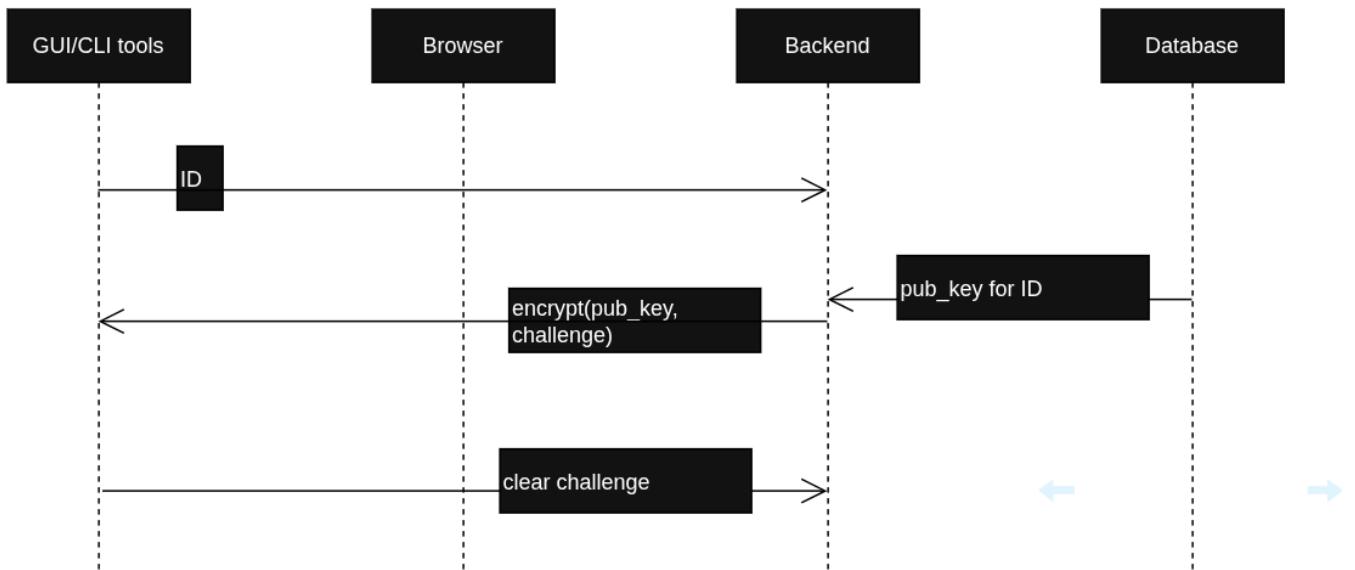


Figure 2: Authentication process

Identification process :

The backend retrieves from the database the public key associated with this user (using their ID), and uses it to decrypt the JWT submitted by the user (whose values were encrypted with the user's Private Key). It then re-encrypts the decrypted JWT using the user's Public Key. The backend compares this new encrypted value with the reference store in the database, to verify that the submitted JWT is authentic.

If the verification succeeds, it means that the user has not forged or tampered with the JWT. The backend then signs the submitted JWT (which still contains the values encrypted with user's Private Key) , to produce a JWTS. This JWTS is then stored in the user's cookies.

Thus the only unencrypted data of the JWTS is the user's account ID. It needs to be in plain-text for the backend to find the user in its database.

The data is never stored in plain-text except when used. The database is now a simple support to verify that the data provided by the user during his connections are correct, containing only encrypted values. It is no longer a storage medium of valuable informations for an attacker.

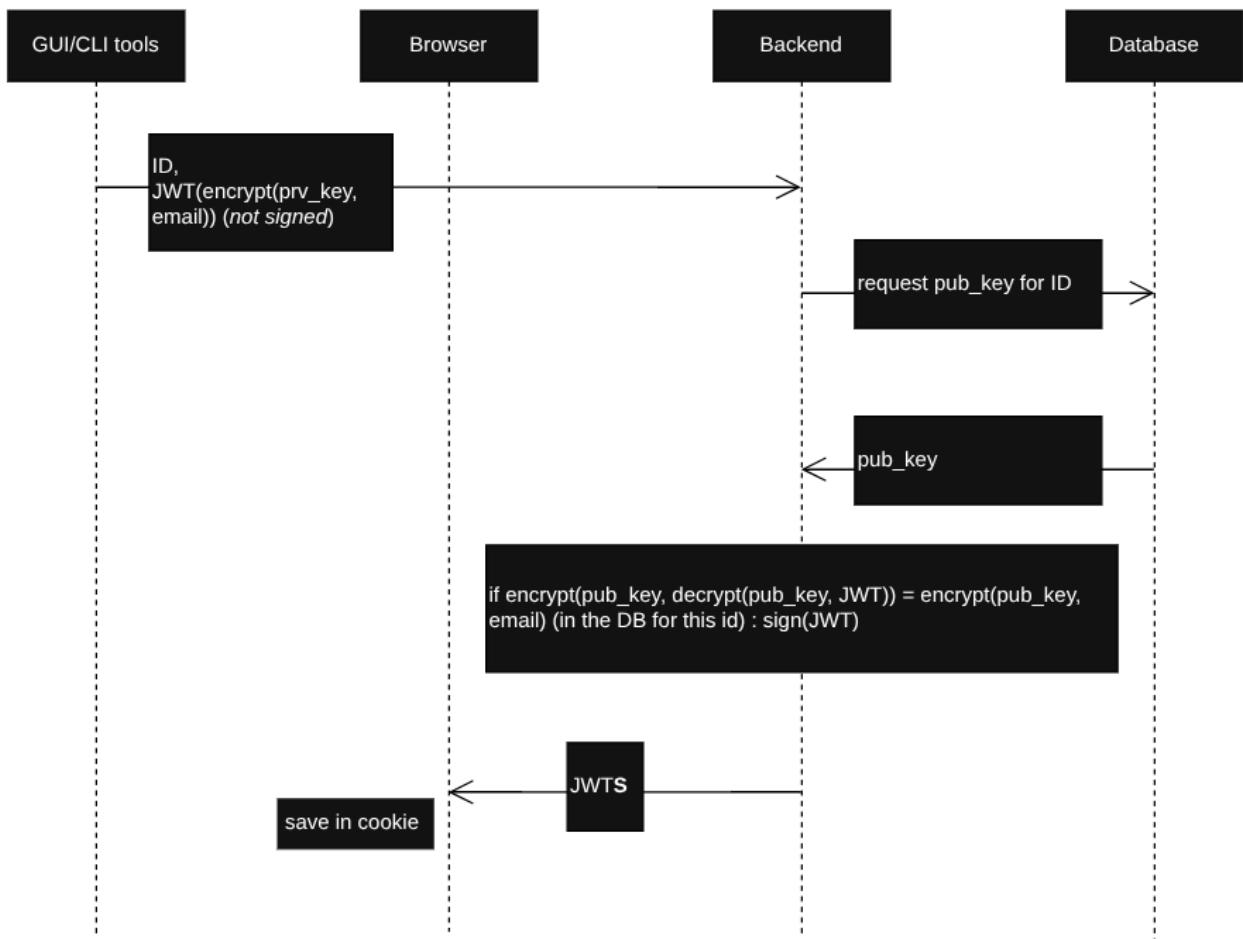


Figure 3: Identification process

Use data from the JWTS :

To retrieve the plain-text data from the JWTS when necessary, the backend can at any point request the public key for the user ID from the database and decrypt the information it needs from the JWTS stored in the browser..

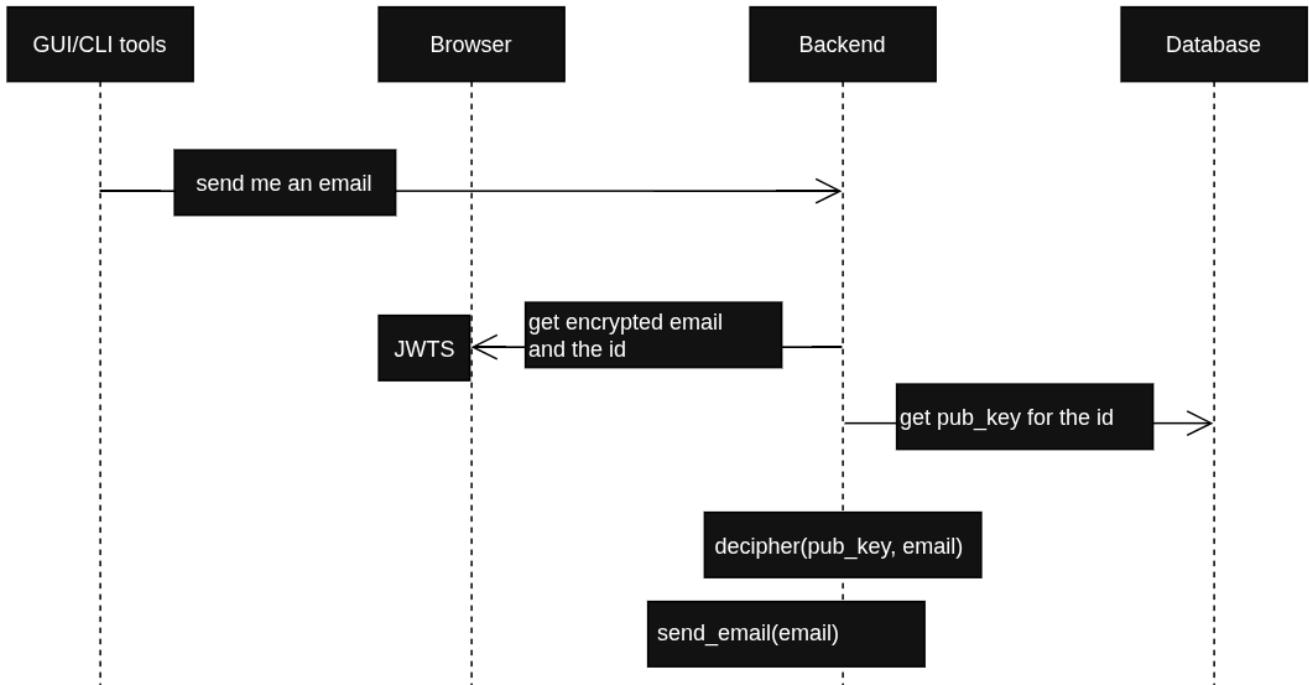


Figure 4: Use data from JWT

Conclusion :

Although the JWT data is always encrypted, it can still be used by an attacker for session hijacking. If they manage to steal the user's cookie (for example, if using unencrypted HTTP, and before the cookie's TTL expires), they can make their own requests to the web application, providing the stolen cookie, thus spoofing the user's identity.

However, the attacker will not be able to extract the data from inside the JWT, since it's been encrypted with the user's Private Key. This can help mitigate certain types of data leaks ; for example in cases where a front-end vulnerability sends users' cookies to an external server when they visit a compromised page.

This JWTS theft scenario may affect the access control of the application but it does not compromise data confidentiality as would occur in a more traditional stateless identification process.

Thanks to it4, without whom this document would have been unreadable.