

Authentication systems in IT solutions based on web architectures

Luca Di Bello, Computer Science, SUPSI, Lugano, Switzerland

luca.dibello@supsi.student.ch

1. Abstract

The web over the years has spread to all areas of our lives, from the most basic to the most complex thus being more and more connected to the Internet. This led to the necessity of having a secure and reliable authentication system to protect users and the system itself. In this article will be presented the most common authentication techniques used in web applications and the advantages and disadvantages of each one, followed by real-world study cases. The main goal is to present the most common authentication systems and their respective characteristics, so that the reader can have a better understanding of the subject and be able to choose the best authentication system for their project.

Keywords: *json web tokens; oauth2; saml*

2. Introduction

Authentication systems are one of the most important parts of a web application. This system is responsible for identifying the user and granting access to the system. This process comprises a series of steps that are performed in order to verify the identity of the user. It is usually divided into two phases: the first phase is the authentication itself, and the second phase is the client authorization.

Authentication is the process of verifying the identity of a user. It is the first step of the authorization process, which is the process of verifying that the user has the rights to perform the action they are trying to perform. The authentication process is performed by a system or service, which in the case of web applications is the web server. A user can authenticate on a web server by entering a username and password, which is the most common method, but there are other methods that will be presented later in this article. The web server receives the username and password and then verifies them, and if they are correct, the user is authenticated and the server returns an authentication token to the client.

The authentication process is usually performed by the user, but in some cases, the system can perform the authentication process automatically, for example, when the user is already logged in.

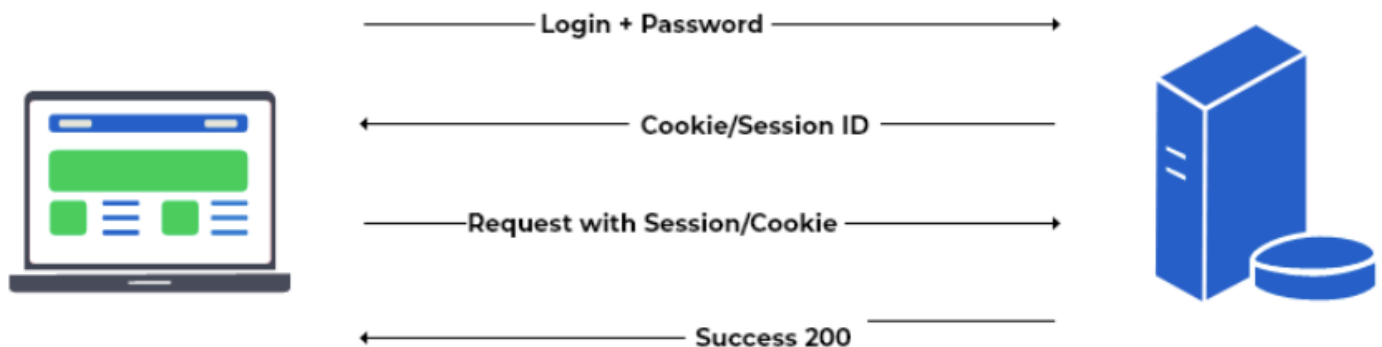
Authorization is the process of verifying that the user has the rights to perform the action they are trying to perform. The authorization process is performed by a system or service, which in the case of web applications is the web server. The web server receives an authenticated session signature and verifies if the user has the rights to perform the action they are trying to perform. Usually, if the user is authorized to perform a certain action, the server returns the operation result to the client, and in the other hand, if the user is not authorized, the server returns an error to the client and the operation is not performed.

3. Authentication systems in web applications

3.1. Different kinds of authentication systems

Authentication systems for web-based IT solutions can be categorized into two main groups: session-based and token-based.

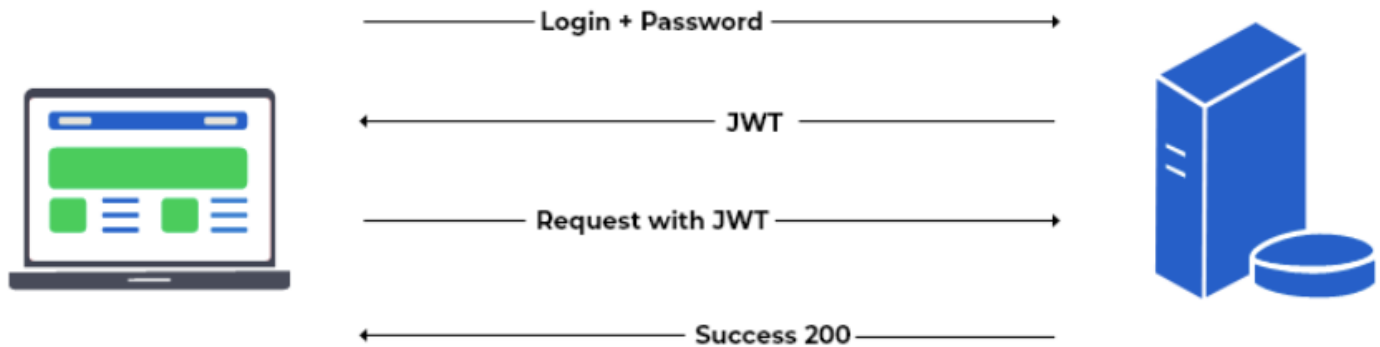
Session-based authentication is a method of stateful-authentication [1] that uses a session identifier to identify the user. After a successful user authentication, the server creates a session and shares its unique identifier (referred as *session ID*) with the related user. This identifier is later stored in the client-side (usually through the use of cookies) and used to access protected API routes. The two main drawbacks of this method are that the server must store the session data in memory, which can lead to scalability issues and, since the session ID is stored in the client-side, is prone to XSS (*Cross-site scripting*) and CSRF (*Cross-site request forgery, also known as one-click attack*) attacks.



Source: <https://beaglesecurity.com/blog/article/session-security.html>

Token-based authentication is a method of stateless-authentication [2] that uses a token to identify the user. After the user logs in, the server generates a token that is signed with a secret key or a key

pair and returns it to the user. The user must then send this token in the header of every request to access protected API routes.



Source: <https://beaglesecurity.com/blog/article/session-security.html>

[1]: Stateful Authentication is a way to verify users by having the server or backend store much of the session information, such as user properties.

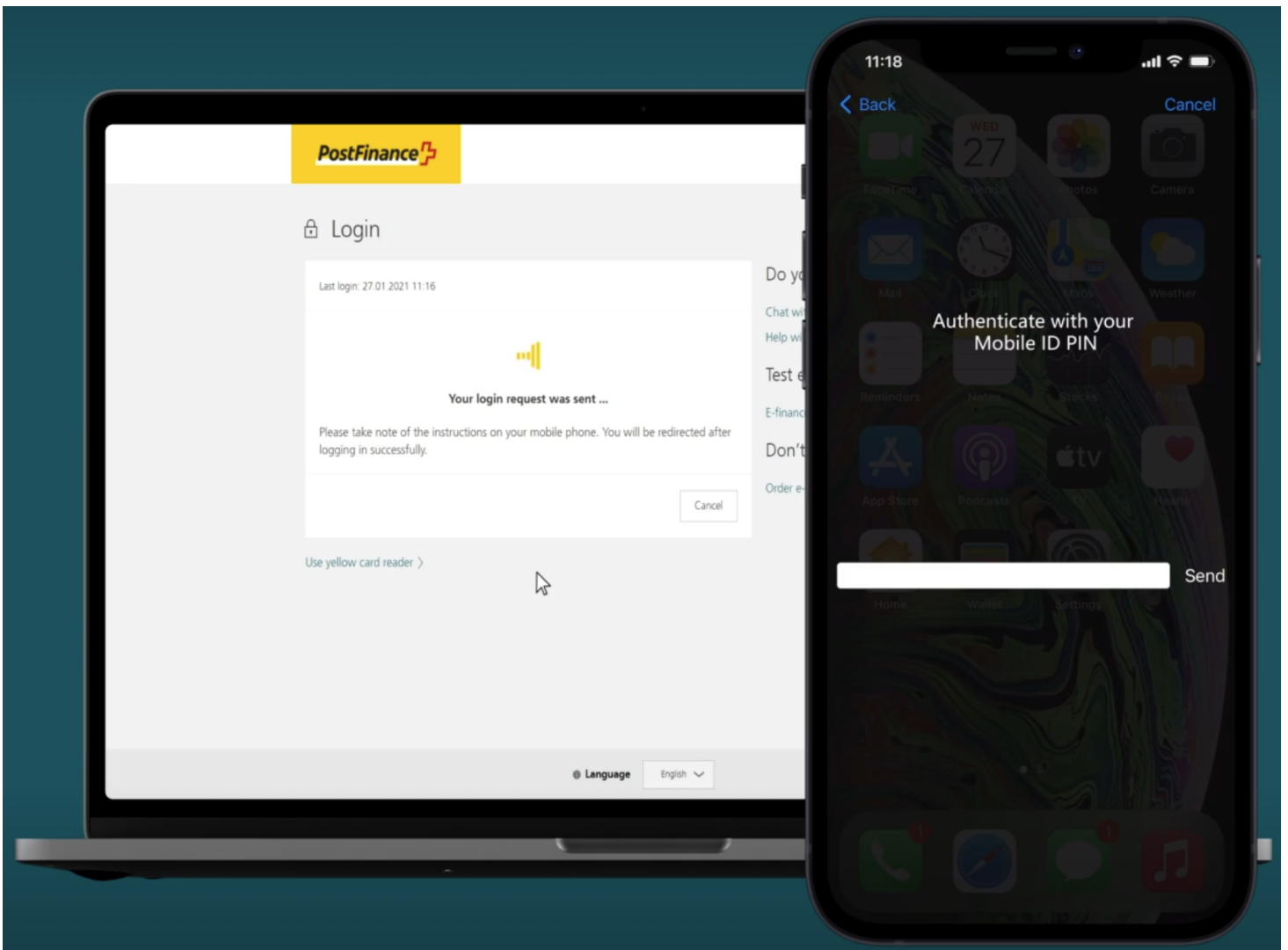
[2]: It is commonly referred to as stateless authentication since the token can be a self-contained entity that transmits all the necessary information for authenticating the request.

3.2. Study cases

3.2.1. PostFinance - Mobile ID

PostFinance is a Swiss bank that offers a wide range of financial services, including banking, insurance, and asset management. It is a subsidiary of the Swiss Post, which is the Swiss postal service.

User can access the online banking service using the PostFinance Mobile App (which is available for both Android and iOS) or using the web portal. If Mobile ID two-factor authentication is enabled, after the user enters their username and password they will be asked to confirm the login attempt using the Mobile ID. On the mobile app, the user will be asked to confirm the login attempt by entering the MobileID PIN code set during two-factor authentication setup. If the PIN code is correct, the user is authenticated and can access the online banking service.



Source: <https://www.postfinance.ch/en/private/products/digital-banking/mobile-id.html>

Mobile ID in depth

Mobile ID is a managed authentication service from Swisscom (Swiss telecommunication company) based on a secure hardware tokens. The token can be either the user Mobile ID compliant SIM/eSIM or the mobile device running the Mobile ID app (available on both iOS and Android, used when SIM/eSIM not supported). An account could have either the eSIM/SIM method, the MobileID app method, or both at the same time.

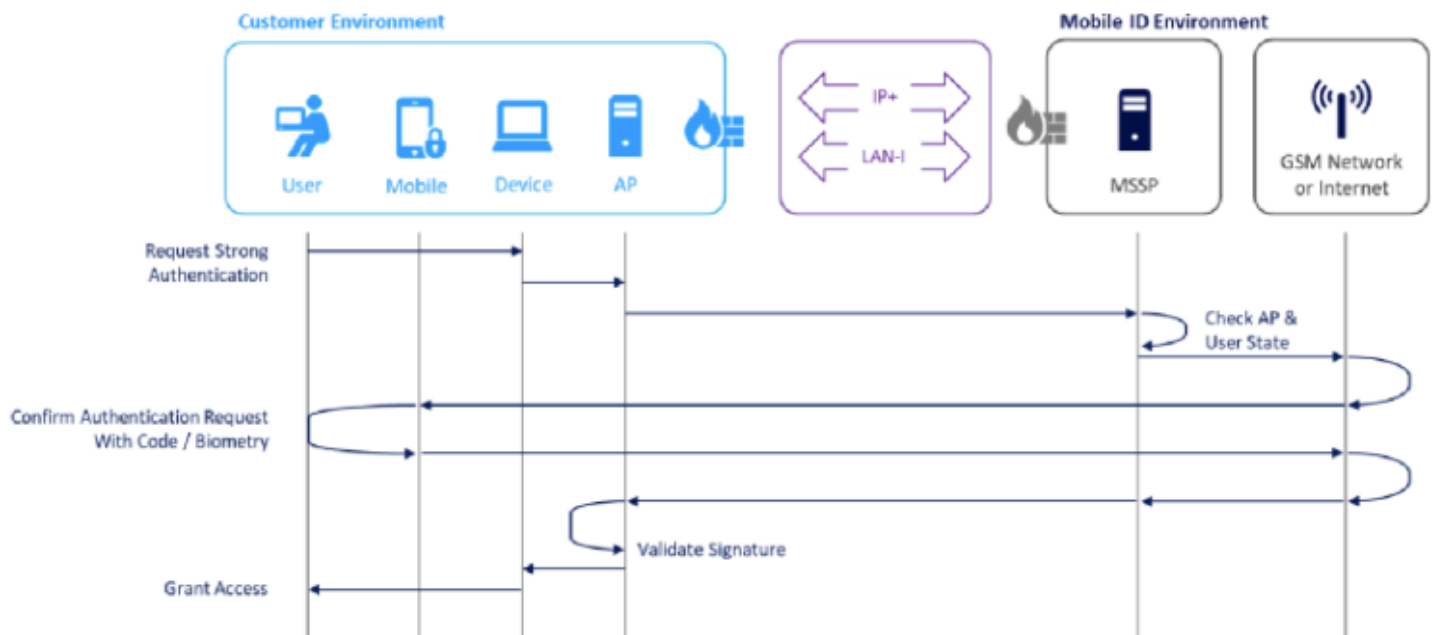
This is an example of Mobile ID authentication flow:

Mobile ID (e-)SIM: An application provider (in this study case *PostFinance*) can request SIM Toolkit (STK) based authentication. To use this method, the user must have a Mobile ID compliant SIM/eSIM.

Data exchange between the Mobile ID server and the STK application is done with SMS messages using data packets which are not visible to the user. Since the Mobile ID SIM Toolkit runs on SIM card

environment, the Mobile ID SIM/eSIM authentication availability does not depend on the mobile device.

This is an example of a Mobile ID SIM/eSIM authentication:



Source <https://github.com/MobileID-Strong-Authentication/mobileid-api/blob/main/doc/mobile-id-reference-guide-v-3-9.pdf>

The Mobile ID authentication flow is the following:

1. The end-user uses any application relying on Mobile ID authentication, which sends a mobile signature request through the dedicated web interface (hosted by the application provider) that includes the personal identification number (MSISDN, a number that uniquely identifies a mobile phone subscription, commonly called *phone number*)
2. The application provider receives the end-user request, forms the contents to be signed, and sends the request to the Mobile ID server providing the mobile signature service.
3. The mobile signature server receives the signature request and validate the application provider in accordance with the service agreement.
4. The mobile signature server ensures that the end-user signature request is allowed and forwards it to the end-user mobile device.
5. The end-user gets a message on his mobile phone to sign the mobile signature request. The end-user confirms the authentication request either by providing the Mobile ID PIN (only if SIM/eSIM method is used) or using a passcode / biometric authentication (if Mobile ID app is used).
6. After the application provider has received a valid response, the end-user will be granted access to the corporate application.

Source: <https://github.com/MobileID-Strong-Authentication/mobileid-api/blob/main/doc/mobile-id-reference-guide-v-3-9.pdf>

Main advantages of this method are:

- Strong two-factor authentication
 - You have physical access to the SIM/eSIM (*possession factor*)
 - You need to enter a personal Mobile ID PIN (*knowledge factor*)
- High security level
 - Authentication is done through a secure and isolated (encrypted) channel
 - Tamper-proof key storage based on EAL5+ and Evaluation level 3 certified hardware
- Mobile ID STK application is preinstalled on the SIM/eSIM
- Supported by most Swiss Mobile network operators: Swisscom, Salt, Sunrisem, M-Budget

Mobile ID app: An application provider (in this study case *PostFinance*) can request App-based authentication.

The user must first install the Mobile ID app on their mobile devices (available on both [Google Play Store](#) and [Apple App Store](#)) and then proceed by linking the Mobile ID app with the mobile device by accessing the official Mobile ID web portal (available at <https://www.mobileid.ch/>) and scanning a QR code.



Main advantages of this method are:

- Strong two-factor authentication
 - You need physical access to the mobile device (*possession factor*)
 - In order to get access to the device you need to know the phone passcode (*knowledge factor*) or use biometric authentication such as FaceID, fingerprint, etc. (*Inherence factor*)

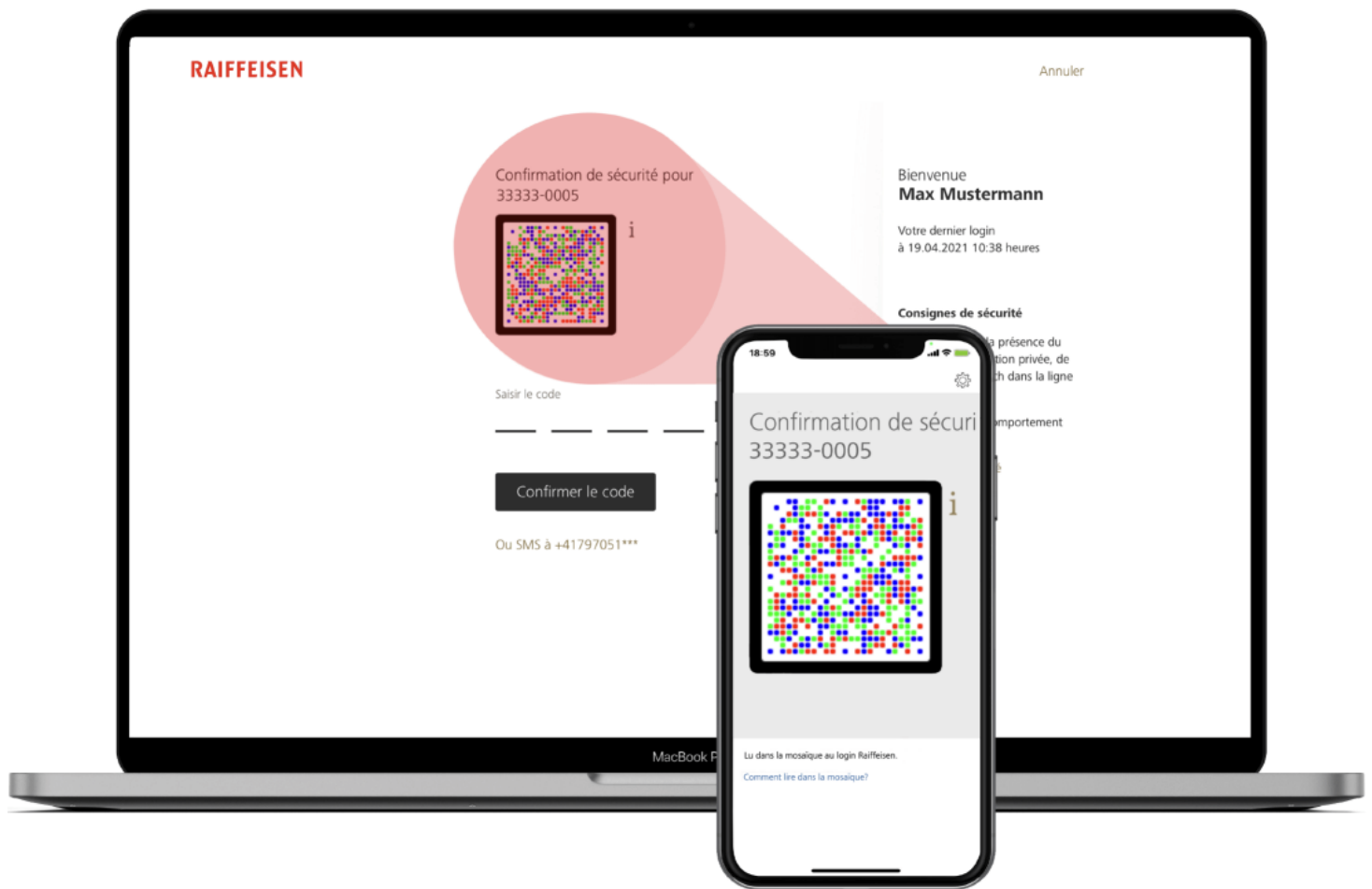
Learn more about MobileID API and certificates: <https://github.com/MobileID-Strong-Authentication/mobileid-api>

3.2.2. Raiffeisen - PhotoTAN

Raiffeisen is another Swiss bank that offers a wide range of financial services, including banking, insurance, and asset management. It is a subsidiary of Raiffeisen Schweiz, which is the Swiss Raiffeisen banking group.

User can access the online banking service using the Raiffeisen Mobile App (which is available for both Android and iOS) or using the web portal. If PhotoTAN two-factor authentication is enabled, after the user enters their username and password they will be asked to confirm the login attempt by scanning a PhotoTAN mosaic image using the official Raiffeisen PhotoTAN mobile app (available on both [Google Play Store](#) and [Apple App Store](#)). After the user scans the PhotoTAN mosaic image, the app will generate a 9-digit code that the user must enter in the web portal to confirm the login attempt.

This is an example using the Raiffeisen PhotoTAN mobile app:



Source: <https://www.raiffeisen.ch/rch/fr/outils-calculateurs/aide-login/login-avec-phototan.html>

If the user does not have a smartphone, they can use a PhotoTAN reader to scan the mosaic image. The PhotoTAN reader is a small device that can be purchased from Raiffeisen bank branches.



Source: <https://www.raiffeisen.ch/rch/fr/outils-calculateurs/aide-login/login-avec-phototan.html>

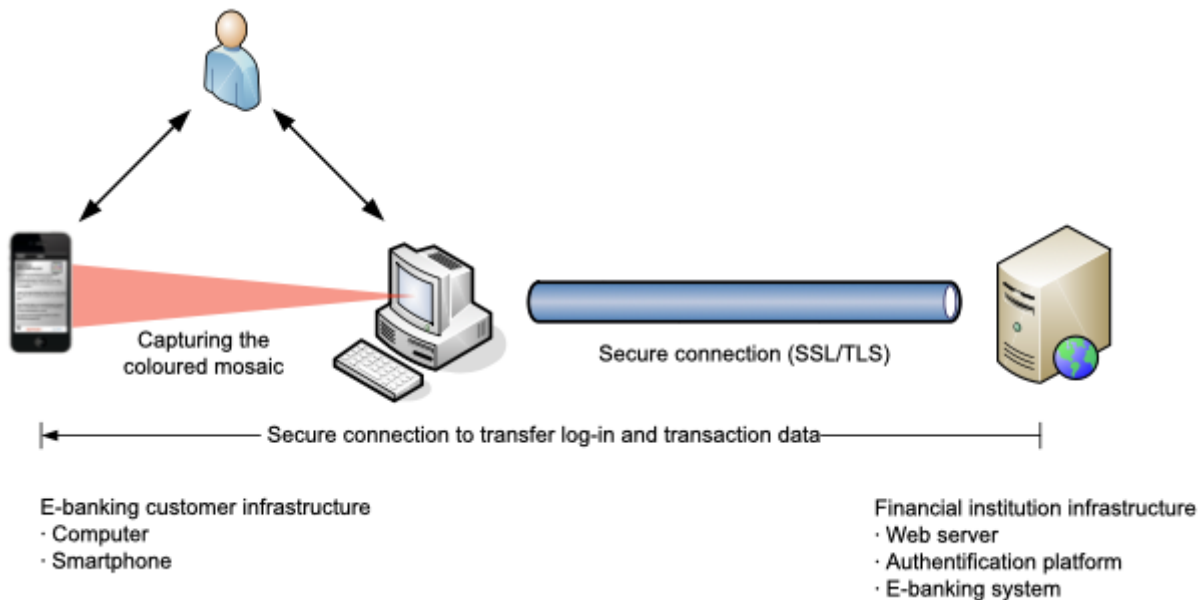
If you have not yet activated your e-banking contract for PhotoTAN, log in to e-banking and choose the option "Log in" in your profile under "Security", then activate PhotoTAN. A mail will be sent to you by post within two to three working days.

Inside the mail can be found a PhotoTAN mosaic image needed to link a photoTAN reader (reader device or smartphone) to the user's e-banking contract.

Source: <https://github.com/MobileID-Strong-Authentication/mobileid-api/blob/main/doc/mobile-id-reference-guide-v-3-9.pdf>

PhotoTAN in depth

This is an example of PhotoTAN authentication flow:



Unfortunately, there is any technical documentation available to understand how PhotoTAN works. We can only guess that the mosaic image is a QR code that contains the user's username and a random N-digit code. The PhotoTAN mobile app or reader scans the QR code and with an algorithm calibrated against the user's "first mosaic image" (Sent by mail, needed to link the PhotoTAN reader to the user's E-Banking account) computes the N-digit code that the user must enter in the web portal to confirm the login attempt.

The main advantages of this method are:

- You need physical access to the smartphone or PhotoTAN reader (*possession factor*)
- You need to know the smartphone passcode or use biometric authentication (*knowledge factor*)

The main disadvantages of this method are:

- Since a physical PhotoTAN reader does offer any kind of authentication method, a linked device can be stolen and used by an attacker to login to the user's account. In fact, raiffeisen has a [FAQ](#) page where they explain how to handle this situation:

Sources:

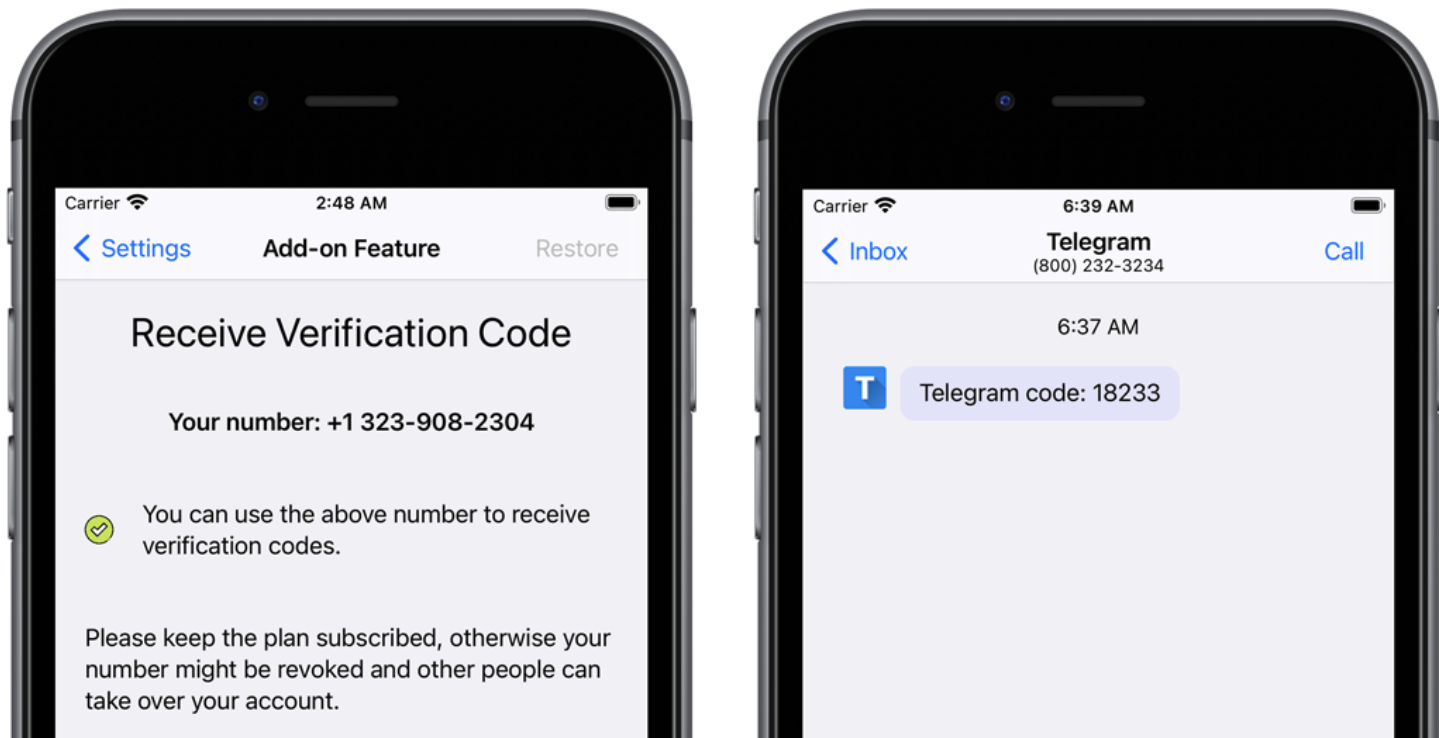
- <https://www.raiffeisen.ch/rch/de/privatkunden/e-banking/mobile-banking-zahlungen-scannen.html>
- <https://www.raiffeisen.ch/content/dam/www/rch/pdf/e-banking/broschueren/de/phototan-geraet-raiffeisen.pdf>
- <https://www.raiffeisen.ch/rch/fr/outils-calculateurs/aide-login/login-avec-phototan.html>

3.2.3. Telegram - mTAN

Telegram is a cloud-based instant messaging service that allows users to send text messages, voice messages, images, videos, and other files. It is available on both Android and iOS.

User can access the Telegram service using the Telegram Mobile App (which is available for both Android and iOS) or using the web portal. If two-factor authentication is enabled, after the user enters their username and password they will be asked to confirm the login attempt by entering a 6-digit code sent via SMS to the user's mobile phone.

This is an example using the Telegram Mobile App:

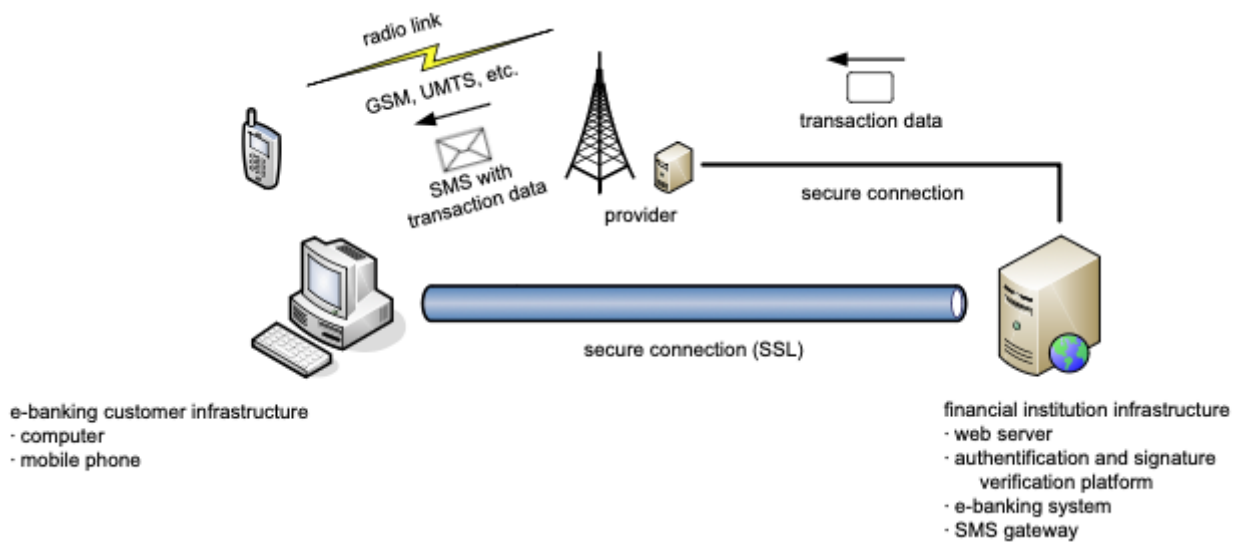


Source: <https://autobizline.com>

mTAN in depth

mTAN (*Mobile Transaction Authentication Number*, known also as Mobile-TAN or SMS-TAN) is a one-time password sent via SMS to the user's mobile phone to confirm an operation. It is a common method used by banks and other financial institutions to confirm a transaction but also used by many other services such as social networks, cloud providers, messaging services, etc.

This is an example of mTAN authentication flow:



The main advantages of this method are:

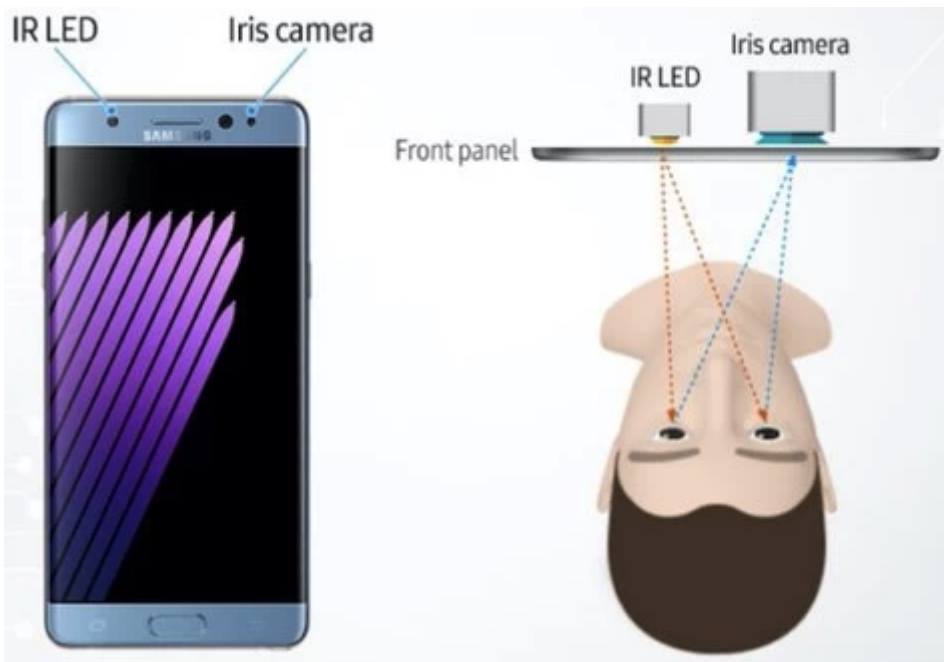
- Strong two-factor authentication
 - You need physical access to the mobile device (*possession factor*)
 - In order to get access to the device you need to know the phone passcode (*knowledge factor*) or use biometric authentication such as FaceID, fingerprint, etc. (*Inherence factor*)
- No need to install any additional software

Source: <https://www.ebas.ch/en/mobile-tan-mtan-sms-tan/>

3.2.4. Samsung - Biometric authentication

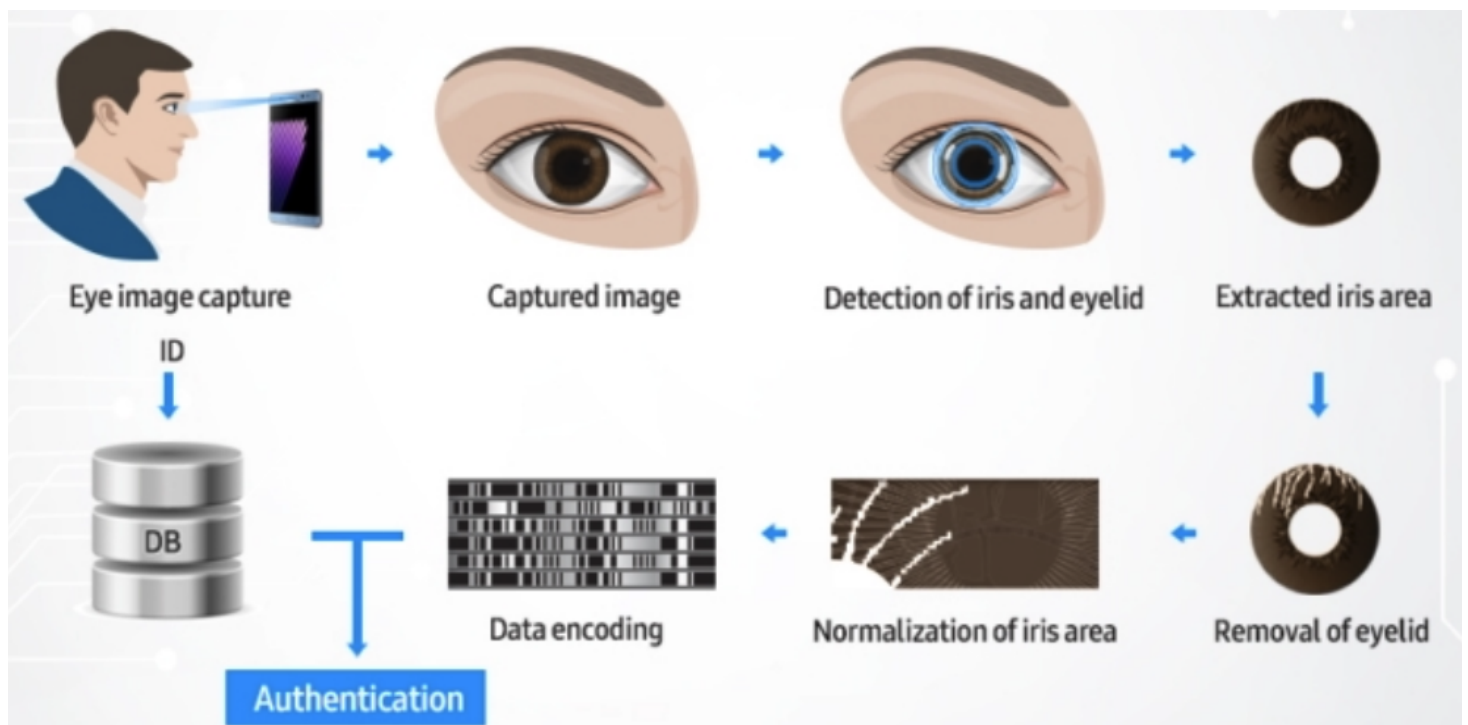
Samsung is a South Korean multinational conglomerate headquartered in Samsung Town, Seoul. It is the largest South Korean chaebol (business conglomerate). Samsung is the world's largest manufacturer of smartphones, smartphones, and memory chips. It is also the world's largest manufacturer of televisions and washing machines.

Samsung in 2017 introduced in the Samsung Note 7 an iris scanner based authentication to unlock the device. The iris scanner is a biometric sensor that recognizes the user's iris pattern, which is unique for each person and virtually impossible to replicate, and allows the user to unlock the device just by looking at it. The iris scanner is located on the front of the device, at the top of the screen next to the camera:



Source: <https://www.pocket-lint.com/phones/news/samsung/138335-samsung-galaxy-note-7-iris-scanner-what-is-it-and-how-does-it-work>

This is the iris scanner based authentication flow on Samsung Note 7:



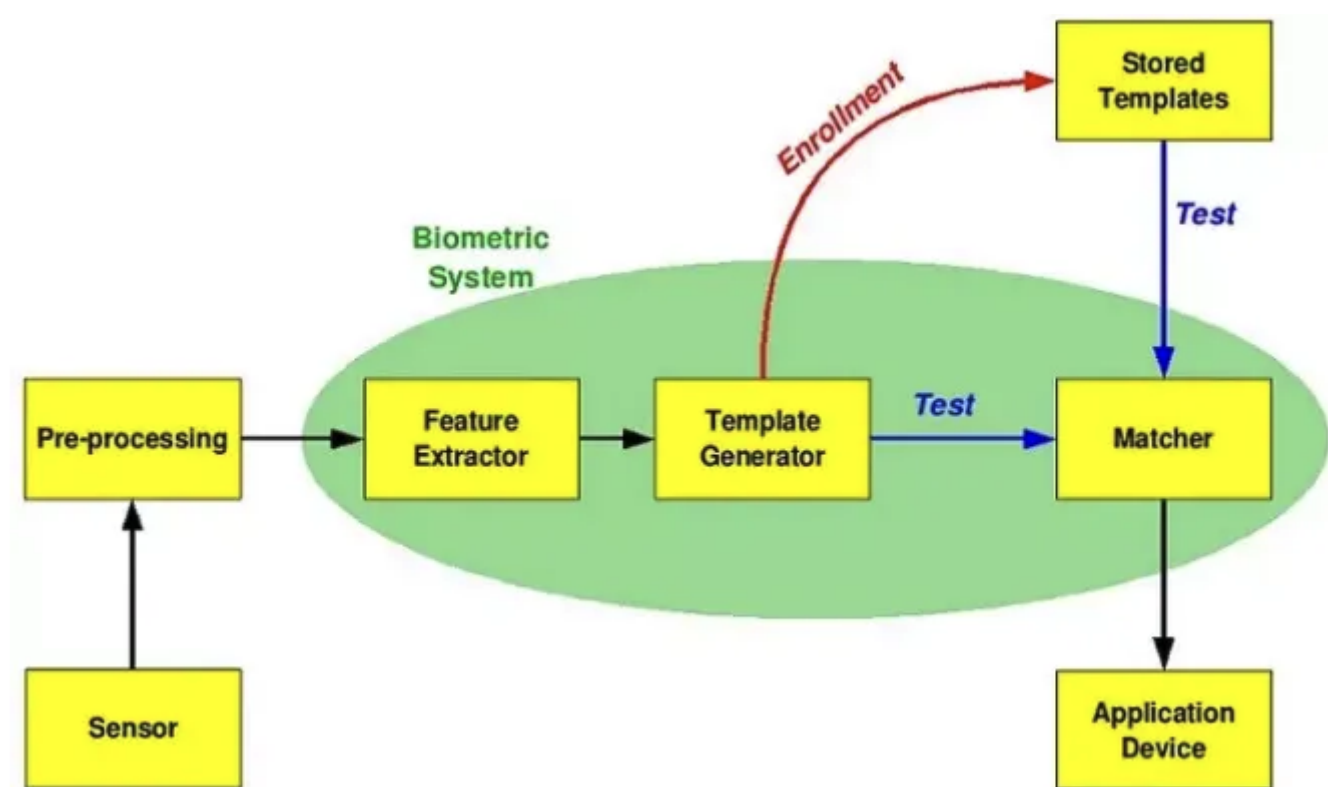
Source: <https://news.samsung.com/global/in-depth-look-keeping-an-eye-on-security-the-iris-scanner-of-the-galaxy-note7>

Biometric authentication in depth

Biometric authentication is a method of authentication that uses the user's physical characteristics to confirm the identity of the user. The most common biometric authentication methods are fingerprint, face recognition, and iris recognition. Biometric authentication is one of the most secure authentication methods available today.

To be able to use biometric authentication, the user must first enroll their biometric data to be used to compare against the fresh collected biometric data during the authentication process. The biometric data is stored in a secure enclave on the systems and is never shared.

This is an generic example of biometric authentication flow:



The main advantages of biometric authentication are:

- It is very easy to use, the user does not need to remember any password or PIN code.
- It is very secure, the biometric data is unique for each person and virtually impossible to replicate.

The main disadvantages of biometric authentication are:

- It is not available on all devices, only on devices that have a biometric sensor.
- It is not available on all services or platforms, only on services that support biometric authentication.
- Could be expensive to implement, the biometric sensor must be integrated into the device.

4. Authentication in APIs and Web Services

In this section will be presented the most common authentication methods used in APIs and Web Services: JSON Web Tokens (JWT), OAuth 2.0 and SAML.

4.1. JSON Web Tokens (JWT)

JSON Web Tokens (JWT) is an open standard ([RFC 7519](https://www.rfc-editor.org/rfc/rfc7519)) 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, needed to be able to verify a token signature later during the authentication process.

The token consist of three parts separated by dots (.), which are:

- **Header** - The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.
- **Payload** - The second part of the token is the payload (also called *claim set*), which contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: registered, public, and private claims, which are explained in the next section.
- **Signature** - To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

JWT tokens are used to authenticate users in APIs and Web Services. The user is authenticated by sending the JWT token in the HTTP request header. The server will validate the token and if it is valid, the user will be authenticated. To be able to send the JWT token inside HTTP requests, the token is *base64url* (view more: <https://www.rfc-editor.org/rfc/rfc4648#section-5>) encoded.

This is an example of a JWT token in both encoded and decoded format:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpXVCJ9.eyJpZCY6IjE2MjY2FkaWJlbGxvLmNoIiwiaXNBNZG1pbiI6dHJ1ZX0.LHB5pfAzgn51r3YR6Vr78rBq5P
WwkJVqjmlp8gNKask

HEADER: ALGORITHM & TOKEN TYPE

```
"alg": "HS256",
"typ": "JWT"
}
```

PAYLOAD: DATA

```
"sub": "1234567890",
"name": "Luca Di Bello",
"iat": 1516239022,
"iss": "lucadibello.ch",
"isAdmin": true
}
```

VERIFY SIGNATURE

```

HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
    my-personal-secret
)

```

☐ secret base64 encoded

Source: <https://jwt.io>

4.1.1. JWT claims

JWT claims are statements about an entity (typically, the user) and additional data. There are three types of claims: registered, public, and private claims.

Registered claims:

These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of the most common registered claims are:

- **iss** (issuer) - The issuer of the token (the server that generated the token).
 - Example: `iss: "https://example.com"`
- **sub** (subject) - The subject of the token, which usually is the user ID.
 - Example: `sub: "1234567890"`
- **aud** (audience) - The audience of the token (the server that will validate the token).
 - Example: `aud: "https://example.com"`
- **exp** (expiration time) - The expiration time of the token (the date and time when the token will expire).
 - Example: `exp: "2019-10-31T23:59:59.999Z"`
- **nbf** (not before) - The date and time before which the token must not be accepted for processing.

- Example: `nbf: "2019-10-31T23:59:59.999Z"`
- **iat** (issued at) - The date and time in which the token was issued.
 - Example: `iat: "2019-10-31T23:59:59.999Z"`
- **jti** (JWT ID) - The unique identifier of the token (used to prevent the JWT from being replayed).
 - Example: `jti: "1234567890"`

Source: <https://www.rfc-editor.org/rfc/rfc7519#section-3>

Public claims:

These are the custom claims created to share information between parties that agree on using them and are neither registered or private claims.

This kind of claims are defined as public since defined in the [IANA JSON Web Token Registry](#) and can be used without collisions.

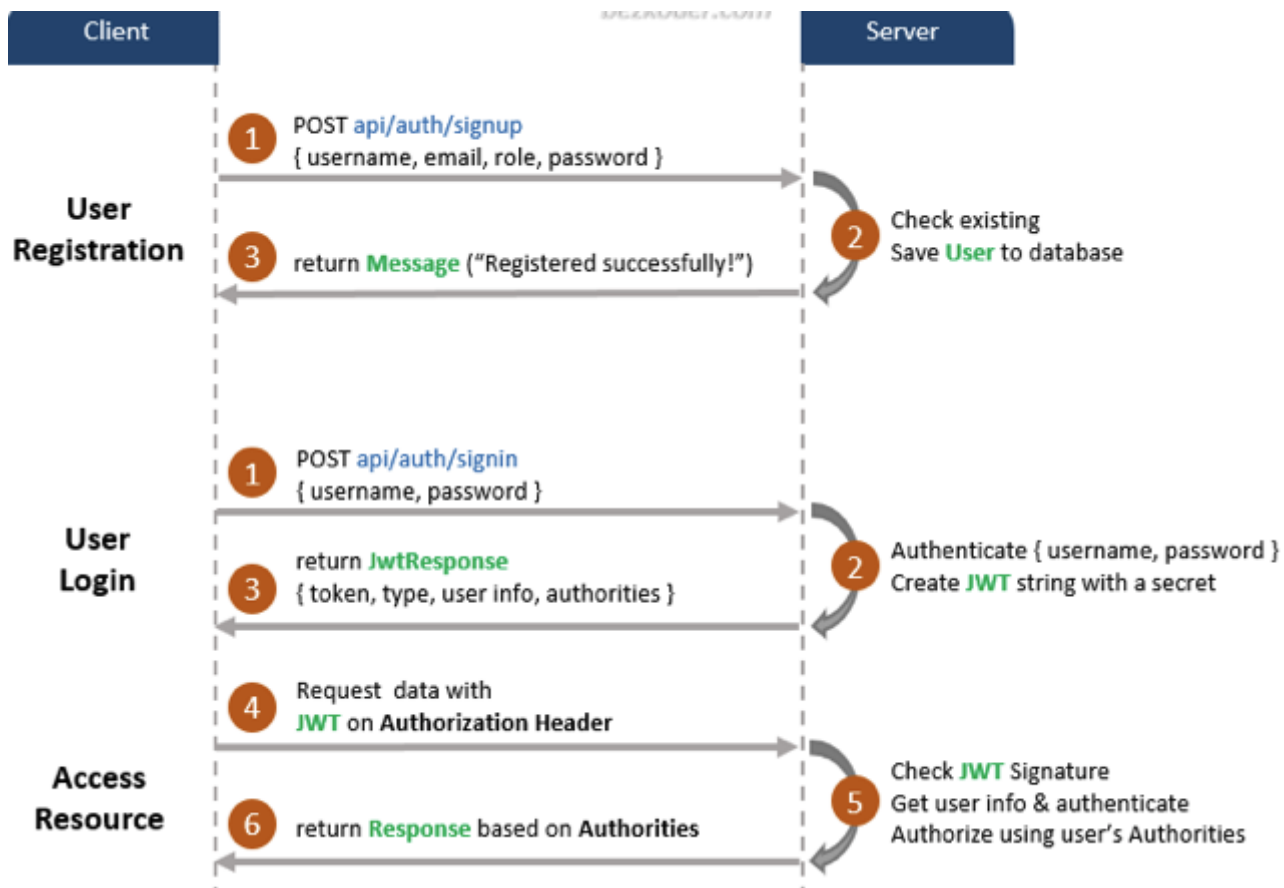
Private claims:

These are the custom claims created to share information between parties that agree on using them and are neither registered or public claims.

This kind of claims are primarily used to share information between parties, such as the user ID, the user role, the user permissions, etc.

4.1.2. JWT authentication flow

This is an example of a JWT authentication flow inside classic web applications:



Source: <https://www.bezkoder.com/spring-boot-jwt-authentication/>

4.1.3. JWT advantages and disadvantages

The main advantages of JWT are:

- JWT tokens are stateless, which means that the server does not need to store any session data for JWT authentication. This makes the server scale easily
- It is very secure, the token is digitally signed and can be verified by the server
- Simple and fast to implement

The main disadvantages of JWT are:

- Very difficult to revoke a token, since the token is stateless, the server does not have any session data to revoke the token
- The token is sent in every request, which could be a security risk if the token is intercepted
- The token payload is not encrypted, which means that the token payload can be read by anyone. This affects privacy and the overall security of the application

Source: <https://jwt.io/introduction/>

4.1.4. JWT privacy considerations

JWT tokens might contain sensitive information about the user, such as the user ID, the user role, the user permissions, etc. This information must be protected and never shared with third parties, since it could be used to as an attack vector.

To prevent privacy issues some of the most common practices are:

- Use an encrypted JWT and authenticate the recipient
- Ensure that JWTs containing unencrypted privacy-sensitive information are only transmitted using protocols utilizing encryption that support endpoint authentication, such as Transport Layer Security (TLS)
- Omitting privacy-sensitive information from a JWT (simplest and most effective solution)

This could negatively affect the user experience, since the user will not be able to access the information that is stored in the JWT token.

Source: <https://tools.ietf.org/html/rfc7523#section-3>

4.2 OAuth 2.0

OAuth 2.0 is an industry-standard ([RFC6749](#)) authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean. This framework is used to provide client applications a "secure delegated access" to server resources on behalf of a resource owner. OAuth 2.0 provides authorization flows for web and desktop applications, and mobile devices.

To allow limited access to user account data OAuth 2.0, instead of using the resource owner's credentials to access protected resources, uses access tokens. An access token is a string opaque to the user (not visible) that represents an authorization issued to a client, and identifies the user, the client, and the scopes and duration of the access granted. Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner and must be used to access protected resources hosted by the resource server.

4.2.1. OAuth 2.0 roles

OAuth 2.0 defines four roles:

- **Resource owner** - The entity that can grant access to a protected resource. When the resource owner is a person, it is referred to as an end-user.
- **Resource server** - The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

- **Client** - An application making protected resource requests on behalf of the resource owner and with its authorization.
- **Authorization server** - The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

Source: <https://www.rfc-editor.org/rfc/rfc6749#section-1.1>

4.2.2. OAuth 2.0 authentication flow

OAuth 2.0 defines four authorization flows:

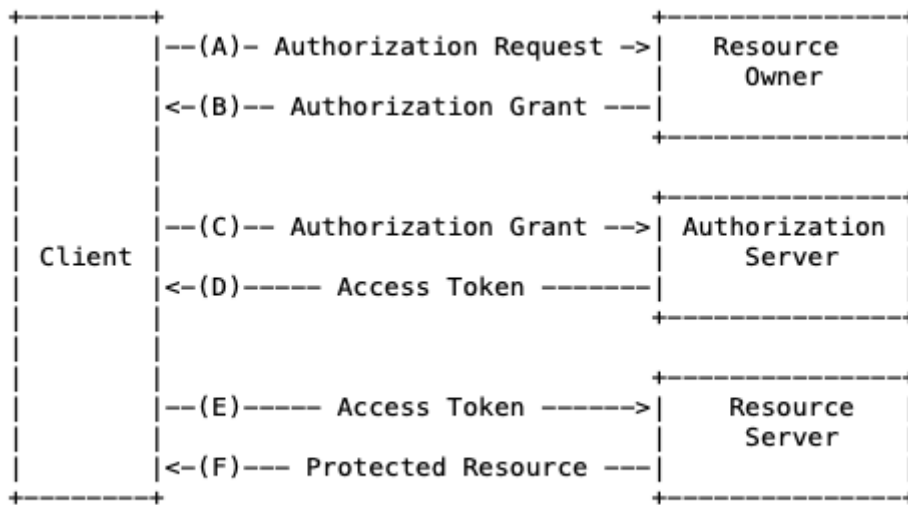


Figure 1: Abstract Protocol Flow

Source: <https://www.rfc-editor.org/rfc/rfc6749#section-1.2>

Steps:

A: The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner, or indirectly via an authorization server as an intermediary (preferable, to learn more see [RFC7649 Section 3.1](#)). The client sends the following parameters:

- **response_type** - The type of the response. The value must be `code`.
- **client_id** - The client identifier as described in [RFC6749 Section 2.2](#).
- **redirect_uri** - The redirection URI to which the response will be sent. This URI must exactly match one of the redirection URIs registered for the client, as described in [RFC6749 Section 3.1.2](#).
- **scope** - The scope of the access request as described by [RFC6749 Section 3.3](#).
- **state** - An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client.

The parameter SHOULD be used for preventing cross-site request forgery as described in [RFC6749 Section 10.12](#).

B: The resource owner grants or denies the access request and sends the client an authorization grant. The grant is a credential representing the resource owner's authorization (to learn more see [RFC6749 Section 1.3](#)). The response can be sent directly to the client, or indirectly via an authorization server as an intermediary (preferable, to learn more see [RFC6749 Section 3.1](#)).

C: The client requests an access token from the authorization server and authenticates by presenting the authorization grant (to learn more see [RFC6749 Section 4.1.3](#)).

D: The authorization server authenticates the client and validates the authorization grant, and if valid, issues an access token (to learn more see [RFC6749 Section 5.1](#)). If the authorization grant is invalid, expired, or revoked, the authorization server returns an error response (to learn more see [RFC6749 Section 5.2](#)). This is a sample HTTP response with a valid authorization grant:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"mF_9.B5f-4.1JqM",
  "token_type":"Bearer",
  "expires_in":3600,
  "refresh_token":"tGzv3J0kF0XG5Qx2TlKWIA"
}
```

We can see that the response contains both access and refresh token. The access token is used to access the protected resources, while the refresh token is used to obtain a new access token when the current access token expires. Learn more in chapter **4.2.4. Refresh token**.

Source: <https://www.rfc-editor.org/rfc/rfc6750#section-4>

E: The client requests the protected resource from the resource server and authenticates by presenting the access token (to learn more see [RFC6749 Section 7.1](#)). The token is sent in the HTTP Authorization header using the Bearer authentication scheme (to learn more see [RFC6750 - Bearer Token Usage](#)).

4.2.3. JWT Bearer Token

The JWT Bearer Token is a security token that is used to access an OAuth 2.0 protected resource. The token is a JSON Web Token (JWT) that is signed using a private key. The token is sent in the

HTTP Authorization header using the Bearer authentication scheme (to learn more see [RFC6750 - JWT Access tokens](#)).

Source: <https://www.rfc-editor.org/rfc/rfc6749#section-1.5>

4.2.4. Refresh token

OAuth 2.0 defines a refresh token, which is a credential used to obtain fresh access token. The refresh token is issued to the client by the authorization server and is used to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope (access tokens may have a shorter lifetime and fewer permissions than authorized by the resource owner). The refresh token is issued only if the authorization server supports the use of refresh tokens and only if the client has provided the appropriate authorization. The authorization server issues a new refresh token, and invalidates the old refresh token after issuing a new access token.

This tokens are meant to be used only with authorization servers, should not be shared with other parties, and should be stored securely on the device.

This is the refresh token flow:

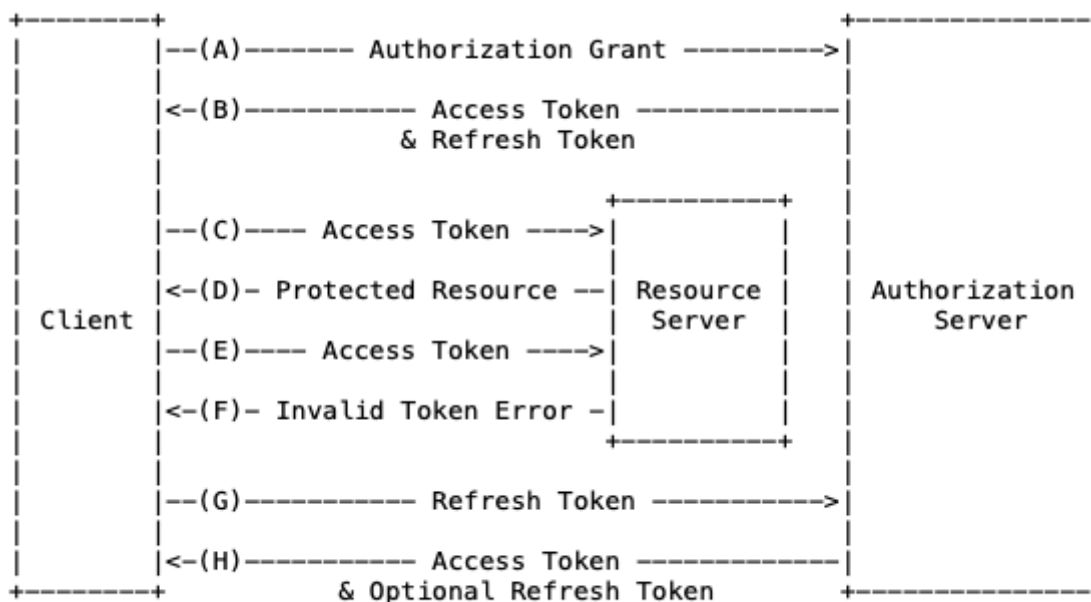


Figure 2: Refreshing an Expired Access Token

Source: <https://www.rfc-editor.org/rfc/rfc6749#section-1.5>

We can see that the refresh token is only used to obtain a new access token when it becomes invalid or expires. To obtain a new access token, the client sends the refresh token to the authorization server, which validates the refresh token and issues a new access token and (optionally) also a new refresh token.

1.5. Demo

To showcase a practical example of a token-based authentication system, will be presented a NextJS 13 application that uses JWT tokens (access token and related refresh token) for authentication and authorization, with a two-factor authentication via E-Mail. This application is a boilerplate that can be used as a starting point to implement custom JWT-based authentication for any NextJS ≥ 13 application.

1.5.1. Main features

- Fully-typed with TypeScript
- Login with email and password (hashed with bcrypt)
- Role-based access control (by default: *User*, *Admin*)
- Automatic JWT access token refresh
- Two-factor authentication via email
- Front-end `useAuth` hook to easily manage the user session
- User session persistence via cookies and local storage
- New flexible back-end middleware management system
- Protected routes and pages

1.5.2. Tech stack

- NextJS v13
- TypeScript
- Chakra UI
- React Hook Form
- SWR (stale-while-revalidate)
- Prisma ORM
- jsonwebtoken

1.5.3. Authentication flow

The authentication flow is the following:

Example two-factor authentication link:

`http://localhost:3000/two-factor?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Imw:`

1.5.4. Screenshots and short demo

Login page:

Log In



Log In

Don't have an account? [Sign up](#)

Two-factor authentication pages:

Two-Factor authentication landing page:



Two Factor Authentication

Please, check your email for the two factor authentication link and click on it. You can close this page now.

If you haven't received the email, please check your spam folder.

Two-Factor authentication success page:



Two Factor Authentication

You have been successfully authenticated. You will be redirected to the home page in a few seconds.

Two-Factor authentication error page:



Two Factor Authentication

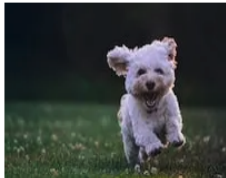
This link is invalid or has expired. Please, log in again to get a new link.

Dashboard page:

User ID 2
Authenticated? Yes
Username: Jane White
Email: luca6469@gmail.com
Admin: Yes

Access token:	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZiZCI6ImwiZW1haWwiOiJsdWNhbjQ2OUBnbWVpYyJ20iLlCjUwY1lljoIjSmFzS1lnN1cm5hbnUuIjAgaG0ZS1lnJvbGUiOiJBRE1JTlslmlhdCl6MTY2OTU0MzI4NiwiZCJwXWljoxNjY5NTQ0MTg2fQ.X3eVUEk6KR-2eOIl0kc-OzVE4lOKVg7h7eYXuYguUpl
Refresh token:	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZiZCI6ImwiZW1haWwiOiJsdWNhbjQ2OUBnbWVpYyJ20iLlCjUwY1lljoIjSmFzS1lnN1cm5hbnUuIjAgaG0ZS1lnJvbGUiOiJBRE1JTlslmlhdCl6MTY2OTU0MzI4NiwiZCJwXWljoxNjY5NTQ0MTg2fQ.4m5b5Oe1Ng50qgkJkYNYaZagRWNPgVY982cdOBQcqr3Dk

Please, like or dislike the posts below. The data is persisted in the DB.



Cute dogs 24/12/2021, 05:47:43

This is a post about cute dogs, come and see them!

↑ 0 ↓ -2



Cute cats 16/02/2021, 03:04:13

This is a post about cute cats, come and see them!

↑ 1 ↓ -1

Showcase video