Authentication and Authorization

Whats the difference

Authentication verifies the identity of a user or service.

Authorization then verifies this the access rights of an verified entity.

Ways of authentication on the web

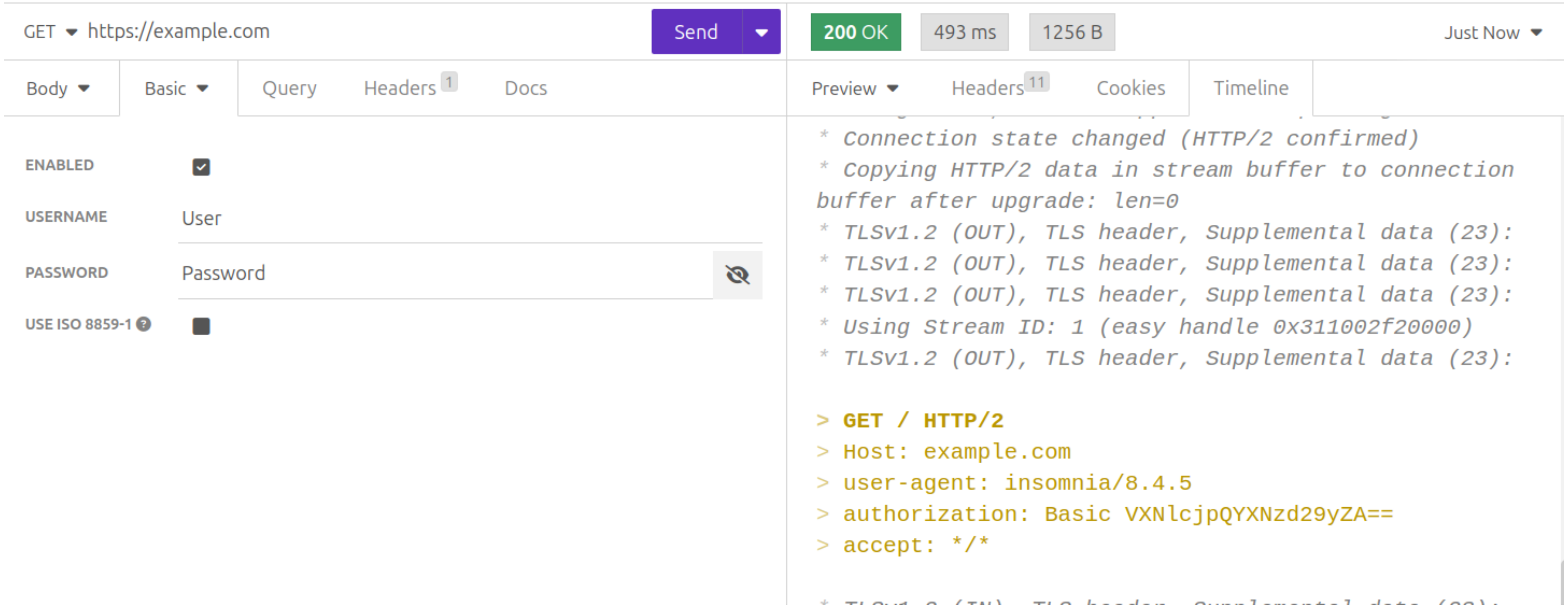
- Basic authentication
- Cookie-based
- Token-based
- Magic links
- OAuth and OAuth provider
- Passkeys

Basic authentication

Basic authentication

One of the earliest forms of authentication.

It needs to be sent in the `Authorization` header and is not very secure.



ENABLED	✓
USERNAME	User
PASSWORD	Password
USE ISO 8859-1	■

```
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection
buffer after upgrade: len=0
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* Using Stream ID: 1 (easy handle 0x311002f20000)
* TLSv1.2 (OUT), TLS header, Supplemental data (23):

> GET / HTTP/2
> Host: example.com
> user-agent: insomnia/8.4.5
> authorization: Basic VXNlcjpQYXNzd29yZA==
> accept: */*
```

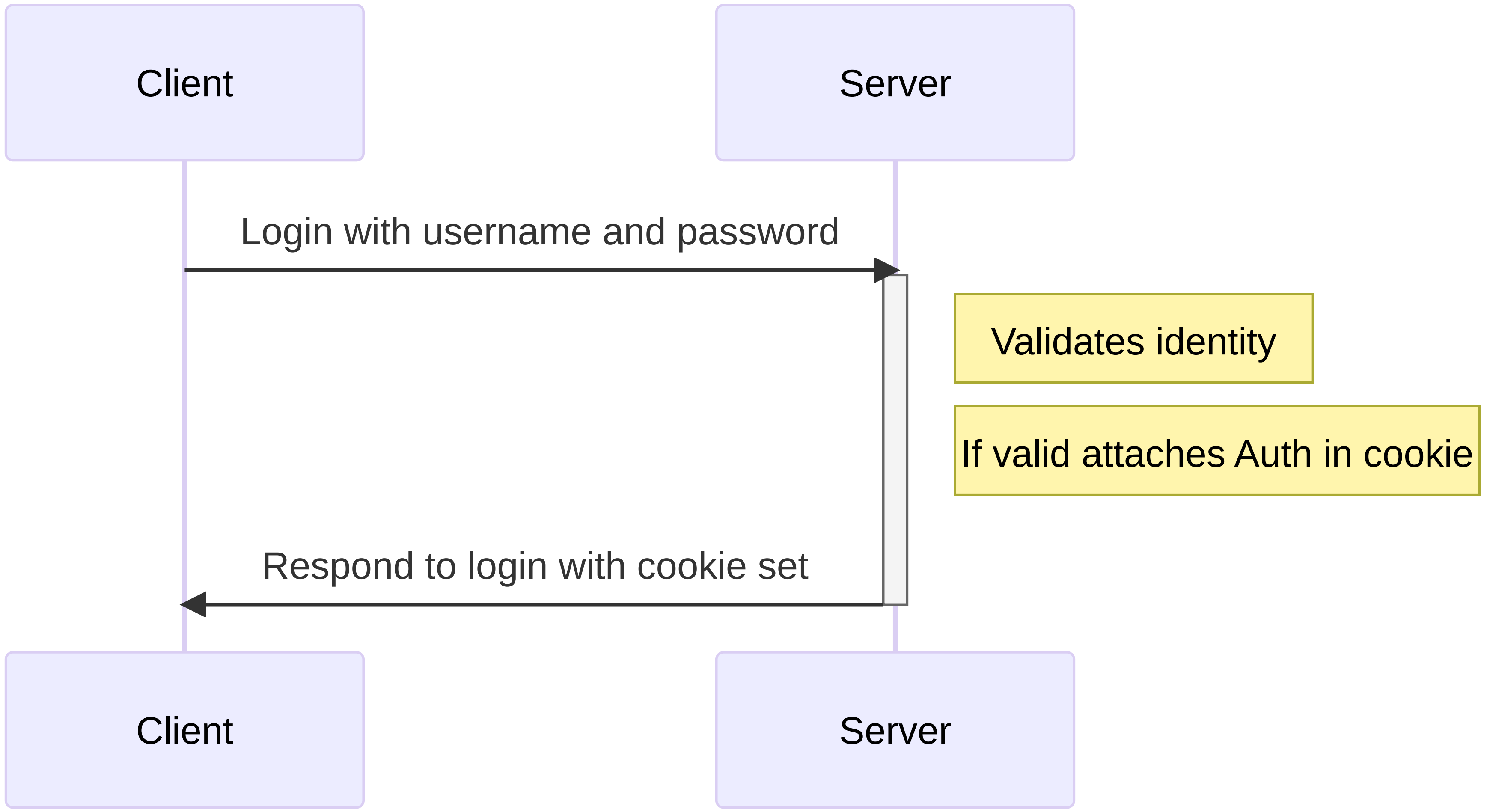
Basic authentication

The primary reason for it being unsecure, is that the value for the Basic authentication, is not encrypted in any way. The value is simply a concatenation of `username:password` and then base64 encoded.

```
const token = btoa('myUsername:password123');
fetch('https://example.com', {
  headers: {
    Authorization: `Basic ${token}`
  }
});
```

Cookie-based authentication

Authentication flow


Client
Server
Login with username and password
Validates identity
If valid attaches Auth in cookie
Respond to login with cookie set
Client
Server

Pros and cons of cookie authentication

Pros

- Very easy to get started with
- No additional logic on the client required, as the cookies attach to the domain.
- Easy to revoke on the server side

Cons

- Hard to scale once the server side needs to scale
- Limited in size, most browsers cap cookies at 4Kb.

Token-based

What is a considered a token

A token can be any type of serializable information that identifies a user entity directly. A token can be contained within an in-memory storage or on hardware devices.

- Connected tokens
- Contactless tokens
- Disconnected tokens
- Software tokens

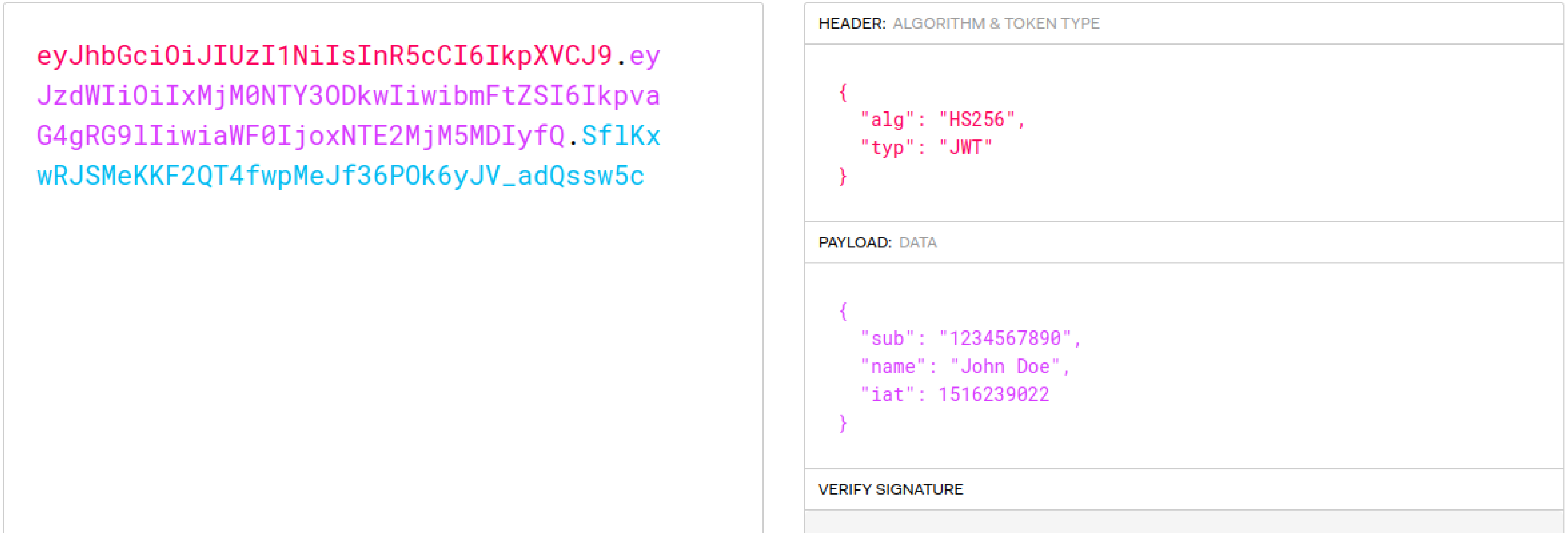
JWT - JSON web tokens

JSON web tokens

Are a standardized way to transmit self contained information in form of a json object. It's usage is not limited to authentication flows, but it's primarily used for it.

JWTs consist of three parts, that make up the information:

`header` . `payload` . `signature`



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

JWT Header

The header simply defines the algorithm that was used to sign the token and what type of token it is.

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

This information is then base64 encoded and put into the header part of the JWT.

JWT Payload

The payload consists of *claims* that describe the entity (in most cases the user).

There are three types of claims, that can go into the payload:

- Registered claims
- Public claims
- Private claims

JWT Payload

The payload is again a json object that consists of the defined or generated claims by the JWT producer.

```
{
  "iat": 1700769154,
  "exp": 1700769214,
  "email": "office@example.com",
  "id": 1
}
```

This information is then base64 encoded and put into the payload portion of the JWT. Since this is only base64 encoded, this can be easily read again, so avoid adding sensitive information into the payload of a JWT.

JWT Signature

The signature portion is the thing that makes the JWT secure or at least tamper proof.

The signature takes the header and payload portion of the JWT, takes a secrect that only the producer / consumer of the JWT knows, and creates a signature.

This signature is then put into the signature portion of the JWT and completes the whole token.

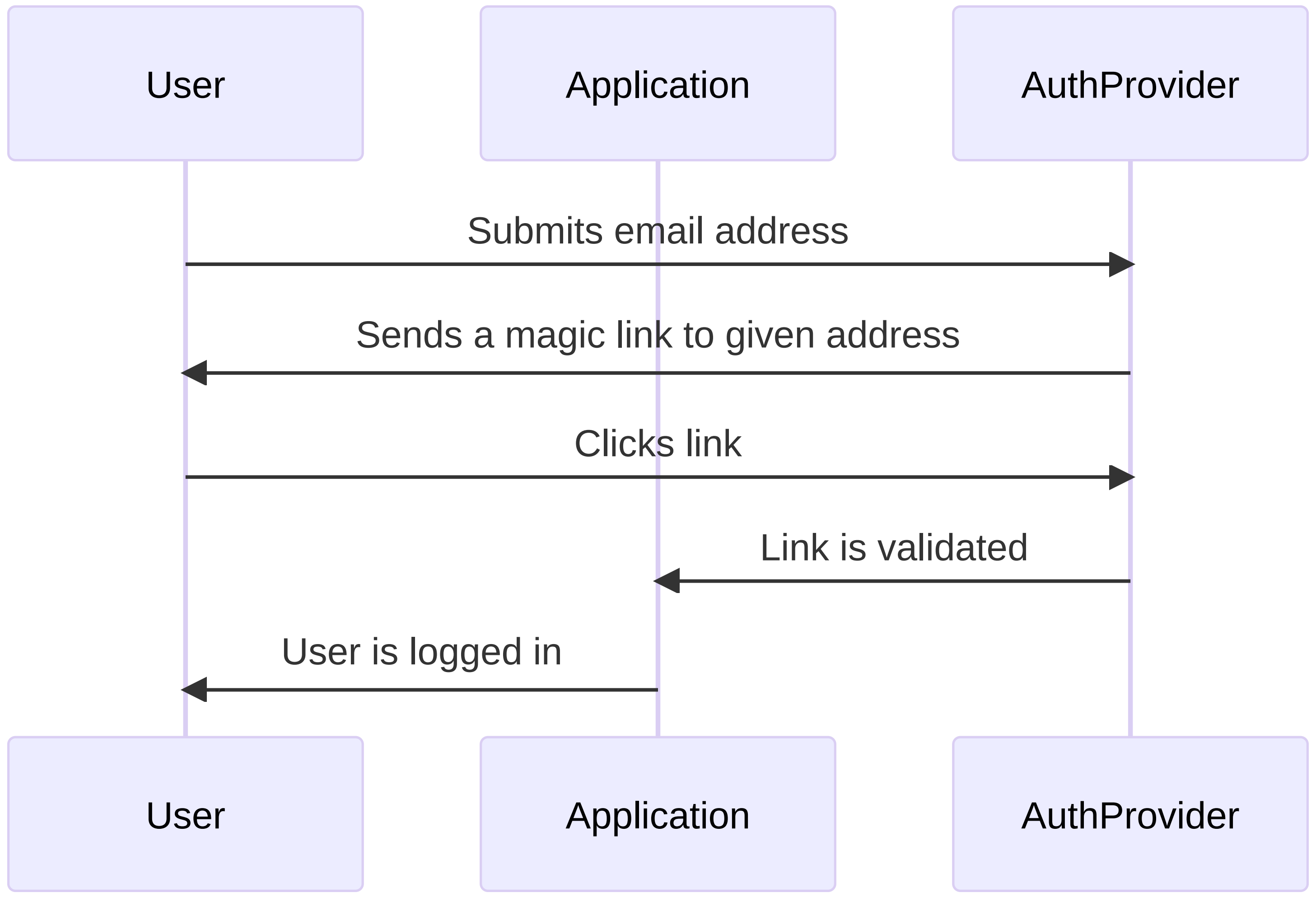
Magic links

Magic links or One time passwords

Magic link authentication workflows are an emerging authentication strategy, that does not require a password. Instead it relies on the fact that the entity requesting the authentication, has access to an already secured form of communictation, mostly email or text message.

It uses a one time password approach with an expiration date to authenticate a user.

Magic link flow



Pros and cons of Magic link authentication

Pros

- Due to the one time password, it can be more secure
- Fairly simple to implement and to use.

Cons

- Requires the same device that opens the email to be the one that gets logged in.
- If access to the email or text message is compromised, all security is gone.

Third party access (OAuth)

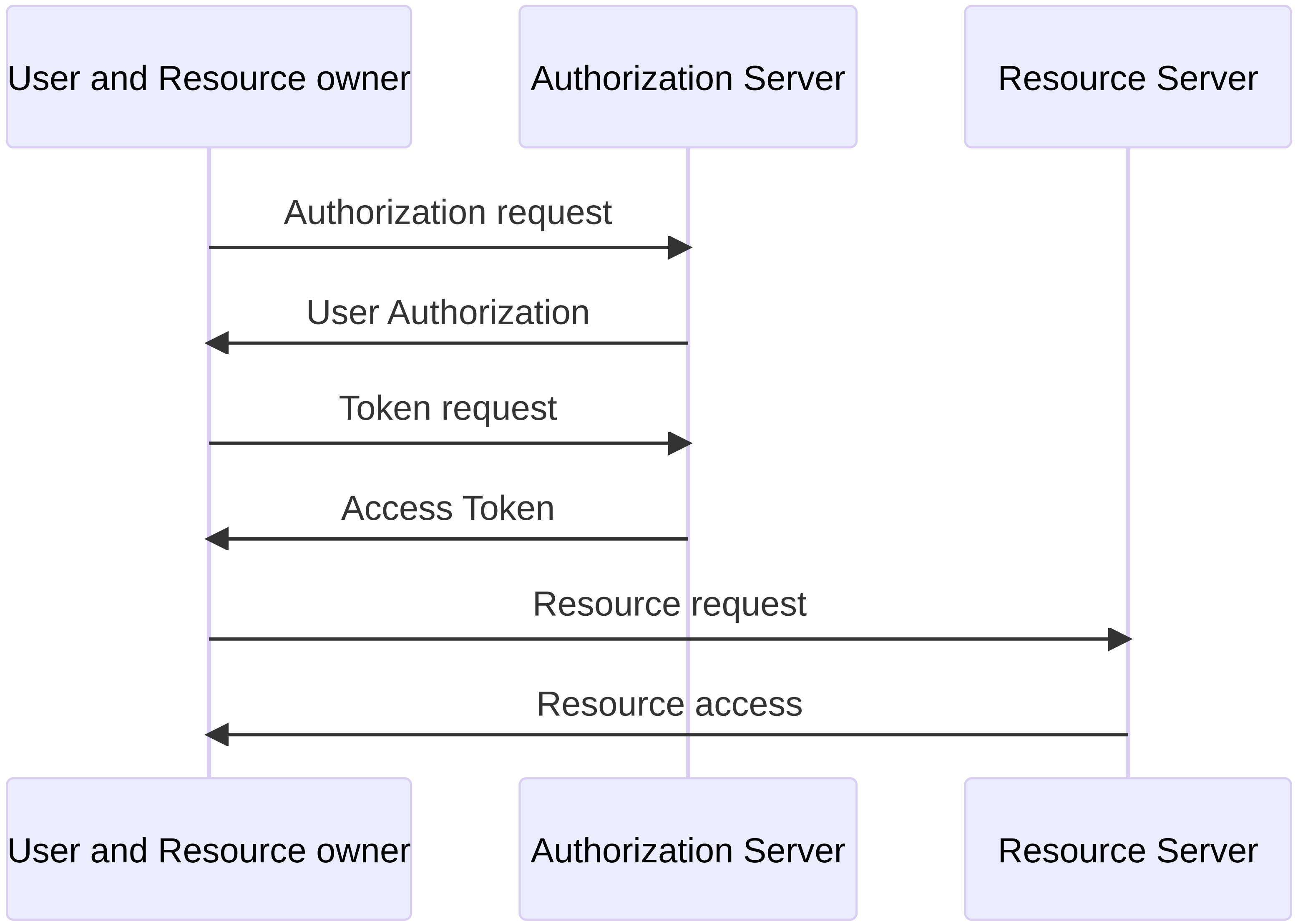
OAuth

OAuth is an open standard authorization protocol or framework that provides applications the ability for *secure designated access*.

Popular providers for OAuth are:

- Social network sites like twitter, facebook, ect.
- Amazon
- Google

OAuth flow


User and Resource owner
Authorization Server
Resource Server
Authorization request
User Authorization
Token request
Access Token
Resource request
Resource access
User and Resource owner
Authorization Server
Resource Server

Multi factor authentication

Factors of authentication

Knowledge factor

Refers to something that the user *knows*, such as passwords, pin numbers or even answers to very personal questions.

Possession factor

Refers to something that the user *has*, such as a mobile phone or a key generator.

Biometric factor

Refers to something that the user *is*, such as fingerprints, voice patterns, iris scans or face id.

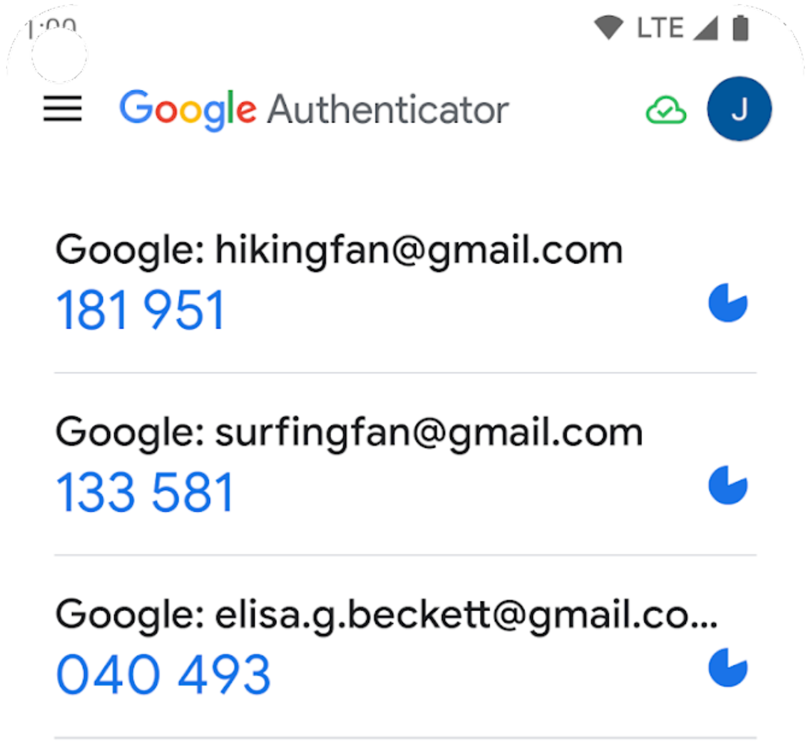
Concept of multi factor authentication

The basic idea is that for authenticating an entity, at least two factors are picked.

Most of the time it is a *knowledge factor* and a *possession factor*, such as password and form of key generation.

Authenticator tokens

Authenticator tokens work by intially sharing a generated secret and an agreement on how to generate new keys based on the current time and the secrect, these keys are generated.



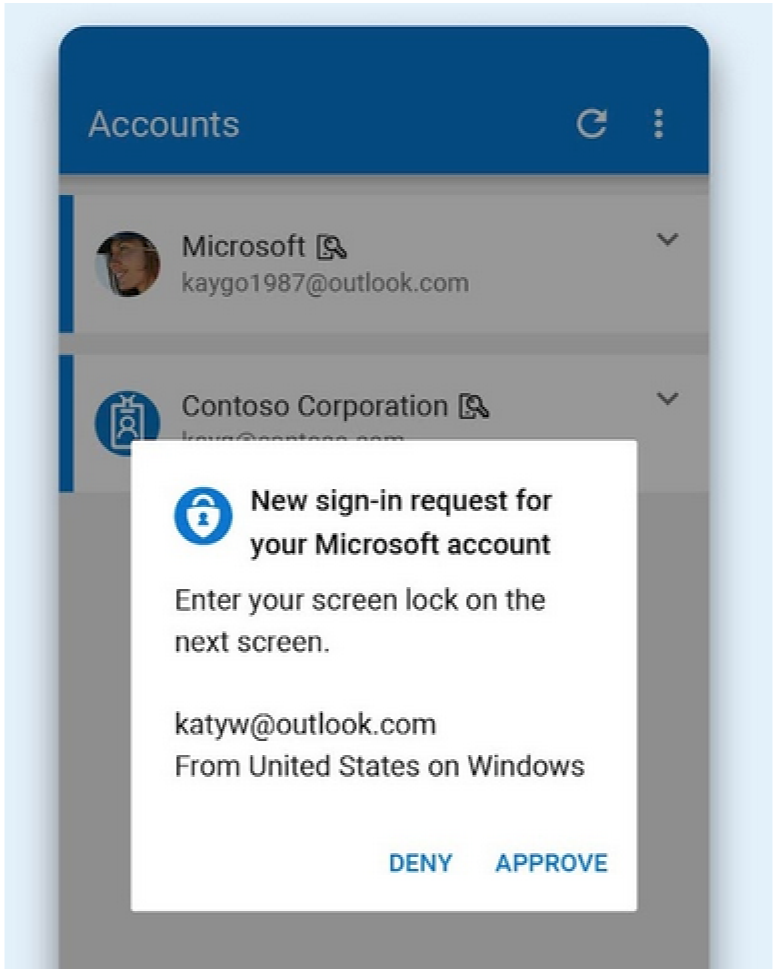
Google: hikingfan@gmail.com
181 951

Google: surfingfan@gmail.com
133 581

Google: elisa.g.beckett@gmail.co...
040 493

Simpler validation options

It's not entirely necessary to have generated authenticator tokens, most of the time, it's a one time code or a simple acceptance interaction.



New sign-in request for your Microsoft account

Enter your screen lock on the next screen.

katyw@outlook.com
From United States on Windows

Passkeys

Concepts of passkeys

- Passkeys are cryptographic keys and each passkey consists of a private and a public key
- Passkeys unlocks with biometrics or device access
- The public key gets shared with the website wheras the private key and the biometric information will always remain on the device.

Authorization types

- Entity based access control
- Role based access control - RBAC
- Access control list - ACL

Resources

- JSON web tokens
- Multi factor authentication
- Implementing passkeys
- OAuth docs
- Role based access control
- Role based access control in nest
- Access control list