Authentication

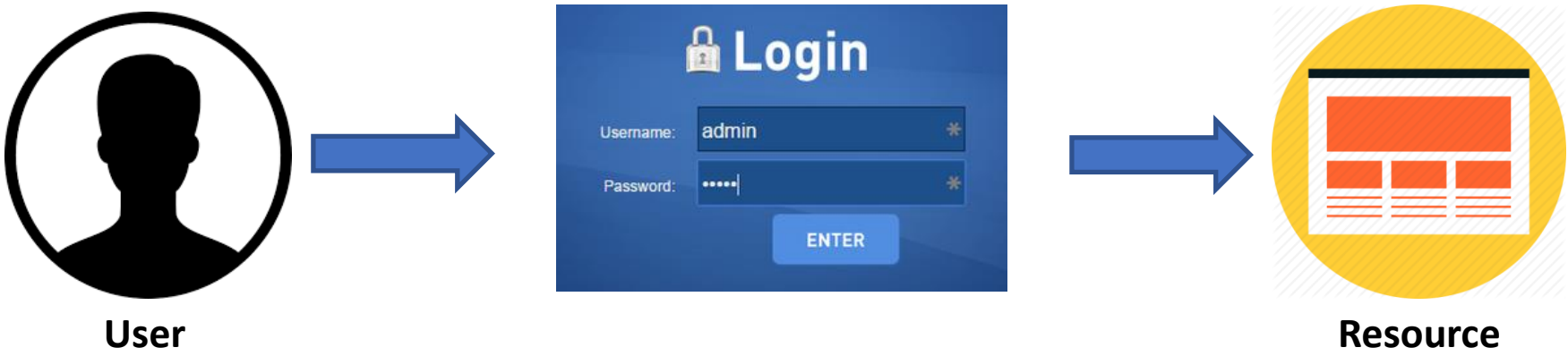
Outline

1. Password-based authentication
2. 2-Factor Authentication (2FA)
3. 2FA in Practice
4. Single Sign-On (SSO) using Token-based Security
5. Delegated Authentication

Application Security Aspects

- **Authentication (Identity verification):**
  - Verify the identity of the user given the credentials received
  - Making sure the user is who he claims to be
- **Authorization:**
  - Determine if the user should be granted access to a particular resource.
- **Confidentiality:**
  - Encrypt sensitive data to prevent unauthorized access in transit or in storage
- **Data Integrality:**
  - Sign sensitive data to prevent the content from being tampered (e.g., changed in transit)

Password-based Authentication



Passwords

- Passwords are the most commonly used authenticator in computer systems
  - A combination of letters, numbers, and special characters that a user supplies in order to prove their identity
  - Frequently passwords are chosen by the user
- But security problems associated with passwords:

Source: https://enterprise.verizon.com/resources/reports/dbir/

Password Vulnerabilities

- Actually not very secure
  - Weak password: Users tend to pick simple passwords and easy to guess passwords
    - *123456* and *password* are the most common passwords!!!
    - Good (random) passwords are almost impossible to remember
  - Password Reuse: re-use passwords between services
    - If one service is hacked, then someone knows your password for all other services

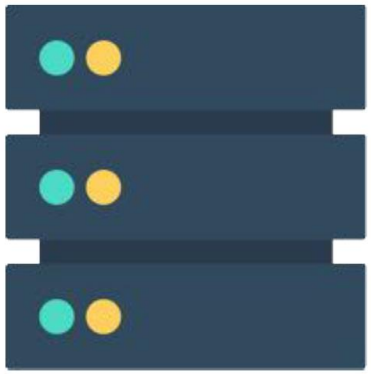
  - Vulnerable to key loggers
  - Vulnerable to phishing attacks

Phishing

User

login - foo
password - 1234

Fake service



Service

login - foo
password - 1234

How Much Password Complexity is Needed?

- In general, a password can be thought of as good if brute-forcing it would require the same amount of work as a 64-bit encryption key
- How do we measure brute-force complexity of a password?

Brute force Complexity for Passwords

- How many different 4 character, all lowercase passwords are there?

$$? \quad ? \quad ? \quad ?$$

$$26 * 26 * 26 * 26$$

$$= 26^4$$

Translating into Key Complexity

$$26^4 = 2^x$$

$$x = \log_2(26^4)$$

$$x = 18.8 \text{ bits}$$

$$26^4 \approx 2^{19}$$

So, a 4 character password of all lowercase letters has the same brute force complexity as a ~19 bit encryption key

Let's make it better!

- 8 characters, all lowercase
  - 26^8 -> 37.6 bits
- 8 characters, lower and uppercase
  - 52^8 -> 45.6 bits
- 8 characters, lower, upper and numbers
  - 62^8 -> 47.6 bits
- 8 character, lower, upper, numbers, special
  - 94^8 -> 52.4 bits

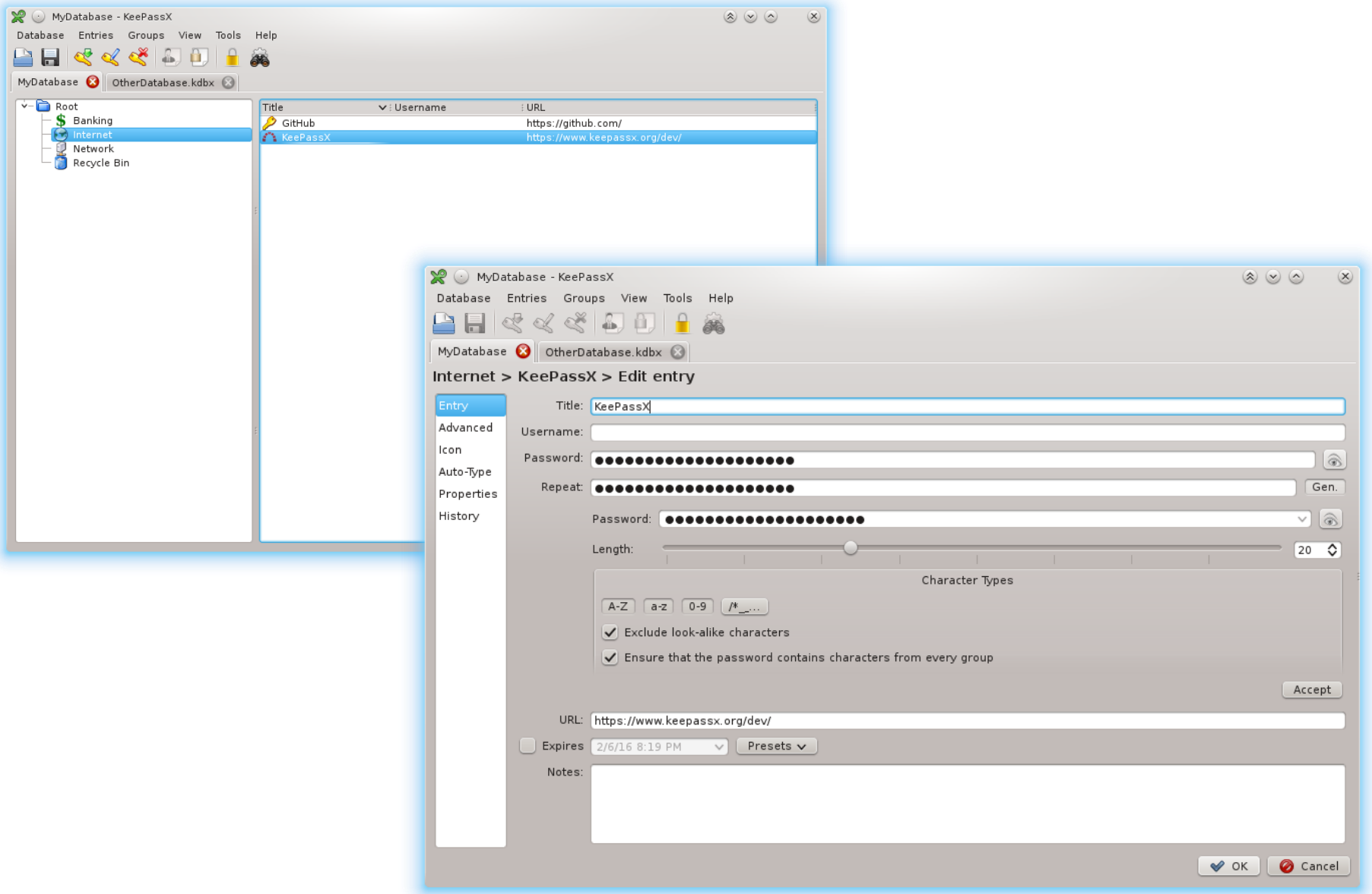
=> A randomly generated 8 character password with lowercase, uppercase, numbers, and special characters *is still not secure enough*

- How long would the password need to be for 64-bits of security?
  - 94^{10} -> 65.54 bits

Improvements - End-User Password Hygiene

- Use a different password for every account
- Make your passwords complex or randomly generated
- Use a password safe to generate and store passwords so that you don't need to remember them
  - All your passwords can be truly random!
- Open source password manager
  - http://www.keepassx.org/
- Commercial password managers
  - https://www.pcmag.com/article2/0,2817,2407168,00.asp

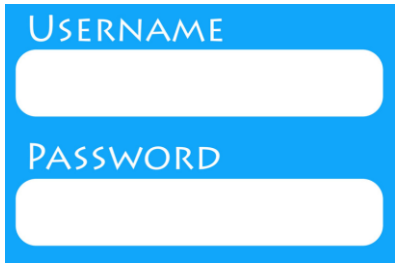
Password Safe



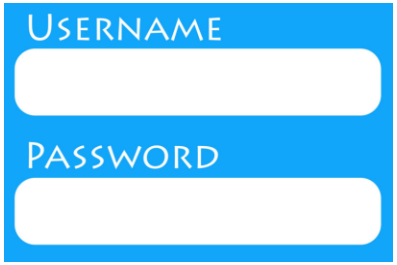
Improvements - Service Provider

- Lock the account after 3 unsuccessful attempts
- Use 2-Factor Authentication (2FA) as Single Factor Authentication is insufficiency

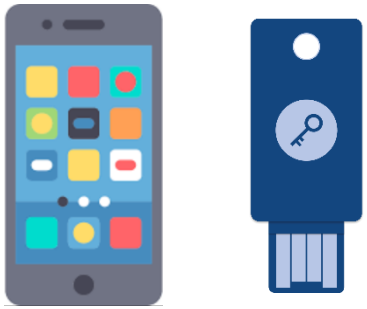
2-Factor Authentication (2FA)



Authentication Factors



Something the user knows
(PIN, password)



Something the user has
(mobile phone, device)

Something the user is
(biometric, retina, fingerprint)

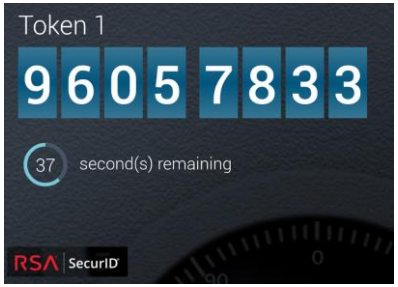
1. Something the user knows

- Information that would only be known by the user
- Examples:
  - Password (most common)
  - PIN number
  - Security questions (usually biographical information)

2. Something the user has

- A physical item the user has
- Examples
  - Smart phone
  - Smart card
  - Security token

- Security Token is Device or App that displays a number that changes ~every 30 seconds
  - Commonly used for corporate VPNs, generating one-time passwords (OTPs), etc.



- USB security key

3. Something the user is

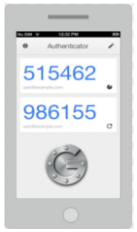
- *Biometrics*
  - Based on physical characteristics of the user
- Examples
  - Fingerprints and Retina/Iris scan used in Qatar for e-government and e-gate systems
- Not revokeable or transferable…
- But the use of biometric information is less common since fingerprint or retina recognition software is expensive and difficult to implement.

Multi-Factor Authentication

- Using multiple factors of authentication increases security
  - Must include *unique* factors
  - In practice 2-Factor Authentication (2FA) is used
- Example:
  - Requiring a password and a security question is still only one-factor (what the user knows)
  - Requiring a password and the code from an SMS sent your phone is two-factor (what the user knows and what the user has)
  - Requiring a password and fingerprint: *two-factor* authentication (what the user knows and what the user is)

2FA in Practice

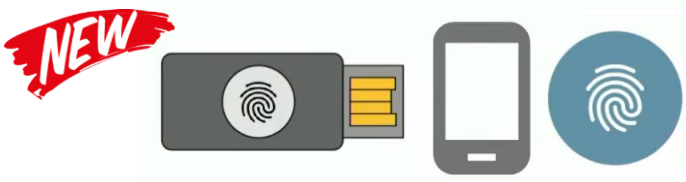
SMS



Time-based One-time Passwords (TOTP)
e.g. Google Authenticator

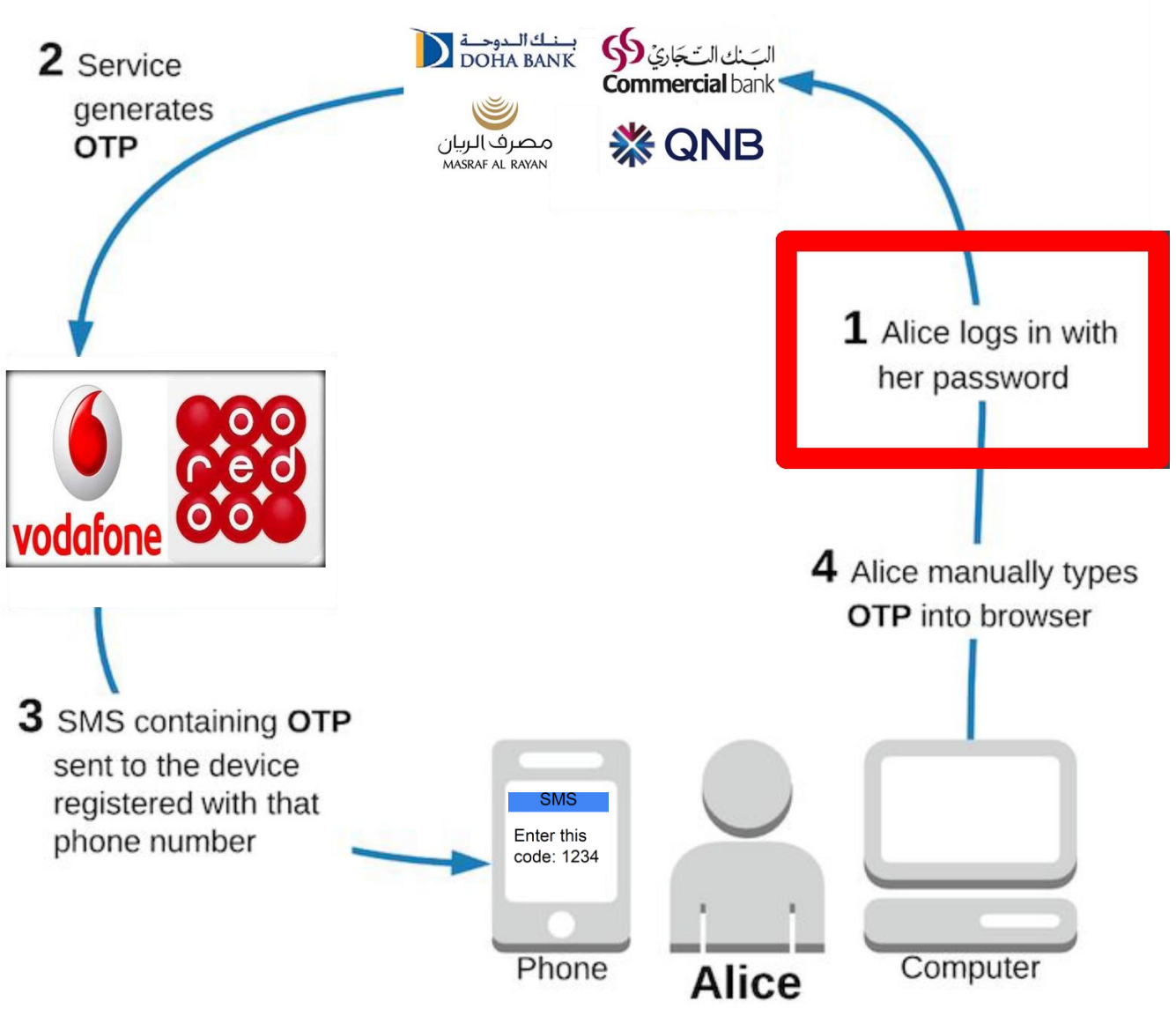
Push Notifications
e.g. Google Prompt

Universal 2nd Factor (U2F)
e.g. Yubico Security Key
(Public-Key Cryptography Device)



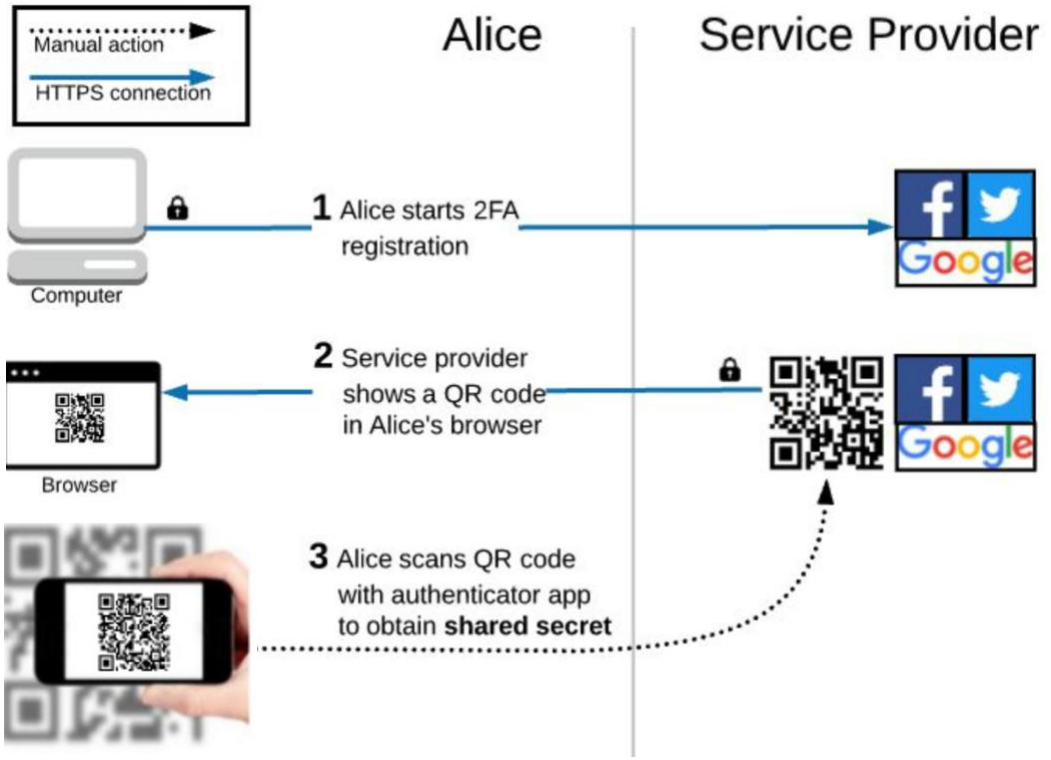
Web Authentication API
WebAuthn

SMS – 2FA Authentication Flow

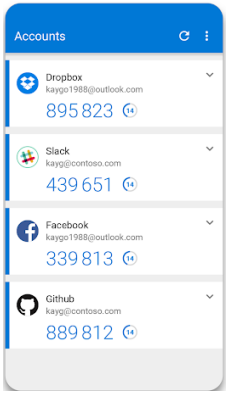


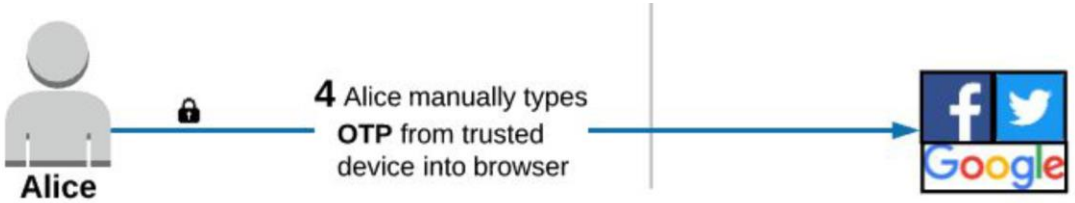
Time-based One-Time Password (TOTP)

Registration Flow



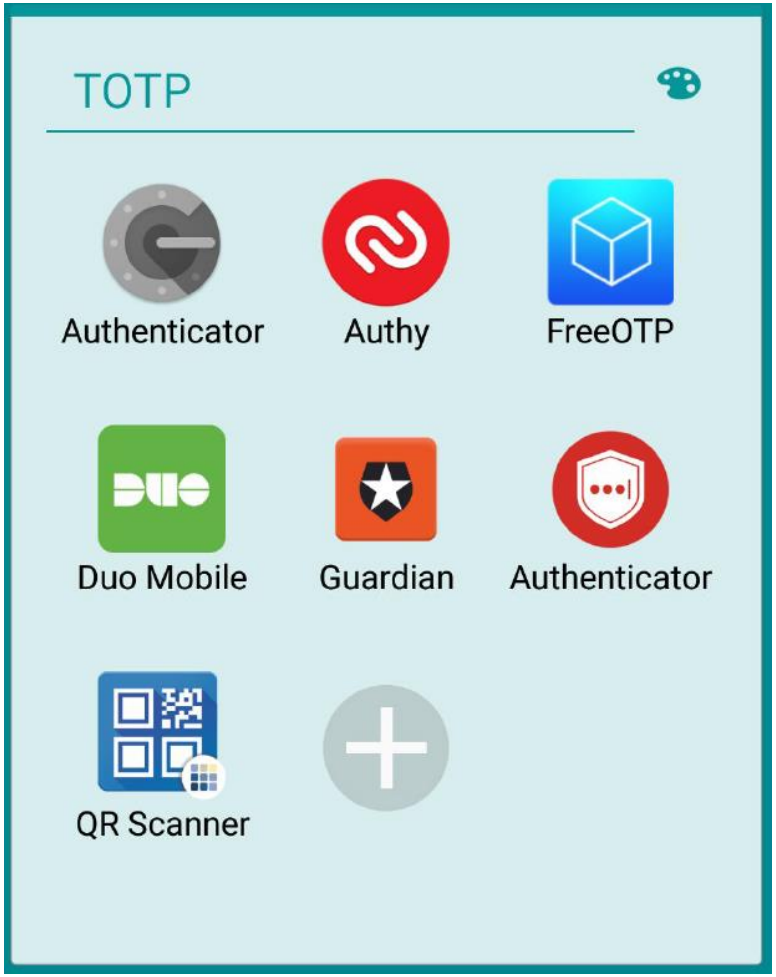
Authentication Flow

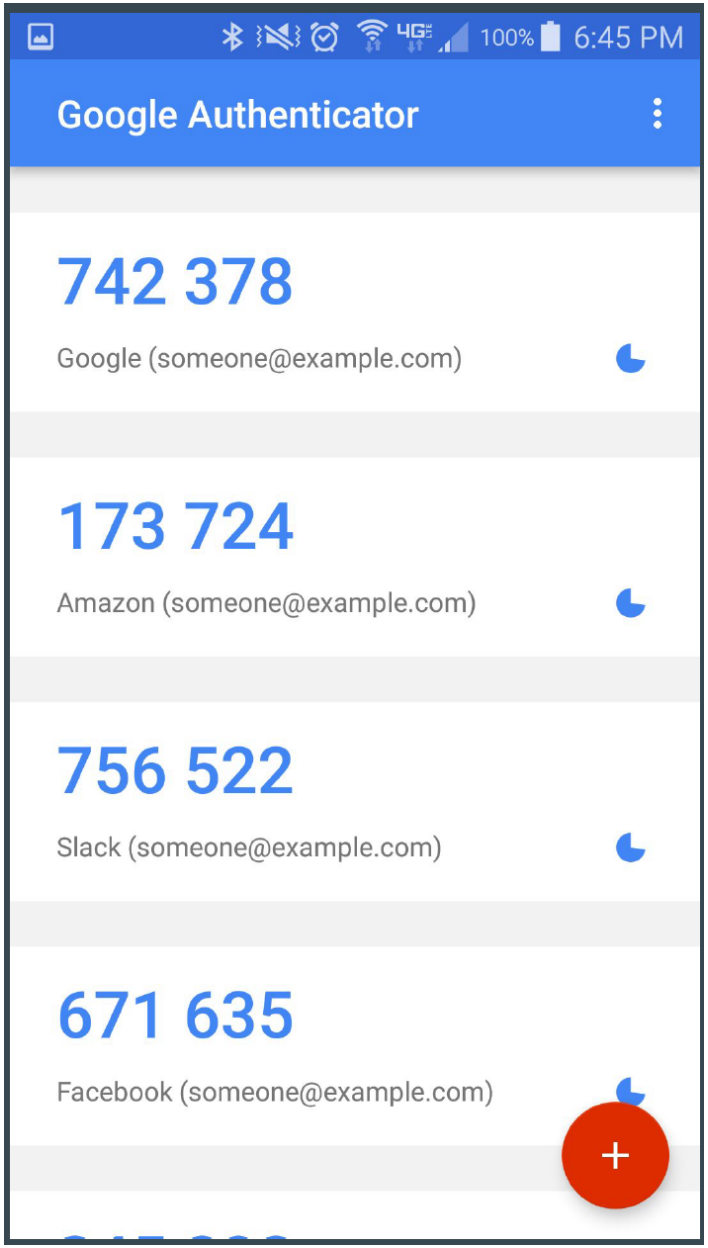




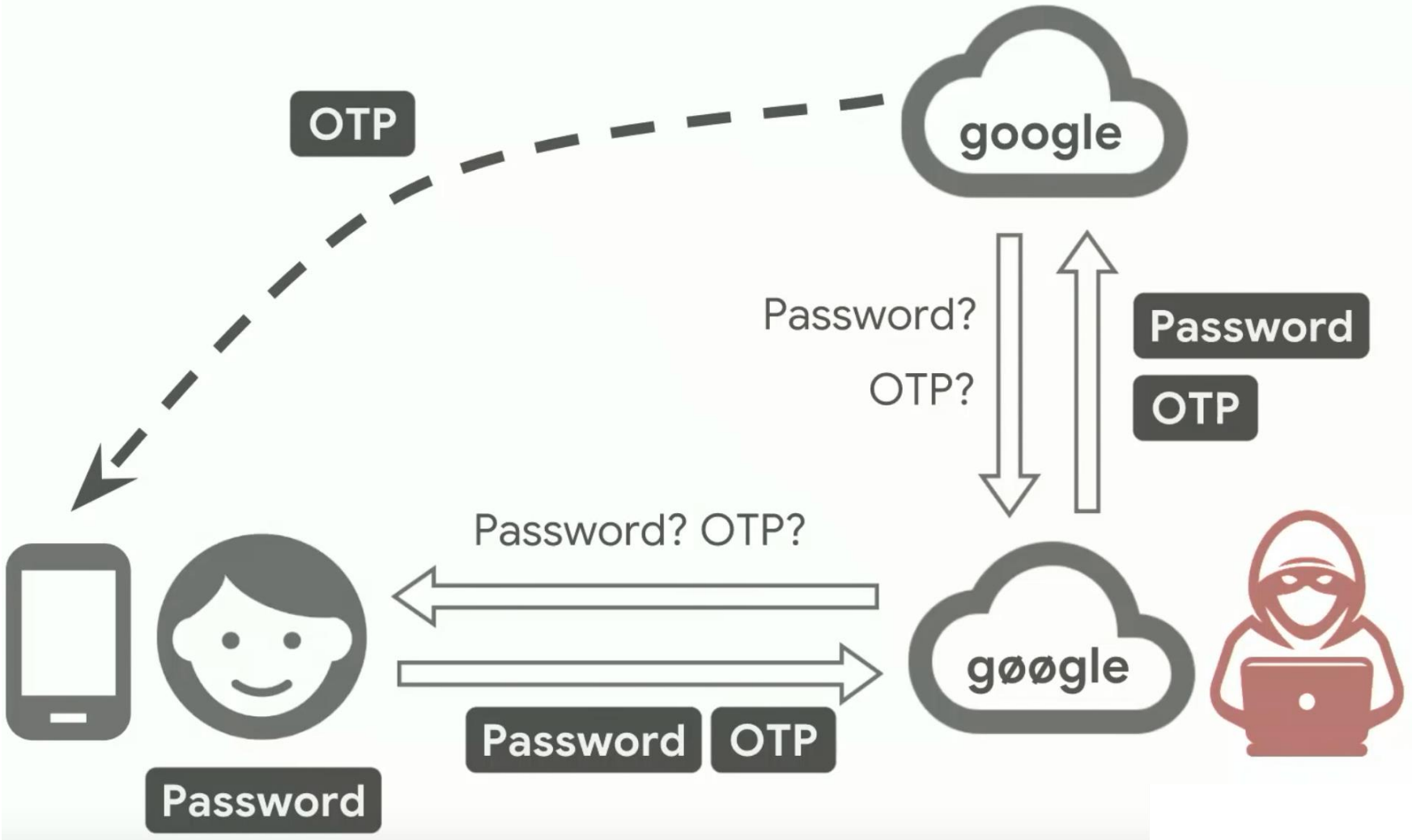
TOTP = HMAC-SHA-1 (shared secret + time)

TOTP: Example Authenticator Apps

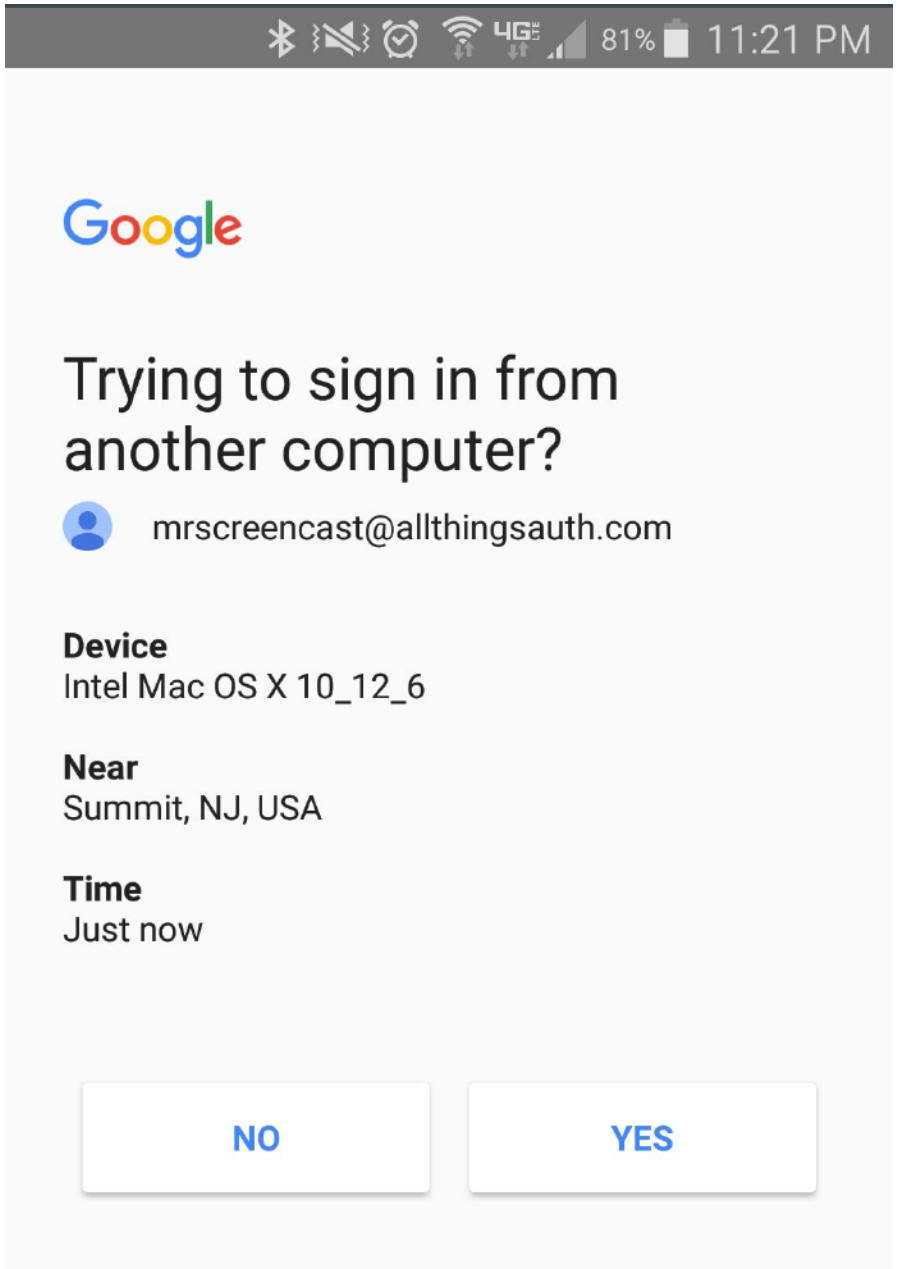




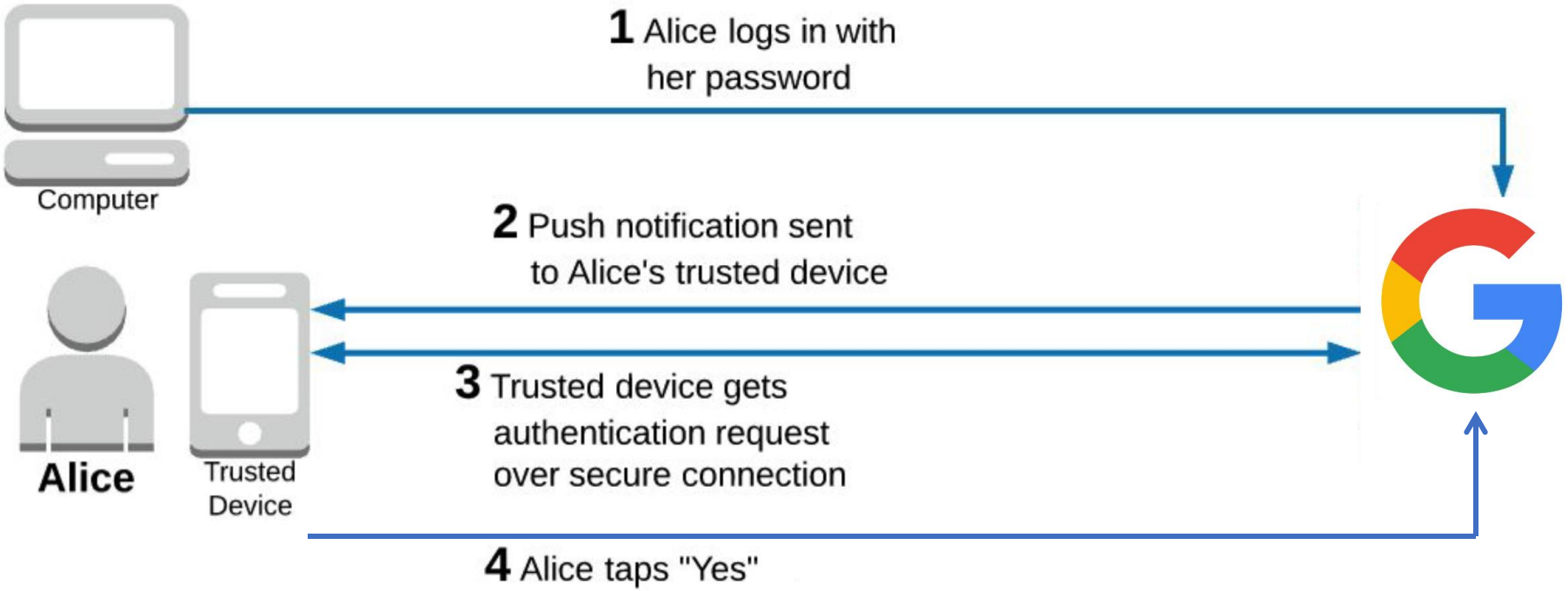
SMS and OTP are still vulnerable to Phishing Attacks



Push: Authentication prompt

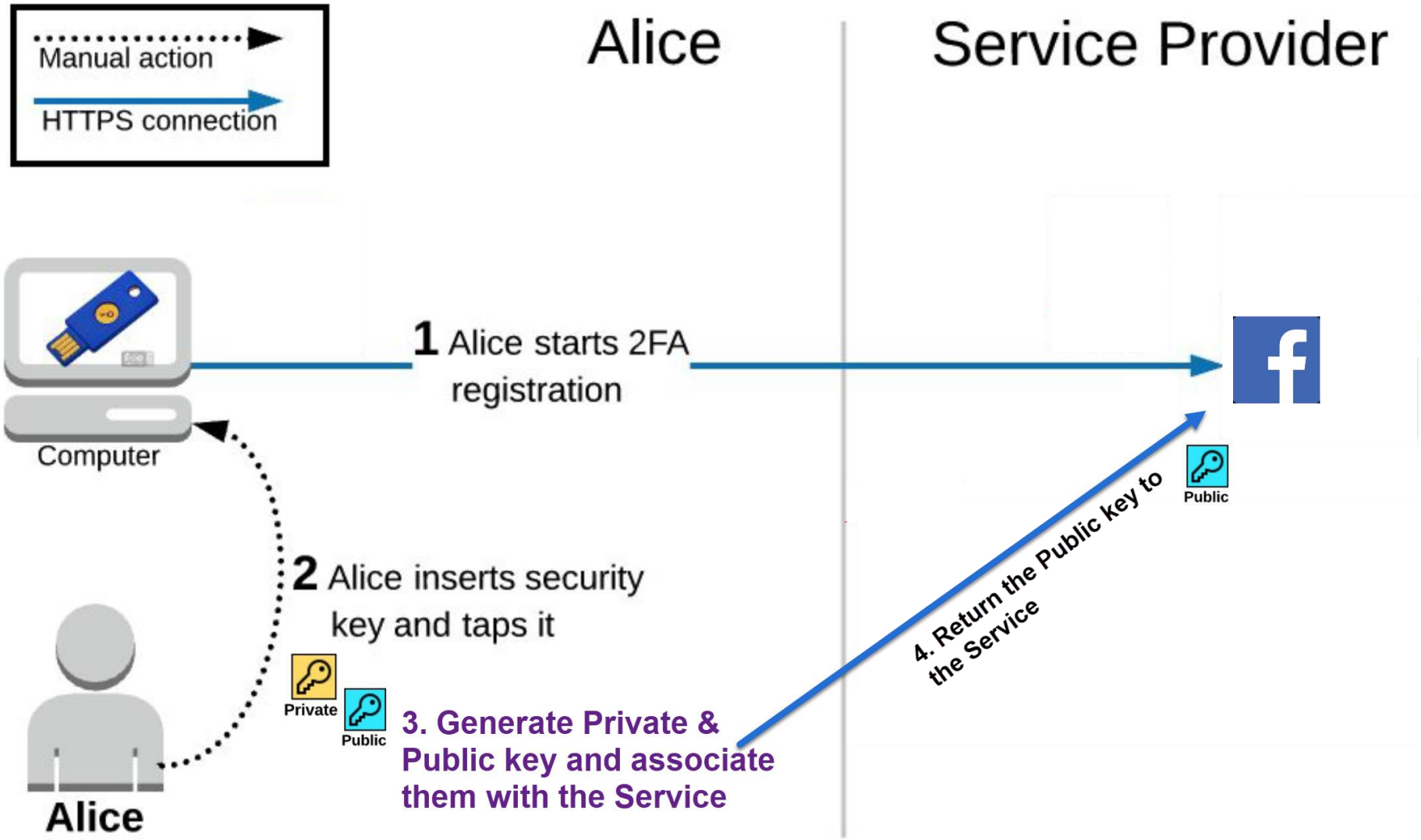


Push: Authentication flow

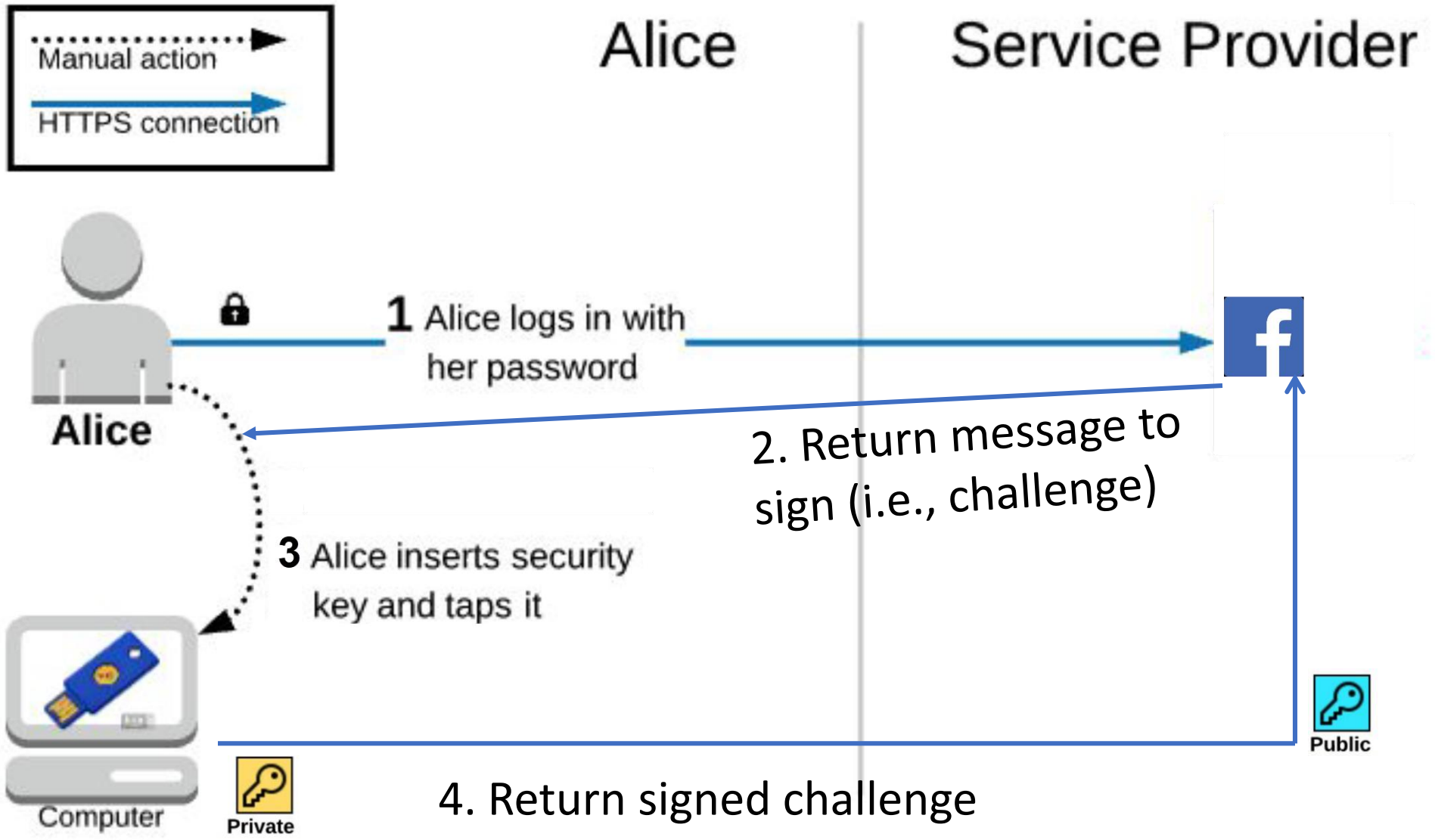


- Secure communication over HTTPS using public key cryptography
- Other solutions: Duo, Authy

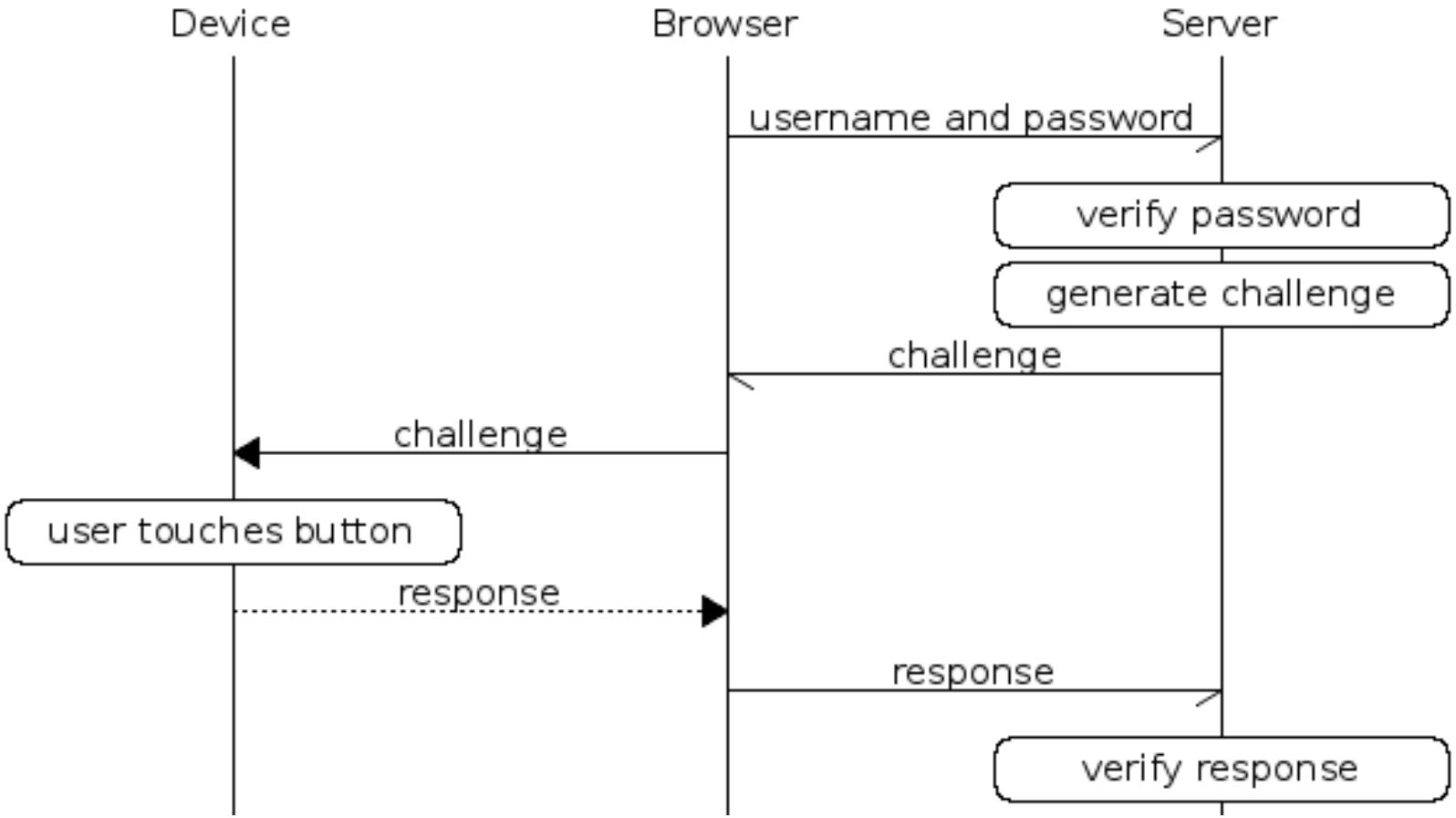
U2F: Registration flow



U2F: Authentication flow



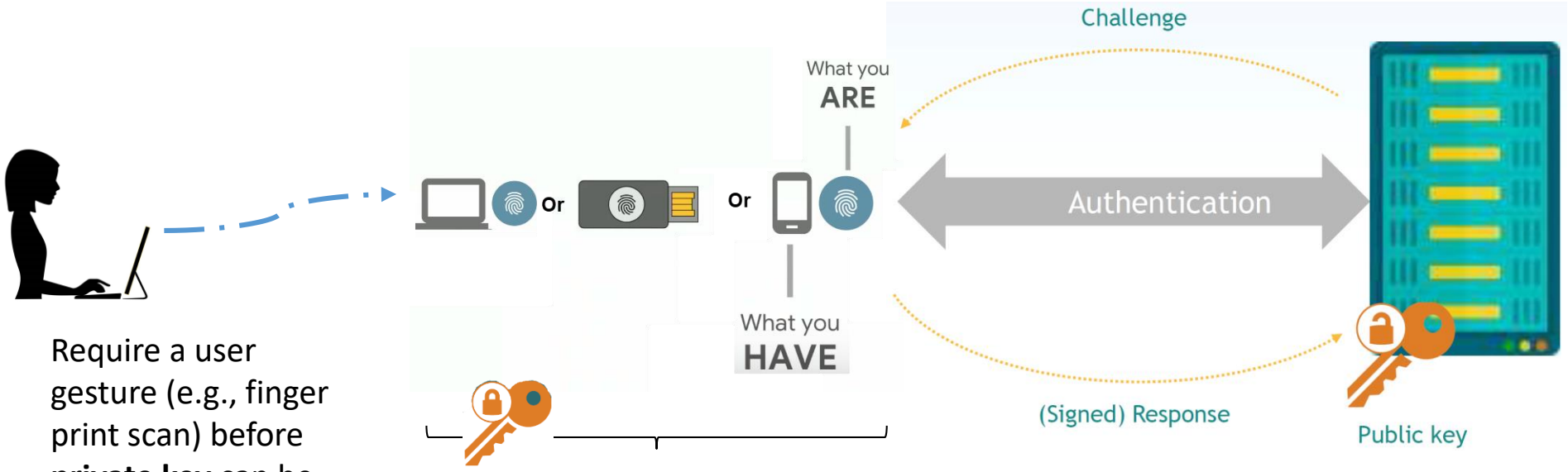
U2F: Authentication flow (simplified)



WebAuthn - Web Authentication API

WebAuthn uses Public key cryptography for **passwordless authentication**

- Unbreakable security & Great user experience
- Requires a one time setup flow to register an **Authenticator** with the user account
- Every time the user wants to authenticate they have to prove to the server that they possess the private key. This is done through a challenge-response based protocol.

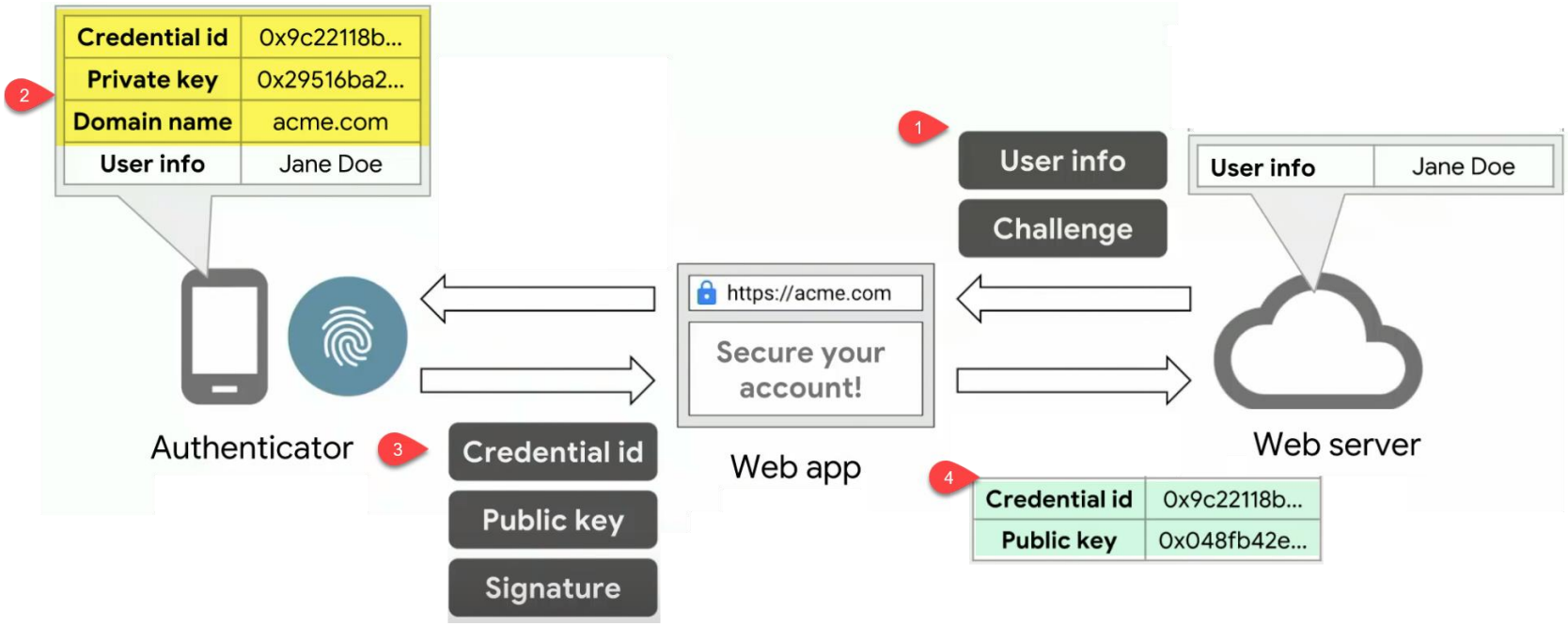


Require a user gesture (e.g., finger print scan) before **private key** can be used to sign the challenge

Authenticator

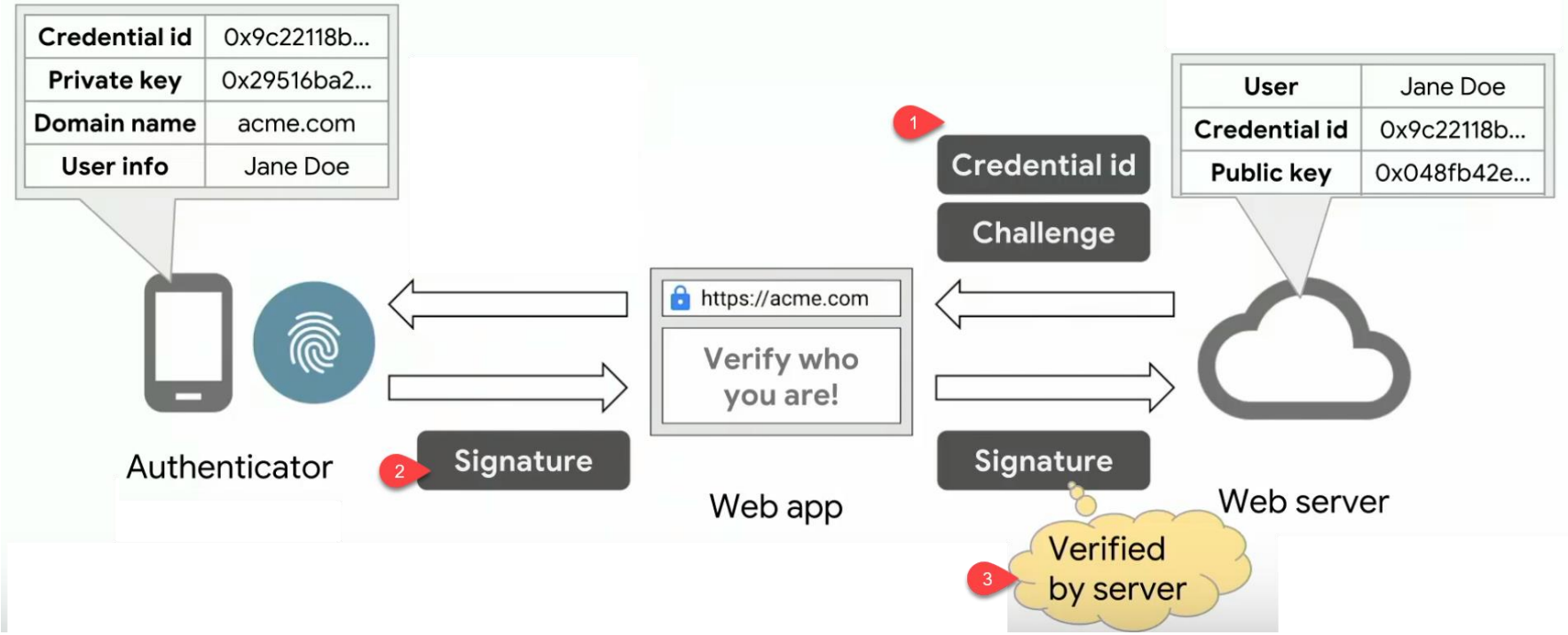
Responsible for Biometric User Verification at the client-side

WebAuthn - Registration Flow



1. The Web server first generates a challenge (i.e., a random number) and sends it along with the User info to the Web app running in the browser.
2. The Authenticator asks for the **user consent** (e.g., figure print scan) then generates a Public Key and Private Key pairs. It store the private key locally along with the Credential id, User info and the Domain name.
3. The Authenticator **signs** the Credential id, the **Public key** and the Challenge and transmits them to the Web server.
4. The server validates the signature then stores the Public key and the Credential id with the user account.

WebAuthn - Authentication Flow



- Every time the user wants to authenticate they have to prove to the server that they possess the private key. This is done through a challenge-response based protocol:

1. The Web server first generates a challenge (i.e., a random number) and sends it along with the Credential id to the Web app running in the browser.
2. The Authenticator asks for the **user to authenticate** (e.g., figure print scan). Then it uses the private key to sign the challenge and send the signature to the server.

   The finger print never leaves the device, it is only used locally to verify the user.

1. The server validates the signature using the user's public key.

Single Sign-On (SSO) using Token-Based Security

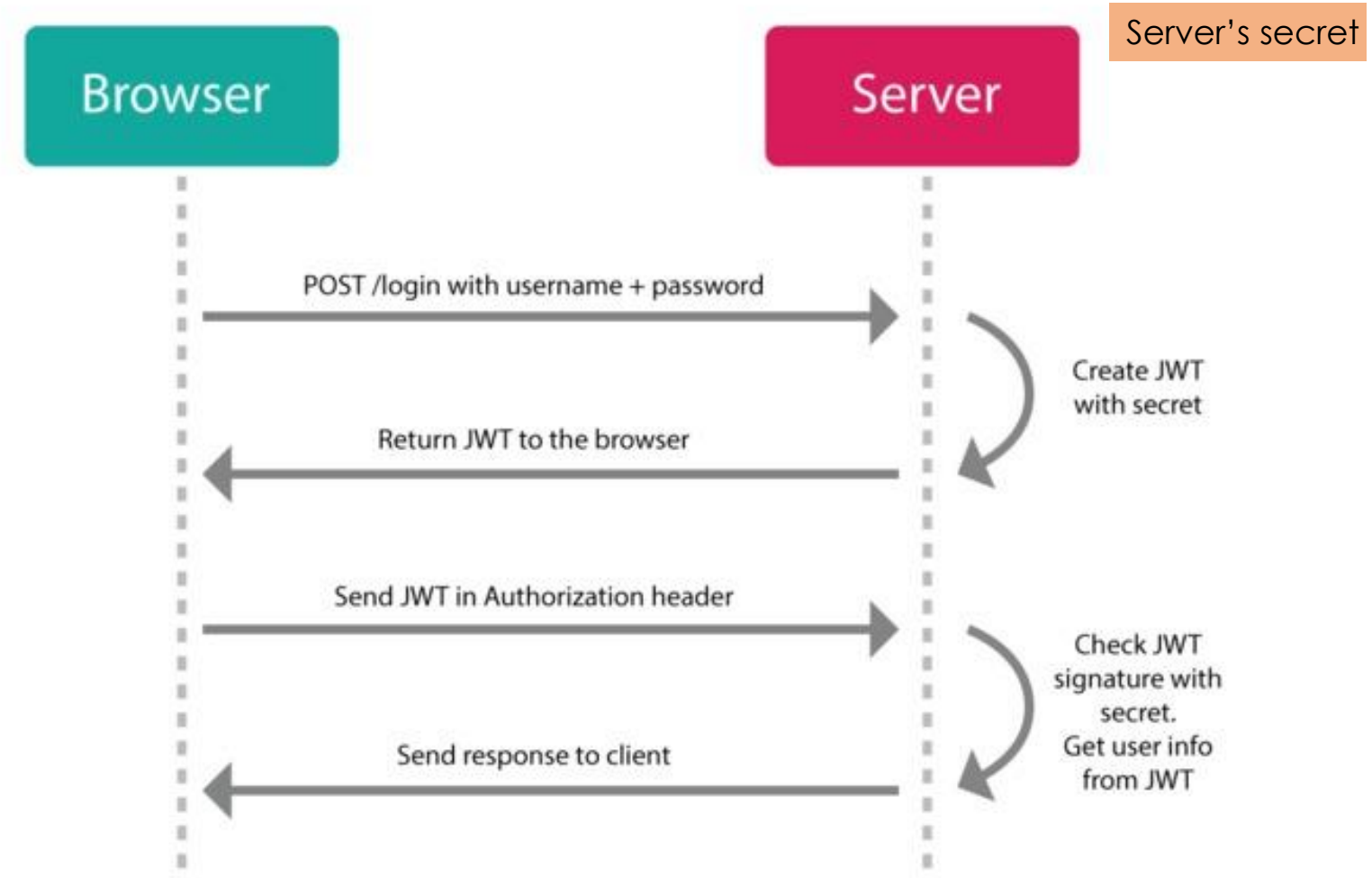
Token-based Single Sign-On (SSO)

- After a successful authentication a **token** is issued by the server and communicated to the client
- JSON Web Token (JWT) is a widely used token format. It is a **signed json object**
  - Contains claims (i.e., information about the issuer and the user, e.g., user details and their role)
  - Signed (tamper proof & authenticity)
  - Typically contain an expiration time
- JWT is added to the HTTP header of subsequent requests to other Web resources
  - The resource validates the token to authenticate the user (without asking for username and password again)

Token-based Single Sign-On (SSO)

- JWT enables SSO by sharing the JWT between different applications
  - Web App gets a request that includes a JWT token
  - It checks that the JWT token is valid
  - It uses info in the token to make an access control decision

JSON Web Token (JWT)



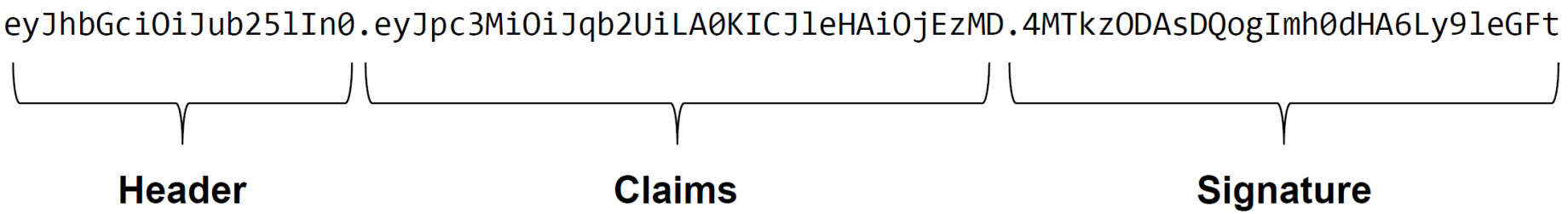
JWT Structure

Header

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

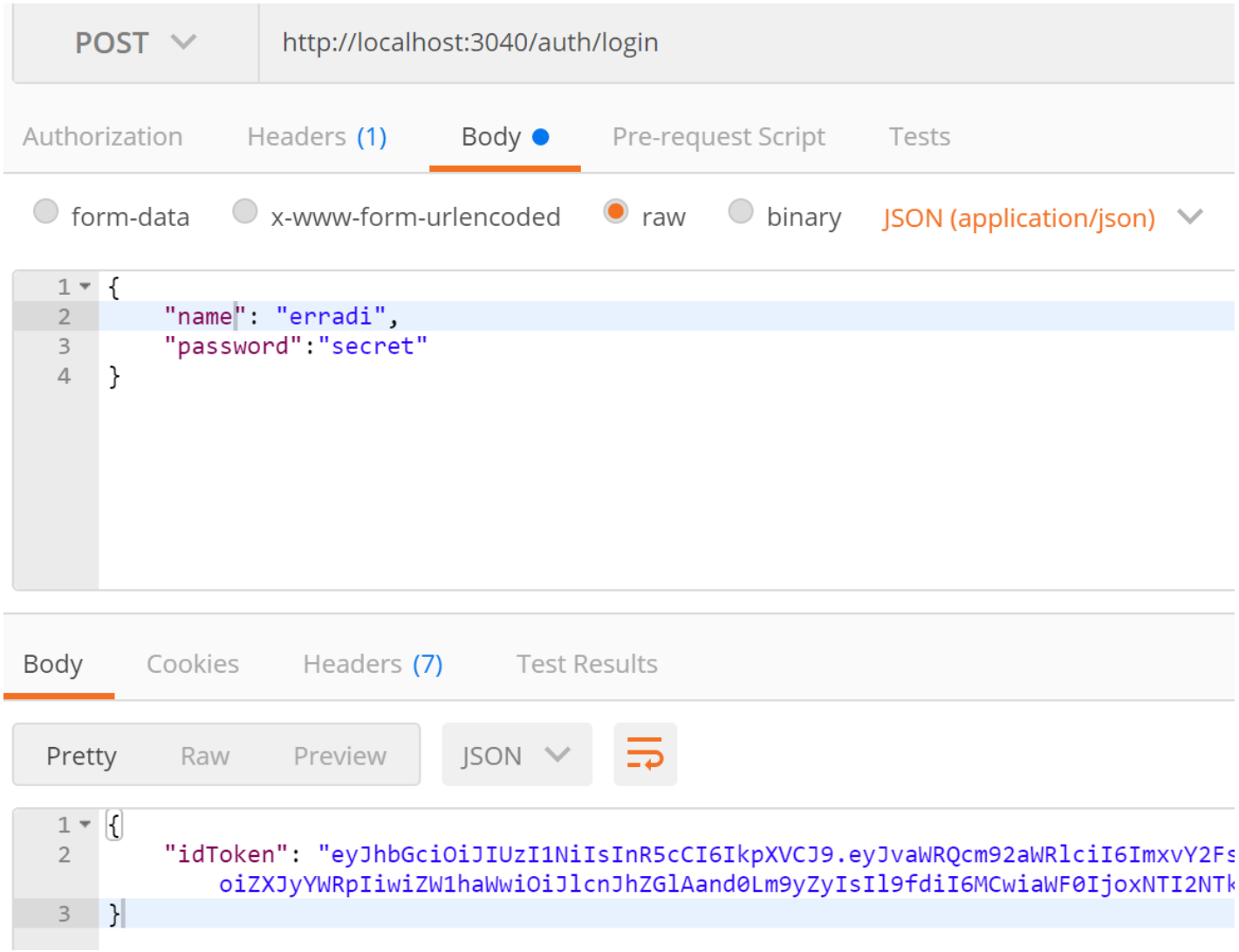
Claims

```
{
    role: "Admin",
    given_name: "Abdelkarim",
    family_name: "Erradi",
    name: "erradi",
    email: "erradi@jwt.org",
    iat: 1526597430,
    exp: 1526604630
}
```



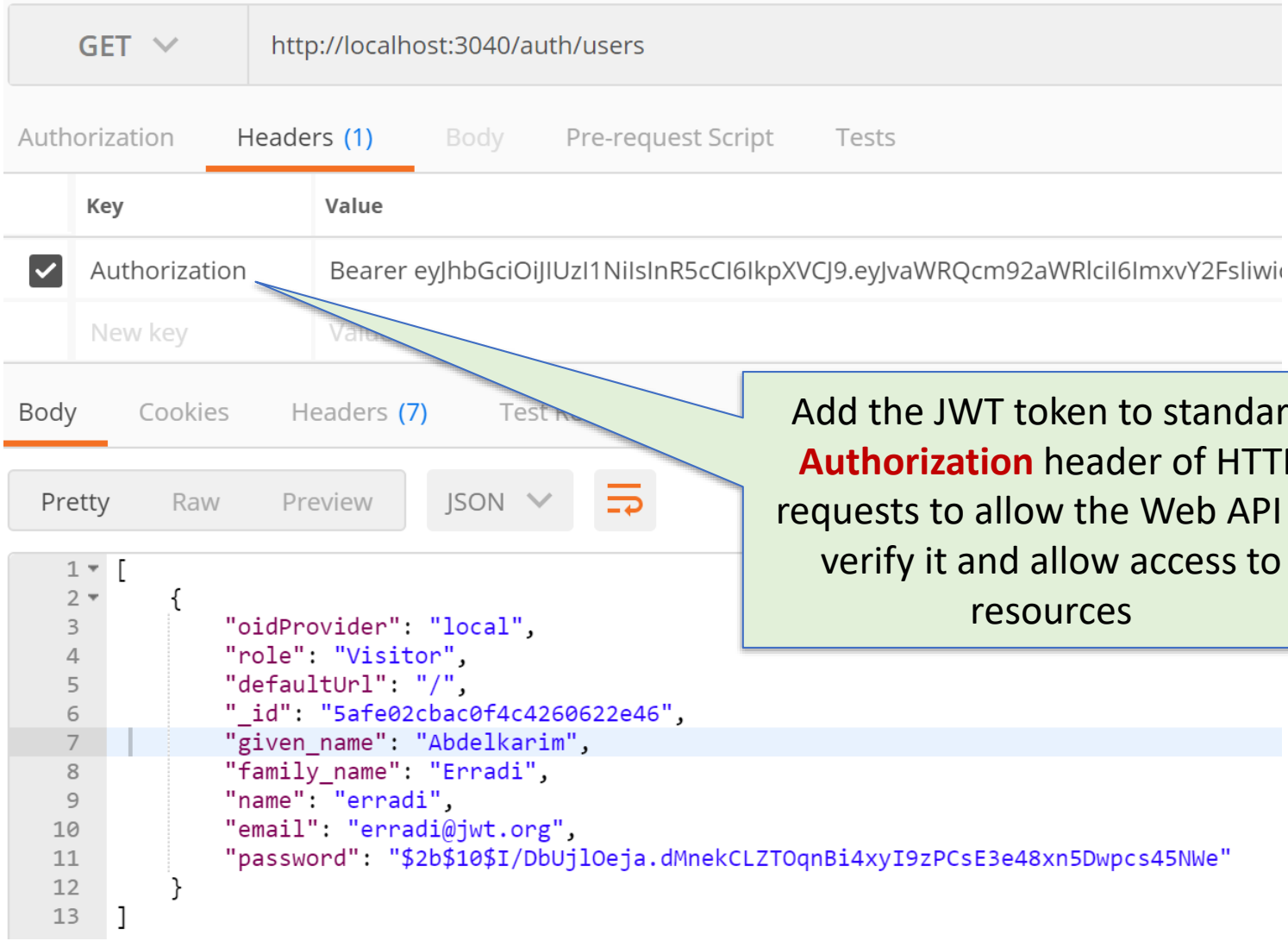
JWT Issued after Successful Login

- Example sign in http://localhost:3040/auth/login



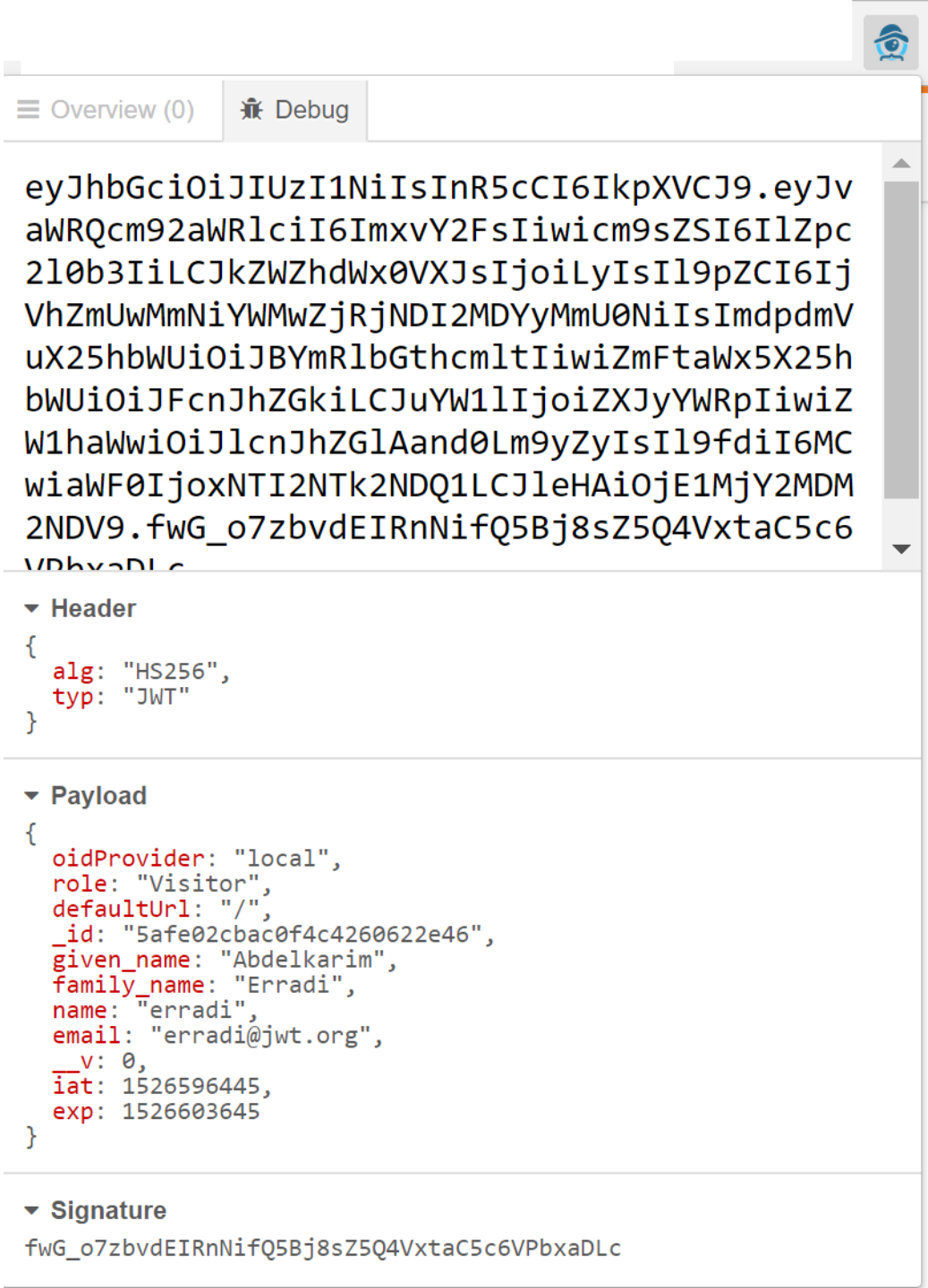
Use JWT to Access Protected Resource

- Get users http://localhost:3040/auth/users



https://jwtinspector.io/

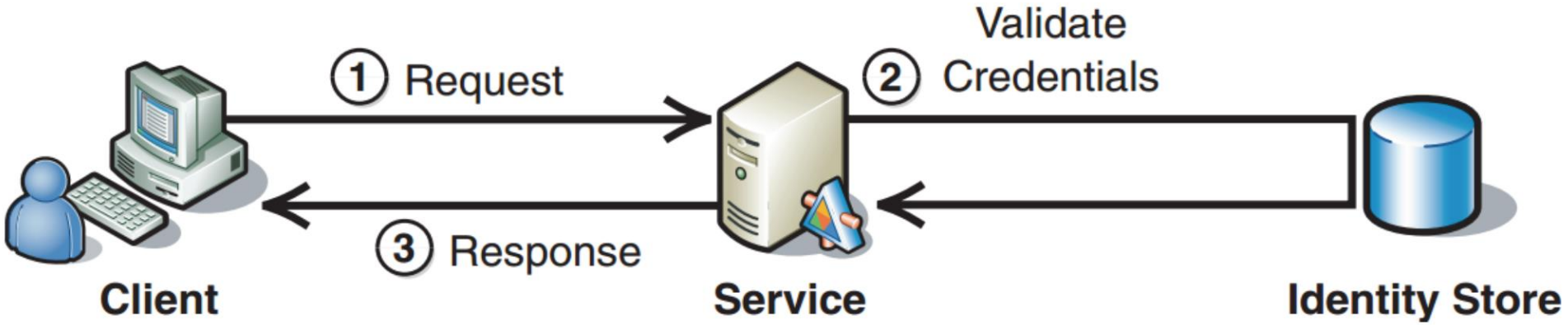
JWT Inspector is a chrome extension that lets you **decode** and **inspect** JWT in requests, and local storage

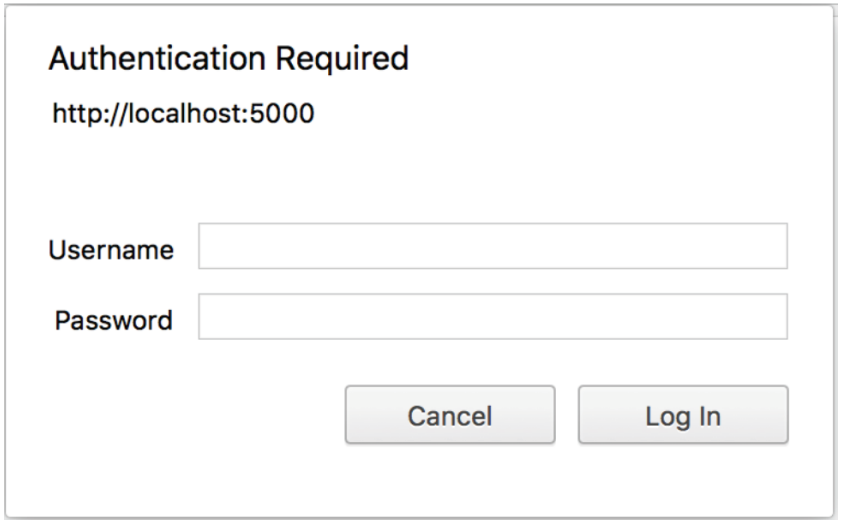


Delegated Authentication

Direct Authentication

- The App implements its own authentication and the **server maintains the user identity**.



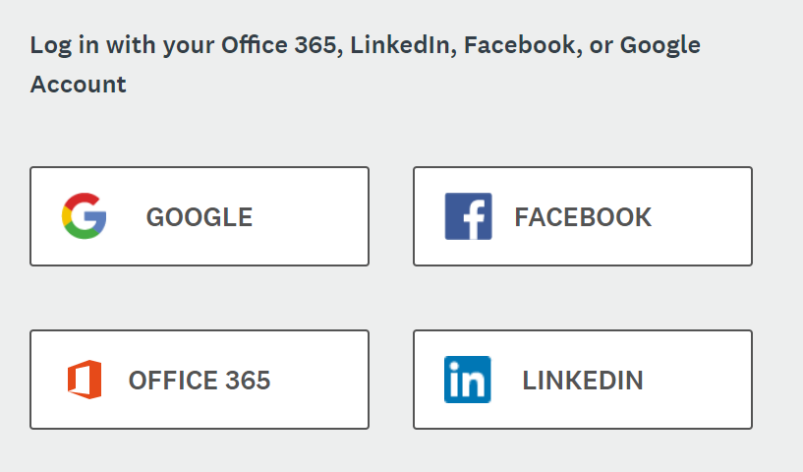


Authentication is hard

- Trying to write your own login system is difficult:
  - Need to save passwords securely
  - Provide recovery of forgotten passwords
  - Make sure users set a good password
  - Detect logins from suspecious locations or new devices
  - etc.
- Luckily, **you don't have to build your own authentication!**
- You can use **OpenID Connect** to delegate login to an **Identity Provider** and get the user's profile

OpenID Connect

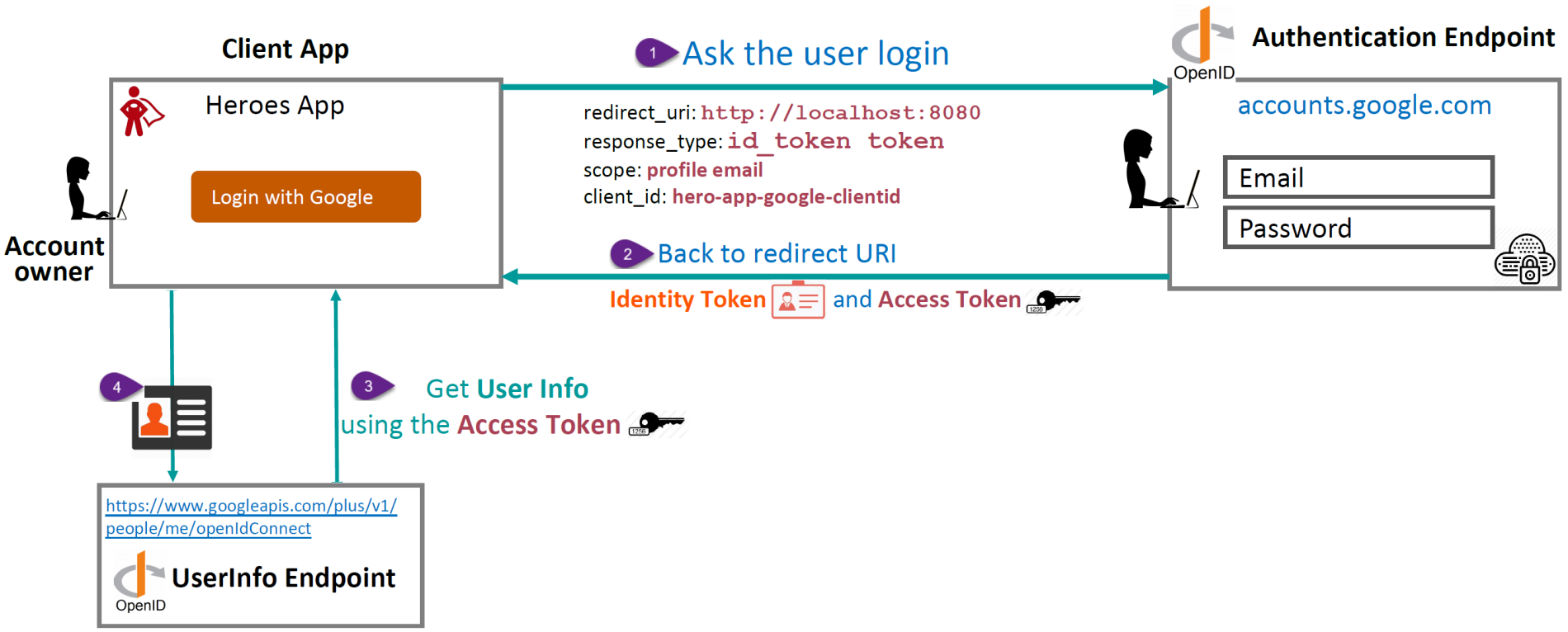
- **OpenID Connect** is a standard for user authentication
  - For users:
    - It allows a user to log into a website like AirBnB via some other service, like Google or Facebook
  - For developers:
    - It lets you authenticate a user without having to implement log in
  - Examples: "Log in with Facebook"



OpenID Connect APIs

- Companies like Google, Facebook, Twitter, and GitHub offer OpenID Connect APIs:
  - Google Sign-in API
  - Facebook Login API
  - Twitter Login API
  - GitHub Apps/Integrations
  - OpenID Connect is standardized, but the API that these services provide are slightly different
  - You must read the documentation to understand how to connect via their API
- After the user logins, you will get the user profile such name, email, etc

OpenID Connect Authentication Flow



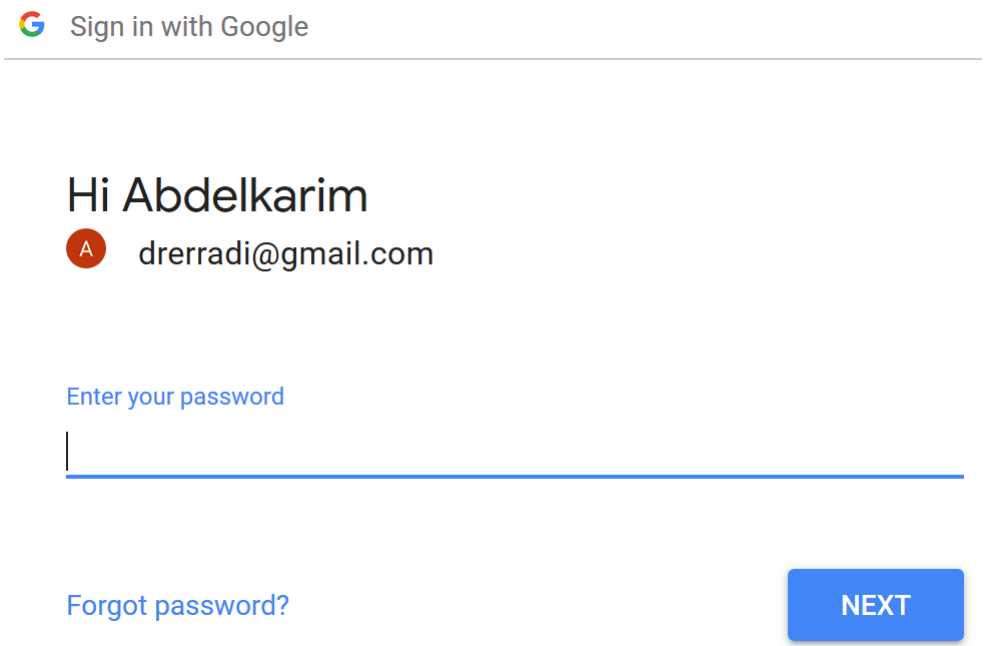
Authenticating via a SPA App

- **User** starts the flow by visiting a Web App
- **Client** sends authentication request with *profile* scope via browser redirect to the **Authorization** endpoint
- **User** authenticates and consents to **Client** to access user's identity
- **ID Token** and **Access Token** is returned to **Client** via browser redirect
- **Client** optionally fetches additional user info with the **Access Token** from **UserInfo** endpoint

Authorization Request

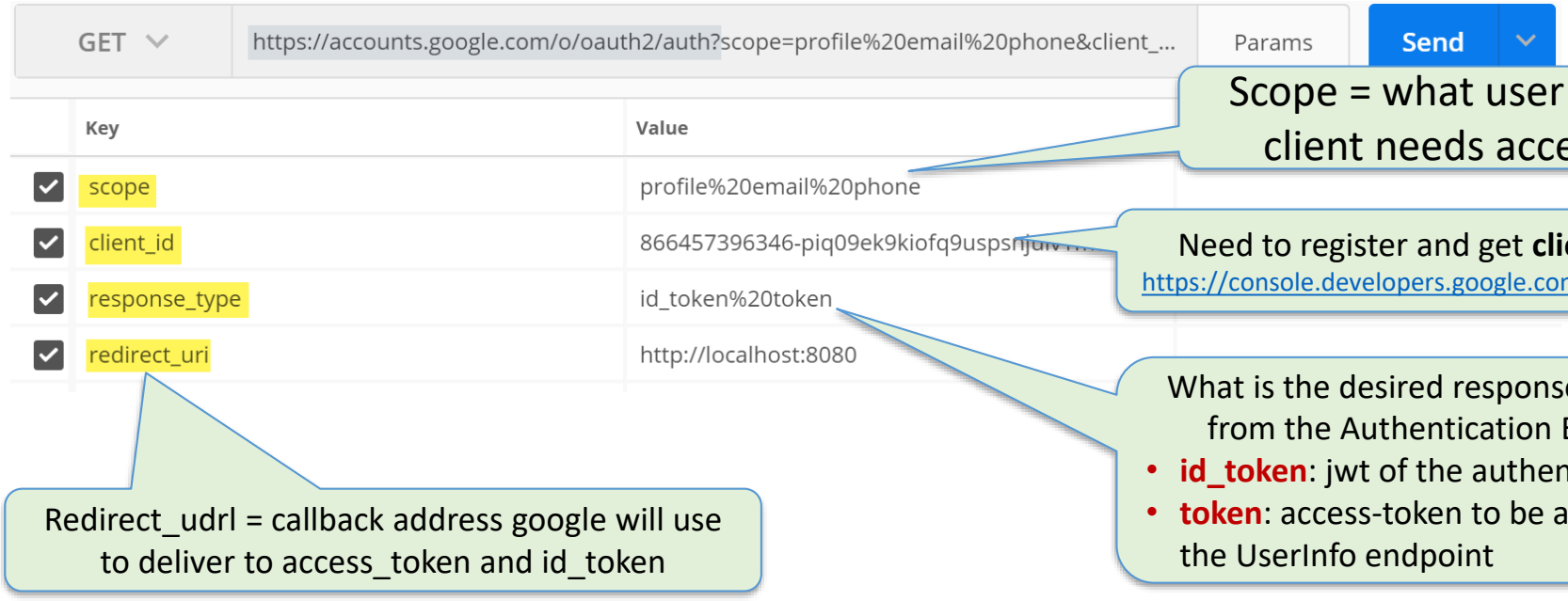
- Ask the user to login via browser redirect to the Authentication Endpoint

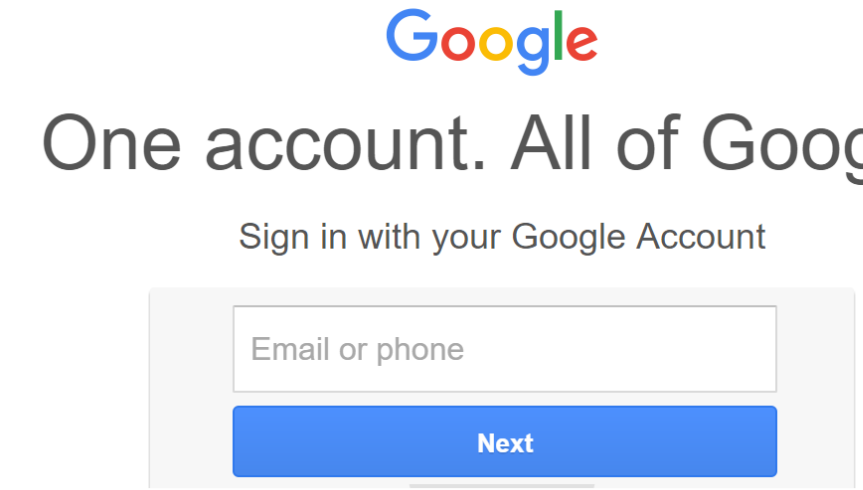
https://accounts.google.com/o/oauth2/auth



- This will return an **Access Token** to the client to allow it to request the user's profile from the UserInfo Endpoint

Authentication Parameters





ID Token

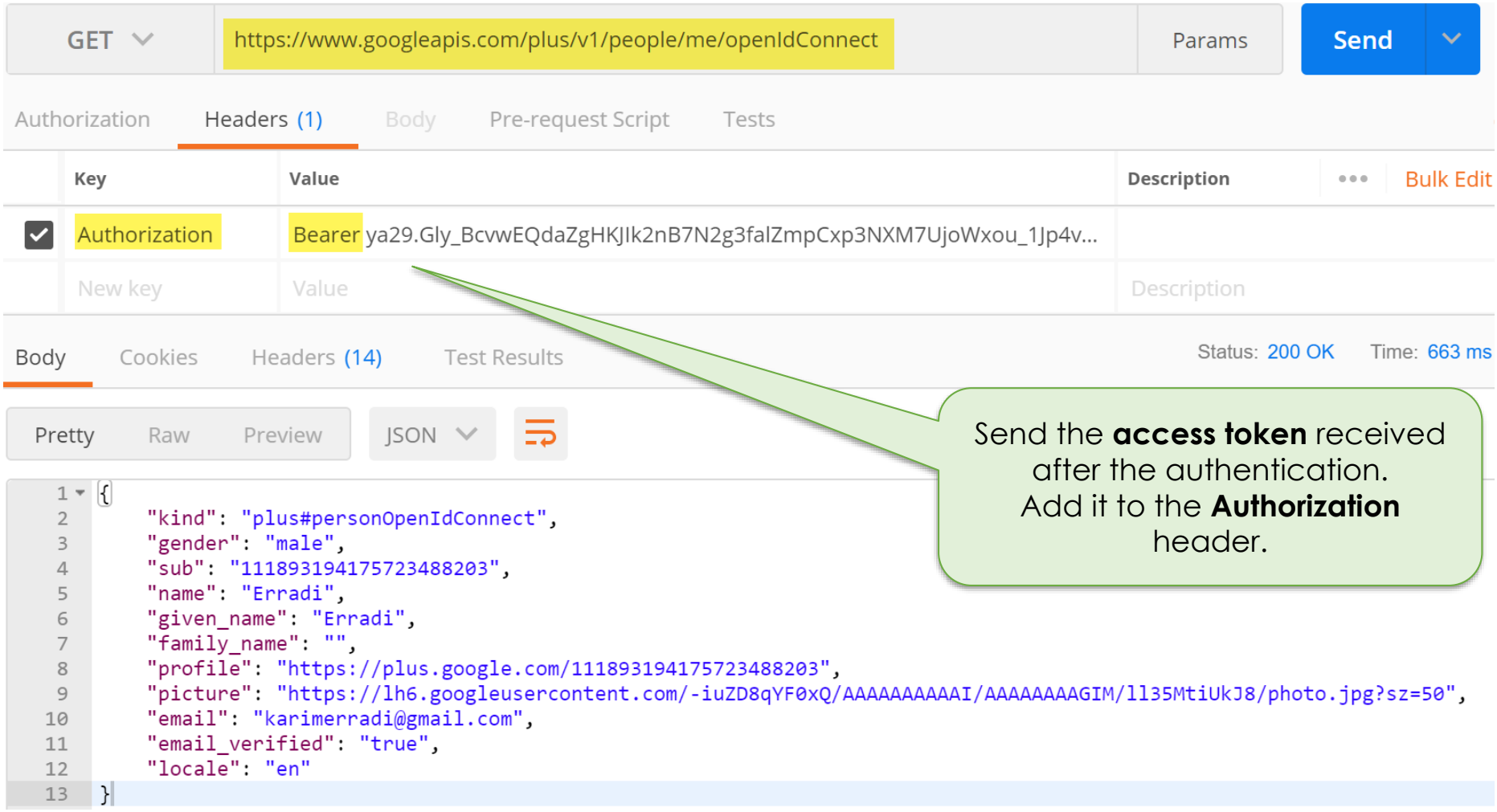
Example **ID Token** from Google

```
{
  iss: "accounts.google.com",
  aud: "1v1muk.apps.googleusercontent.com",
  sub: "111893194175723488203",
  email: "karimerradi@gmail.com",
  email_verified: true,
  exp: 1526656174,
  iat: 1526652574
}
```

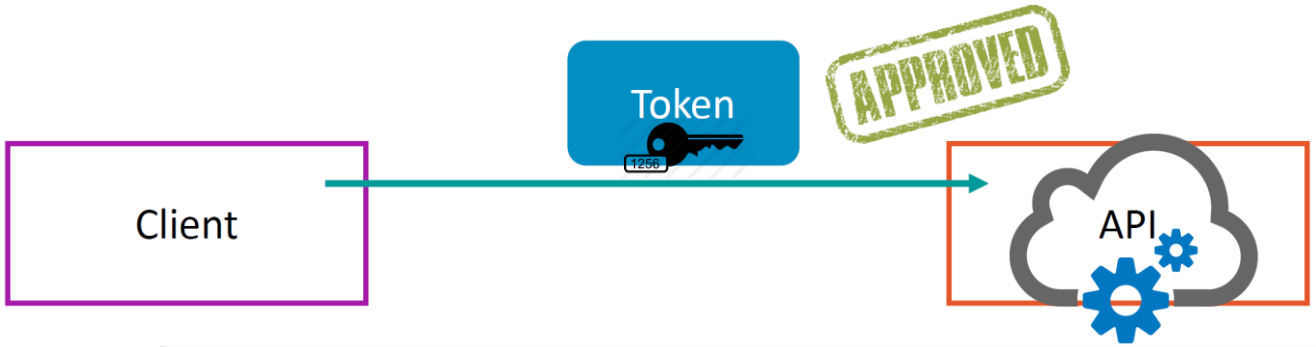
- JWT representing logged-in user
- Claims:
  - `iss` - Issuer
  - `sub` - User Identifier
  - `aud` - Audience for ID Token
  - `exp` - Expiration time
  - `iat` - Time token was issued

Calling the UserInfo Endpoint

- Get the user's profile from the UserInfo Endpoint



Use the Access Token to access Web Resources



- Validate token
- Grant access to the resource

GET https://people.googleapis.com/v1/people/me/connections?personFields=names,emailAddresses,phoneNum... Params Send

Key	Value	Description
personFields	names,emailAddresses,phoneNumbers,addresses,pho...	

Authorization Headers (1) Body Pre-request Script Tests

Key	Value	Description
Authorization	Bearer ya29.GlvABUS5jPiHTOywJTpKr6cCwUEsACshu3...	

Body Cookies (1) Headers (13) Test Results Status: 200 OK Time: 253 ms

Pretty Raw Preview JSON


```
{
    "connections": [
        {
            "resourceName": "people/c3945308633452445077",
            "etag": "%EggBAgMJCxA3LhoMAQIDBAUGBwgJCgsMIgxuWmJUQU5BT3FKOD0=",
            "names": [
                {
                    "metadata": {
                        "primary": true,
                        "source": {
                            "type": "CONTACT",
                            "id": "36c08d4c8a36fd95"
                        }
                    },
                    "displayName": "Qatar University",
                    "familyName": "University",
                    "givenName": "Qatar",
                    "displayNameLastFirst": "University, Qatar"
                }
            ],
```

Summary

- Three types of authentication factors
  1. What the user knows
  2. What the user has
  3. What the user is
- 2 Factors Authentication (2FA) is better
- 

Web Authentication API (WebAuthn) uses Public key cryptography for passwordless authentication
- JWT is easy to create, transmit and validate to protect Web Apps in a scalable way
- Use OpenID Connect for **Delegated Authentication**

Resources

- NIST Digital Identity Guidelines

https://pages.nist.gov/800-63-3/

- Web Authentication API (WebAuthn)

https://en.wikipedia.org/wiki/WebAuthn

- JWT Handbook

https://auth0.com/resources/ebooks/jwt-handbook

- Authentication Survival Guide

https://auth0.com/resources/ebooks/authentication-survival-guide

- What the Heck is OpenID Connect?

https://www.youtube.com/watch?v=6ypYXxRPKgk