Web Authentication

HTTP Authentication and Security Considerations

University of Fribourg
Department of Informatics
Software Engineering Group

October 15, 2020

Outline

1. Introduction
2. HTTP Basic
3. HTTP Digest
4. HMAC authentication
5. Token authentication
6. JSON Web Token (JWT)
7. OAuth2
8. Multi-factor authentication
9. Web Authentication (WebAuthn)
10. OpenID Connect

Authentication vs. authorization

- **Authentication** is the process of ascertaining that somebody really is who he claims to be.
- **Authorization** refers to rules that determine who is allowed to do what. E.g. Alice may be authorized to create and delete records, while Bob is only authorized to read.

```
1 @api.login_required()
2 @api.permission_required(
3         permissions.OwnerRolePermission(Account)
4 )
5 def get(self, args):
6         """
7         Return account balance
8         """
```

Cryptographic primitives

Security Goal	Cryptographic primitive		
	Hash	**MAC**	**Digital signature**
Integrity	✔	✔	✔
Authentication	✘	✔	✔
Non-repudiation	✘	✘	✔
Kind of keys	none	symmetric keys	asymmetric keys

- Integrity
  - The message has not been accidentally modified
- Authentication
  - The message originates from the sender
- Non-repudiation
  - A third party can be confident that the message originates from the sender

HTTP Basic

https://tools.ietf.org/html/rfc2617

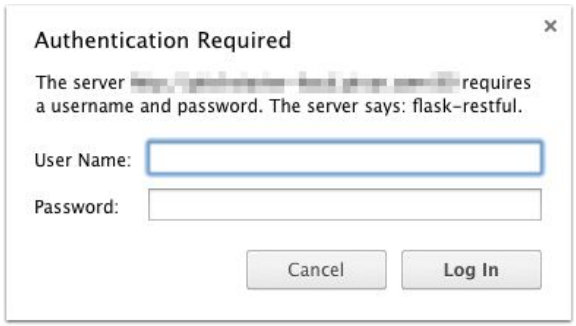
- Simplest authentication
- The server responds 401 Unauthorized and asks for authentication in headers

```
1 HTTP/1.1 401 Unauthorized
2 WWW-Authenticate: Basic realm="flask-restful"
```

- Client adds header with ‘user:password’ encoded in base64

```
1 GET / HTTP/1.1
2 Host: example.org
3 Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

- Handled natively by browser
  - cannot customize dialog box
- Vulnerable against eavesdropping if not using SSL/TLS
  - user and password sent unencrypted



HTTP Digest I

https://tools.ietf.org/html/rfc2617

- Secure alternative to HTTP Basic
  - Challenge-response based
  - More complex
  - Still handled natively by browsers
- The server answers 401 Unauthorized and asks for authentication in headers

```
1 HTTP/1.1 401 Unauthorized
2 WWW-Authenticate: Digest realm="flask-restful",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093"
```

- Client add authorization header

```
1 GET / HTTP/1.1
2 Host: example.org
3 Authorization: Digest <response>
```

HTTP Digest II

https://tools.ietf.org/html/rfc2617

- Simplest case
  - HA1 = MD5(username:realm:password)
  - HA2 = MD5(method:digestURI)
  - response = MD5(HA1:nonce:HA2)
- Optional security enhancements
  - quality of protection (qop)
  - nonce counter incremented by client
  - Client-generated random nonce
- A bit outdated
  - Only prevent replay attacks
    - Cannot prevent security downgrading
  - Was defined before Keyed-Hash Message Authentication Code (HMAC)

HMAC Authentication I

Keyed-Hash Message Authentication Code

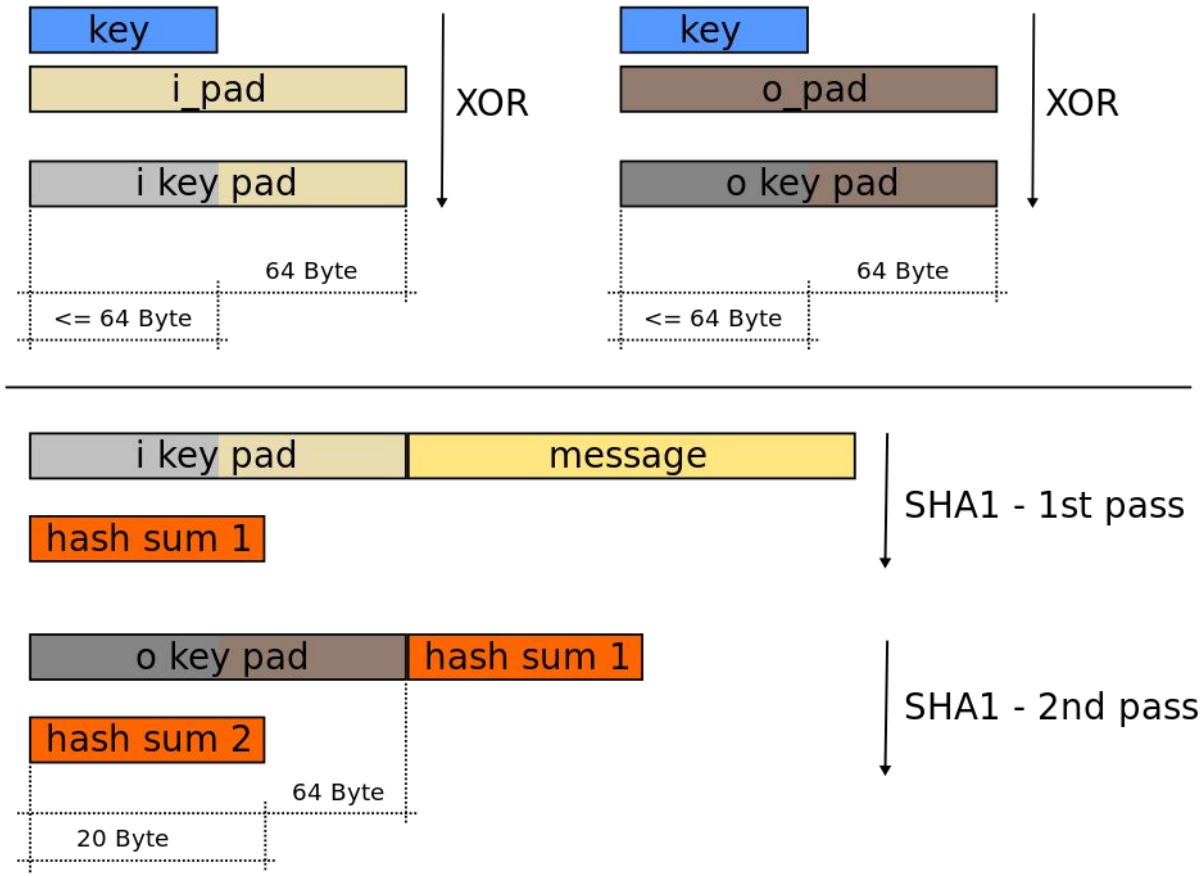
- Custom Authorization header content
  - digest = b64encode(hmac(“sha256”, “secret”, “GET+/users/johndoe/financialrecords”))

```
GET /users/johndoe/financialrecords HTTP/1.1
Host: example.org
Authorization: hmac johndoe:<digest>
```

- Developer can choose hashing algorithm and what data they want to include
  - Ensure data integrity
  - Including the body prevent data tampering
    - Optional in HTTP digest

HMAC Authentication II

Keyed-Hash Message Authentication Code



source: https://commons.wikimedia.org/wiki/File:SHAhmac.svg

Token authentication

- A token represents the fact that a user is authenticated
  - Client authenticates using secret/password and ask for a token
  - Client sends a token in subsequent requests
  - Can be short or long lived
    - Renew a token using a refresh token
    - Can revoke a token
- Authenticate with a token
  - Request parameter

```
GET /api/profile?access_token=LHnDqNVlXg HTTP/1.1
```

  - Authorization header

```
Authorization: Bearer LHnDqNVlXg
```

JSON Web Token (JWT)

https://tools.ietf.org/html/rfc7519

JWTs represent a set of claims as a JSON object that is encoded in a JWS and/or JWE structure.

a claim:

{“userId”: “arnaud”, “role”: “überadmin”}

JSON Web Signature (JWS)

https://tools.ietf.org/html/rfc7515

JSON Web Signature (JWS) represents content secured with digital signatures or Message Authentication Codes (MACs) using JSON-based data structures.

- A JWS consists of:
  - a JOSE header
  - a payload
  - a signature

{header}.{payload}.{signature}
encoded in base64.url

base64url encoding

https://tools.ietf.org/html/rfc4648

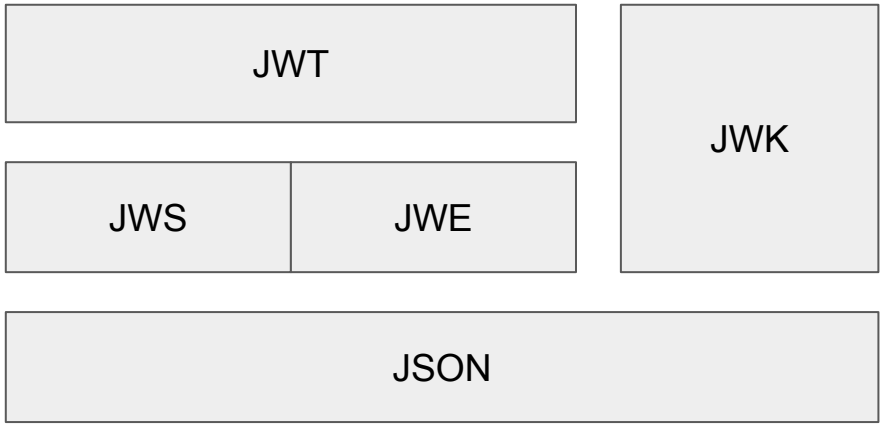
base64url is a base64 variant

- 62 alphanumeric characters
- - instead of +
- _ instead of /
- Padding (“=”) is optional
- URL safe
  - “.” is unreserved, can be used as separator

Value	Char		Value	Char		Value	Char		Value	Char
0	A		16	Q		32	g		48	w
1	B		17	R		33	h		49	x
2	C		18	S		34	i		50	y
3	D		19	T		35	j		51	z
4	E		20	U		36	k		52	0
5	F		21	V		37	l		53	1
6	G		22	W		38	m		54	2
7	H		23	X		39	n		55	3
8	I		24	Y		40	o		56	4
9	J		25	Z		41	p		57	5
10	K		26	a		42	q		58	6
11	L		27	b		43	r		59	7
12	M		28	c		44	s		60	8
13	N		29	d		45	t		61	9
14	O		30	e		46	u		62	-
15	P		31	f		47	v		63	_

JWE, JWK

- JSON Web Encryption (JWE) is like JWT but for encrypting data
  - Enables end-to-end encryption
    - Does not mean we don't need SSL/TLS anymore!!!
- JSON Web Key (JWK) is a JSON representation of a cryptographic web key



OAuth2 I

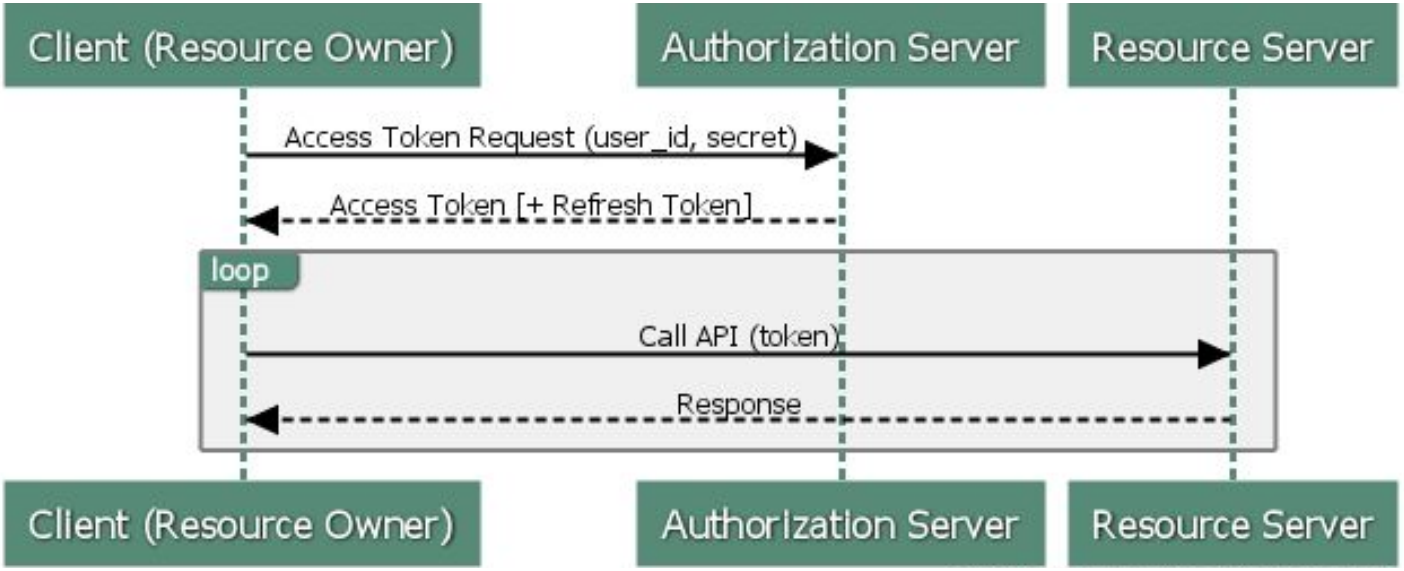
https://tools.ietf.org/html/rfc6749

- Open standard for **authorization**
  - Secure **delegated** access
    - Enable resource access on behalf of a resource owner from a third-party service
- Roles
  - Resource owner
  - Client
  - Resource server
  - Authorization server
- 4 types of authorization
  - Authorization Code Grant (server application)
  - Implicit Grant (client application/browser)
  - Resource Owner Password Credentials Grant
  - Client Credentials Grant

OAuth2 II - Client Credentials Grant

https://tools.ietf.org/html/rfc6749

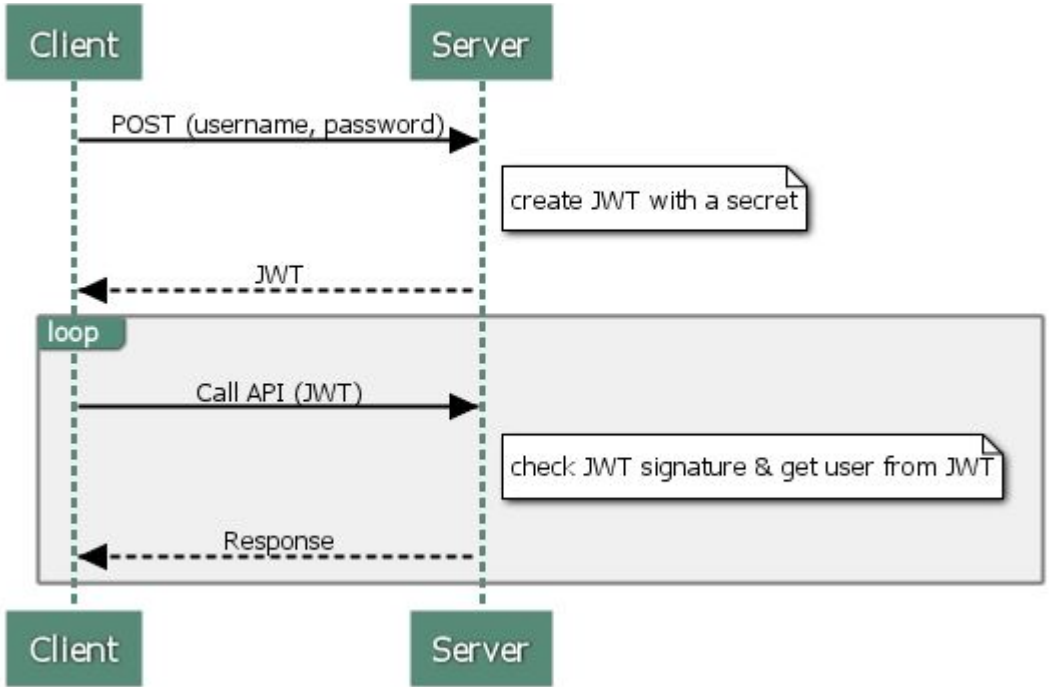
- Client is ressource owner
  - No authorization required for end-user
  - No delegation



Client Credentials Grant with JWT

- Provides JWT token in authorization header

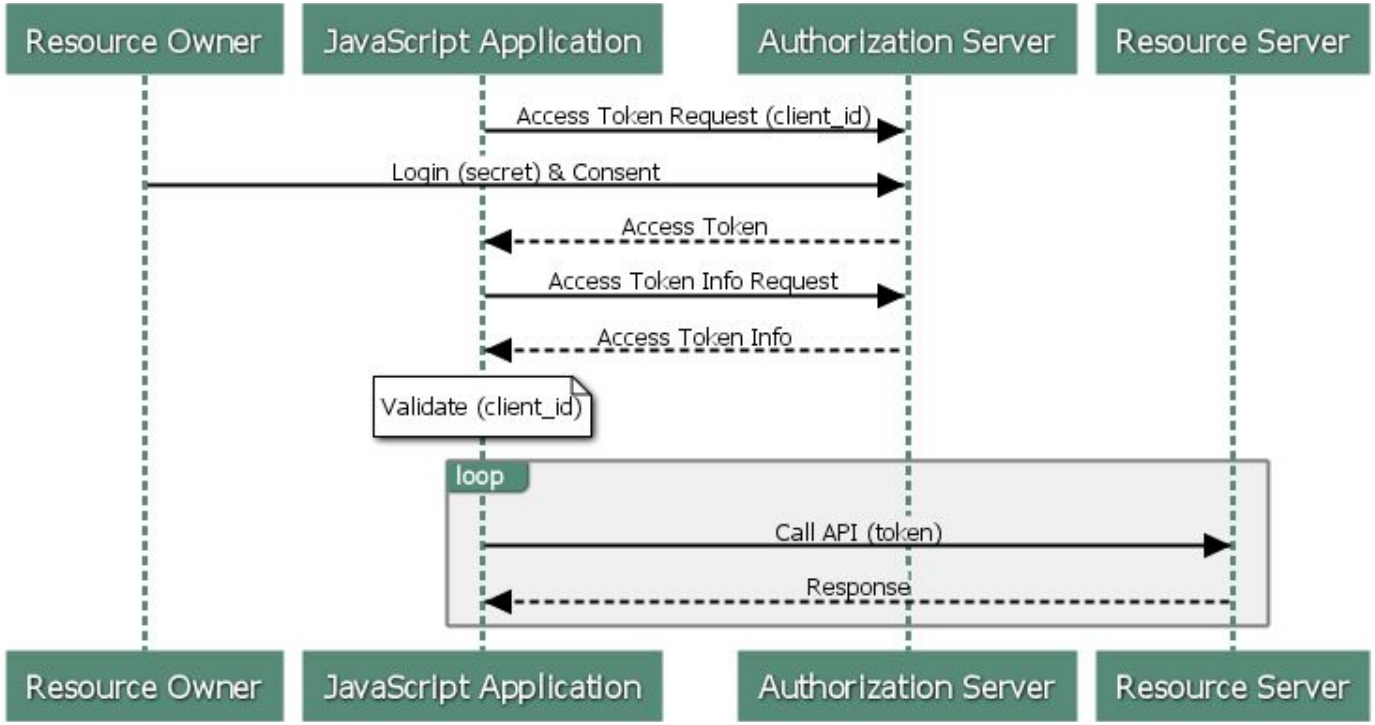
```
Authorization: Bearer <jwt>
```



OAuth2 III - Implicit Grant

https://tools.ietf.org/html/rfc6749

- Client application (typ. JavaScript)
  - e.g. access a Facebook profile from external website
- No refresh token



Multi-factor authentication

Web Authentication (WebAuthn)

- Browser API for authentication with **public key**
  - Useful to authenticate with physical electronic authorization devices (USB tokens, smart-cards...)
- Enable multi-factor authentication with a possession factor
  - Single-factor authentication is also possible
- Universal 2nd Factor (U2F) is a standard interface to USB and NFC tokens.

OpenID Connect

- Open standard for authentication
  - On top of OAuth 2.0
    - REST principles
  - Useful for Single Sign-On
- Client can
  - Verify the identity of an End-User
  - Obtain basic profile information about End User

Single-sign-on is about logging on in one place and having that authenticate you at other locations automatically. OpenID is about delegating authentication to an OpenID provider so you can effectively log on to multiple sites with the one set of credentials.


thank you!