COMP9321: Data services engineering

Week 6: RESTful API Security

Term 1, 2020

By Mortada Al-Banna, CSE UNSW

Key Aspects Security

机密性的

“Security provided by IT Systems can be defined as the IT system’s ability to being able to protect **confidentiality** and **integrity** of processed data, provide **availability** of the system and data, **accountability** for transactions processed, and **assurance** that the system will continue to perform to its design goals”

http://www.opensecurityarchitecture.org/cms/definitions/6-foundations/definitions/3-it-security

Security Design Principles

- Least Privilege
- Fail-Safe Defaults
- Economy of Mechanism
- Complete Mediation
- Open Design
- Separation Privilege
- Least Common Mechanism
- Psychological Acceptability
- Defense in Depth

http://web.mit.edu/Saltzer/www/publications/protection/

Security Design Principles

- **Least privilege**: Every program and every user of the system should operate using the least set of privileges necessary to complete the job.
- **Fail-safe defaults**: Base access decisions on permission rather than exclusion. This principle means that the default situation is lack of access, and the protection scheme identifies conditions under which access is permitted.
- **Economy of mechanism**: Keep the design as simple and small as possible. This well-known principle applies to any aspect of a system

http://web.mit.edu/Saltzer/www/publications/protection/

Security Design Principles

万无一失的

- **Complete mediation**: Every access to every object must be checked for authority. This principle implies that a foolproof method of identifying the source of every request must be devised.
- **Open design**: The design should not be secret. The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, more easily protected, keys or passwords.
- **Separation of privilege**: Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.

http://web.mit.edu/Saltzer/www/publications/protection/

Security Design Principles

- **Least common mechanism**: Minimize the amount of mechanism common to more than one user and depended on by all users.
- **Psychological acceptability**: It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.
- **Defense in Depth**: an approach in which a series of defensive mechanisms are layered in order to protect valuable data and information. If one mechanism fails, another steps up immediately to thwart an attack.

阻碍

http://web.mit.edu/Saltzer/www/publications/protection/

REST API Security…Does it matter?

REST API Security Matters

违背

- Major security breaches happened due to unsecure/unprotected API (e.g., Venmo, Equifax, Impreva)
- It matter enough that OWASP included many instances in their web security **Top ten** related to APIs and they have the **REST Security cheat** sheet.
- REST relies on the elements of the Web for security too (Check OWASP top 10)

https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf
https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

HTTPS (TLS)

- “Strong” server authentication, confidentiality and integrity protection The only feasible way to secure against man-in-the-middle attacks
- Any security sensitive information in REST API should use TLS (formerly known as SSL)

See the OWASP Transport Layer Protection Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

REST APIs Authentication

认证

API developers at least must deal with authentication and authorisation:

验证

Authentication (401 Unauthorized) vs. Authorisation (403 Forbidden):

授权

Common API authentication options:

- HTTP Basic (and Digest) Authentication: IETF RFC 2617
- Token-based Authentication
- API Key [+ Signature]
- OAuth (Open Authorisation) Protocol - strictly uses HTTP protocol elements only

https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

Authentication

The basic idea revolves around: ”login credentials” (for app, not human)

The questions: (i) what would the credentials look like and how would you pass them around “safely”? (ii) how to ensure stateless API interactions? (no ‘session’)

HTTP Basic Authentication Protocol (HTTP Specification)

Initial HTTP request to protected resource

```
GET /secret.html HTTP/1.1
Host: example.org
```

Server responds with

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm=“ProtectedArea”
```

Client resubmits request

```
GET /secret.html HTTP/1.1 Host: example.org
Authorization: Basic Qm9iCnBhc3N3b3JkCg==
```

Further requests with same or deeper path can include the additional Authorization header pre-emptively

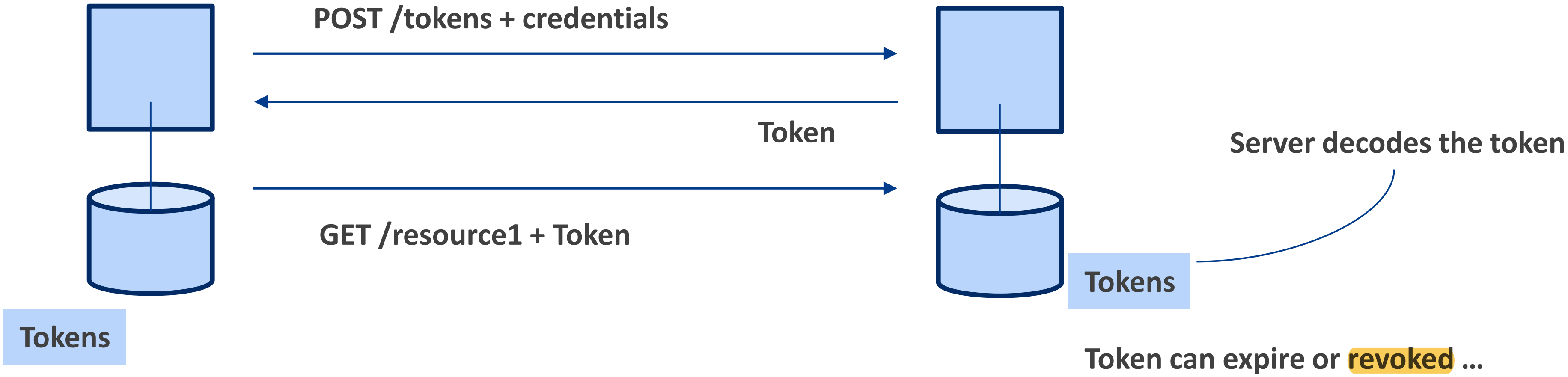
HTTP Basic Auth

Issues with HTTP Basic Auth as an API authentication scheme

- The password is sent over the network in base64 encoding - which can be converted back to plain text
- The password is sent repeatedly, for each request - larger attack window
- HTTP Basic Auth combined with TLS could work for some situations … But normally this scheme is not recommended and considered not secure “enough”

Token-based method

- User enters their login credentials. Server verifies the credentials are correct and returns a token
- This token is stored client-side (local storage). Subsequent requests to the server include this token
- The password is not sent around …



Token can expire or revoked …

撤回，取消

JWT (JSON Web Tokens) – industry standard now (RFC 7519)

e.g., Facebook, Twitter, LinkedIn …

Token-based method

- Question: How could you manage this scheme?
  - Maybe generate a random token/string and store it against each user - ?? User state creeping into the server … Not good (e.g., think about load balancing)

JWT (JSON Web Tokens) – (almost) industry standard now (RFC 7519)

e.g., Facebook, Twitter, LinkedIn …

- The message content consists of three parts JSON data … (encoded and signed)
- A key idea is that the token is self contained. You can store the identity in the JSON, sign it and send the token to the client. The client will use the token in all subsequent requests to authenticate itself.
- Since it’s “signed”, the server can verify and validate the token, without having to do a database look up, session management, etc.
- That is, (i) a token can be effectively used to authenticate requests in a stateless fashion, (ii) login password of the client is not revealed

密码学

The signatures (cryptography technique)

Keyed-Hash Message Authentication Code (HMAC) is an algorithm that combines a certain payload with a secret using a cryptographic hash function.

The result is a code that can be used to verify a message only if both the generating and verifying parties know the secret. In other words, HMACs allow messages to be verified through shared secrets.

Provides authentication as well as integrity properties in security

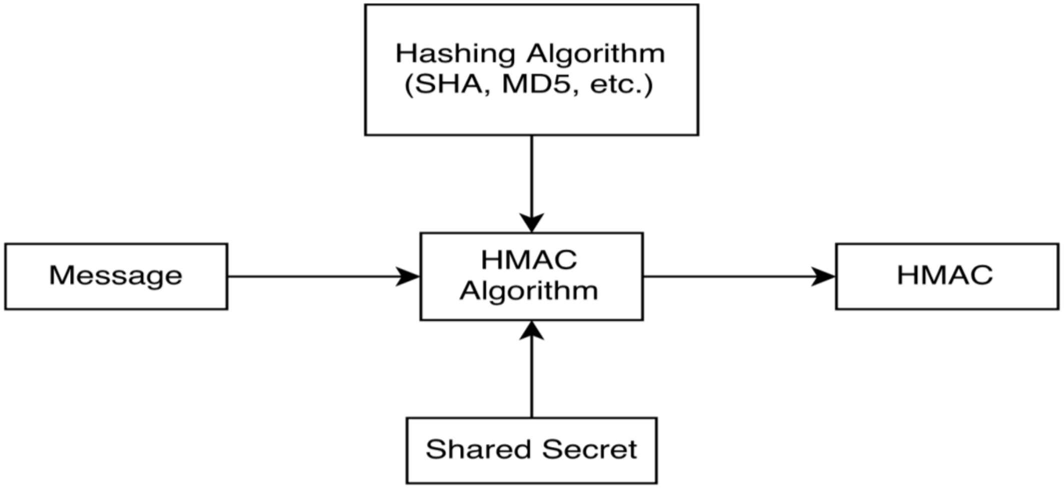
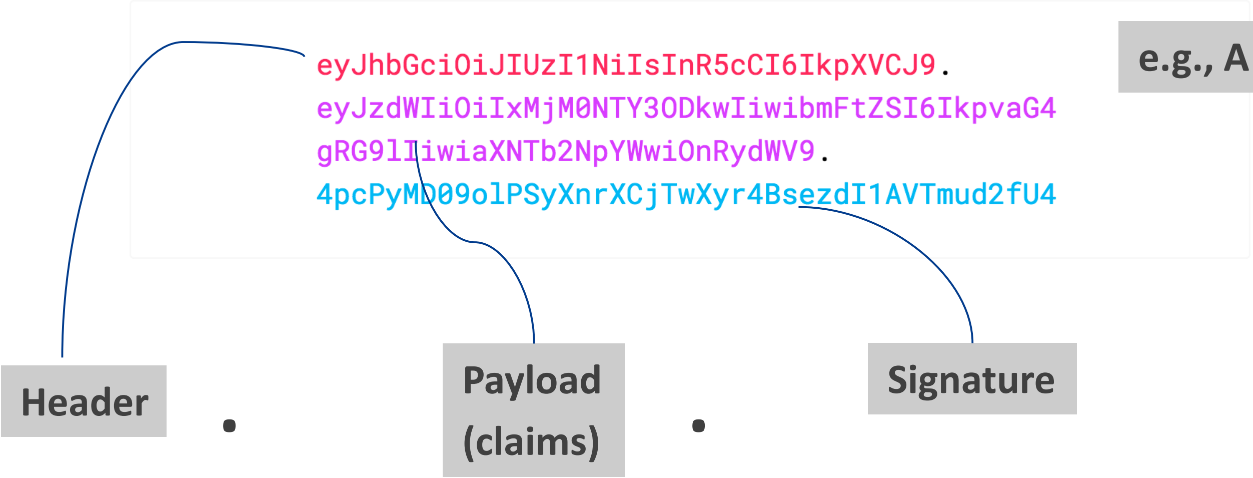


Figure 7.3: HMAC

Token-based method: what is in a token?



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

e.g., A single JWT token

{
type: “JWT”,
alg: “HMAC”
}

Base64 encoded

{
iss: “Twitter API”,
sub:: “12345”,
name: “John Smith”
email: “j.smith@...”
}

Base64 encoded

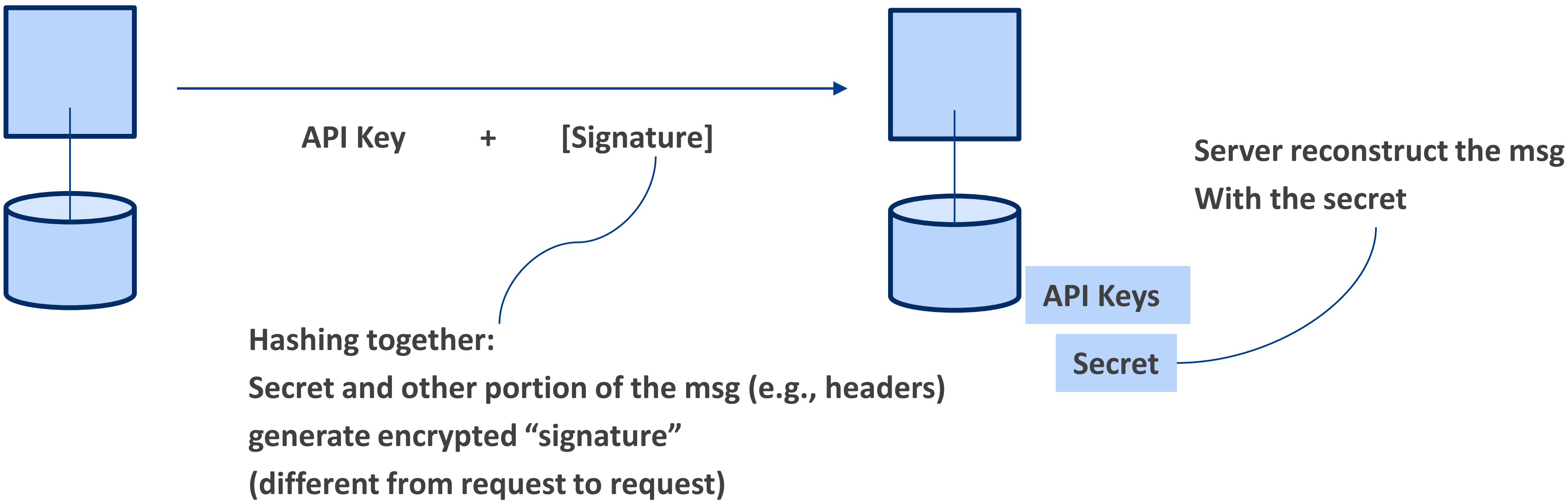
Generated by:

```
HMACSHA256(
      base64UrlEncode(header) + "." +
      base64UrlEncode(payload), secret
)
```

- Token is self-contained
- No separate lookup necessary to authenticate
- Must be used with TLS
- Meant to be short-lived/expire ...

API Keys

- From User (API consumer) point of view:
  - Sign up for the service, API key for the user is issued by the server
  - Copy the issued API key [and secret] in all requests
  - ≈ user id and password, except it is meant to be authenticate the ‘client application’



API Key + [Signature]

Hashing together:
Secret and other portion of the msg (e.g., headers)
generate encrypted “signature”
(different from request to request)

API Keys

Secret

Server reconstruct the msg
With the secret

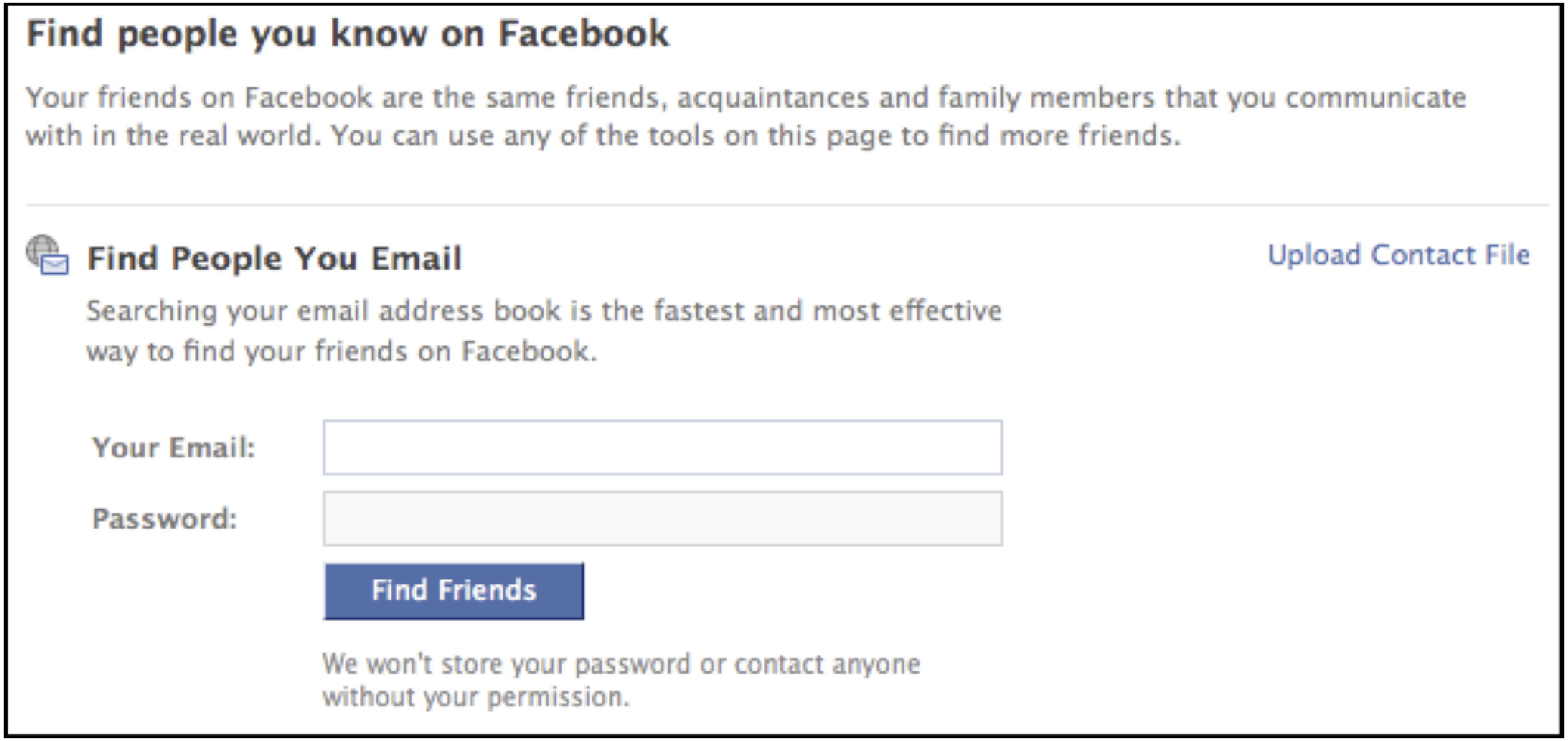
API Key Method

- API key (in combination with Secret) -> an authentication scheme
- Usually, an API key gives you access to a wide range of services from the same provider
- API provider also use it
  - to get usage analytics
  - to limit the usage rate (e.g., 5 calls per second | 429 “Too Many Requests”)
  - to issue further access tokens

- Most API providers would use API management platform (Apigee, Mulesoft, etc.) to handle questions like:
  - Where should the client send the API key and secret?
    - HTTP Header | Query parameters | Request body (custom section)
  - API key & secret management
  - Running an appropriate security scheme
  - Rate limiting and usage analytics

OAuth

- OAuth tries to solve this problem …



Find people you know on Facebook

Your friends on Facebook are the same friends, acquaintances and family members that you communicate with in the real world. You can use any of the tools on this page to find more friends.

Find People You Email

Searching your email address book is the fastest and most effective way to find your friends on Facebook.

Your Email:

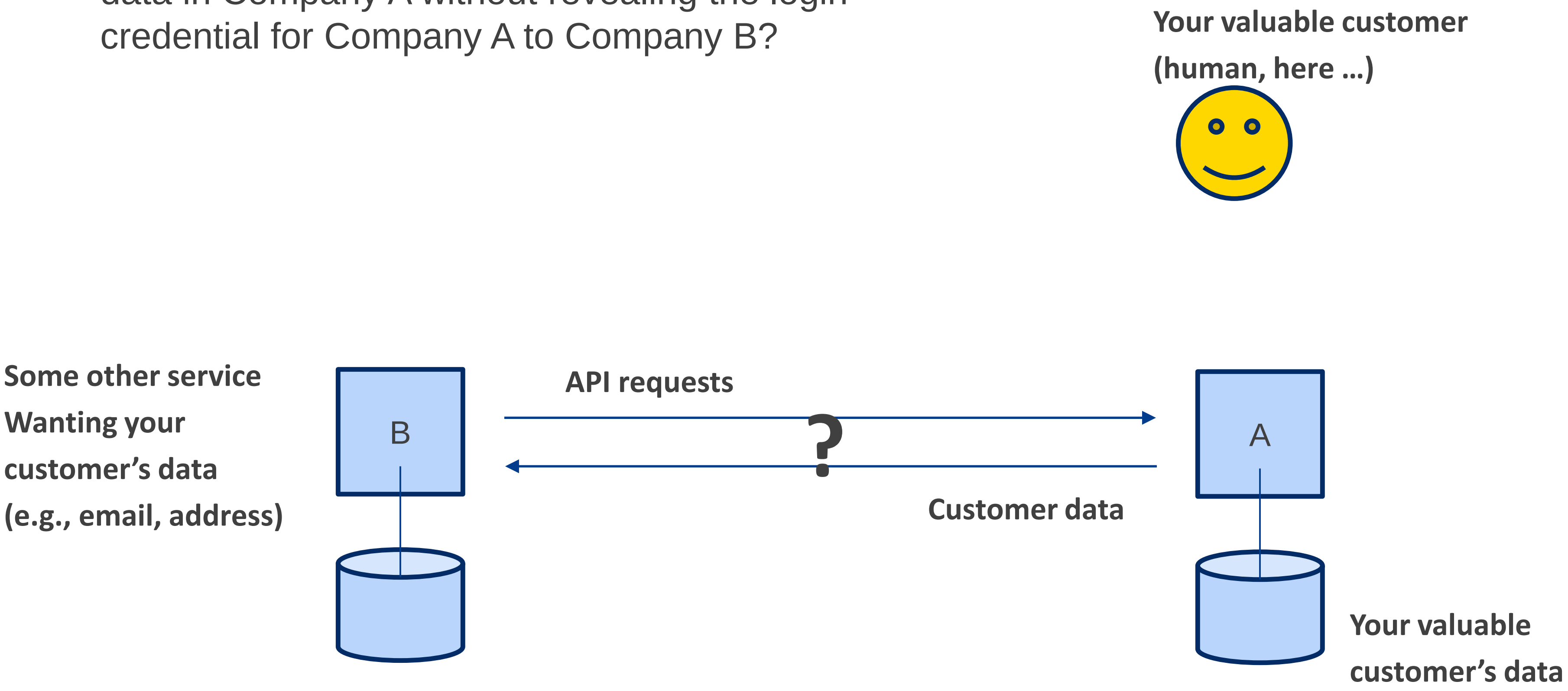
Password:

We won't store your password or contact anyone without your permission.

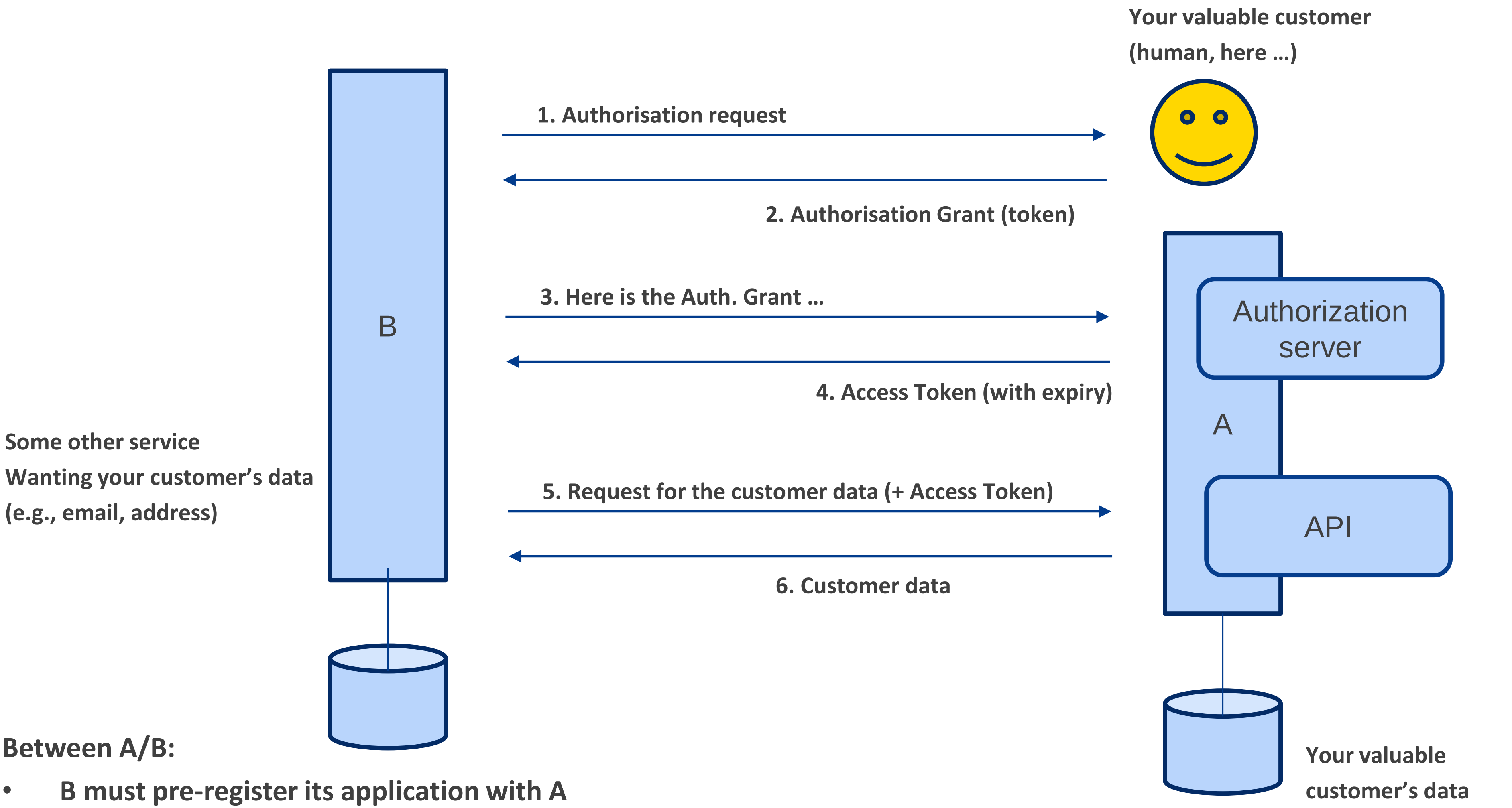
- Essentially, an authorisation scheme for data access …

OAuth (RFC-6749)

How does the user allow Company B to access the data in Company A without revealing the login credential for Company A to Company B?



OAuth workflow (e.g., social login scheme)



Between A/B:

- B must pre-register its application with A
- A issues 'API key' and Secret for B (for Authentication of B)
- Authorisation to data (per customer) is done through OAuth

OAuth has different scope

Facebook:

https://developers.facebook.com/docs/facebook-login/access-tokens

Spotify:

https://beta.developer.spotify.com/documentation/general/guides/scopes/

Should consider the scope during the design of OAuth scheme for your API.

What else you need to know about in regard to REST API security?

- Rate limiting
- Restrict HTTP methods
- Input validation
- Sensitive information in HTTP requests
- Audit logs
- Send Appropriate Error logs

Rate Limiting

- Prevent farming (manage cost)
- Control possible DDoS attacks
- API keys are useful in this regard:
  - ➢Require API keys for every request to the protected endpoint.
  - ➢Return 429 Too Many Requests HTTP response code if requests are coming in too quickly.
  - ➢Revoke the API key if the client violates the usage agreement.

Restrict HTTP methods

- Do Consumers need to use all HTTP methods?
- Apply a whitelist of permitted HTTP Methods e.g. GET, POST, PUT.
- Reject all requests not matching the whitelist with HTTP response code 405 Method not allowed.
- Make sure the caller is authorized to use the incoming HTTP method on the resource collection, action, and record

Input Validation

- Do not trust input parameters/objects.
- Validate input: length / range / format and type.
- Achieve an implicit input validation by using strong types like numbers, booleans, dates, times or fixed data ranges in API parameters.
- Constrain string inputs with regex.
- Reject unexpected/illegal content.

卫生

- Make use of validation/sanitation libraries or frameworks in Python (e.g., Validator Collection)

Sensitive information in HTTP requests

- Sensitive information should not be appearing in the URL
  - ➢In POST/PUT requests sensitive data should be transferred in the request body or request headers.
  - ➢In GET requests sensitive data should be transferred in an HTTP Header.
- Example:

https://example.com/controller/123/action?apiKey =a53f435643de32

Audit logs

- Nothing is 100% secure

事件、插曲

- Logs help detect quickly if there is an incident
- Write audit logs before and after security related events.
- Consider logging token validation errors in order to detect attacks.

使...卫生

- Take care of log injection attacks by sanitizing log data beforehand.

Send Appropriate Error logs

- Use the semantically appropriate status code for the response (remember the API design lecture)
- Respond with generic error messages - avoid revealing details of the failure unnecessarily.
- Do not pass technical details (e.g. call stacks or other internal hints) to the client.

Q&A