



COMP9321:

Data services engineering

Week 6: RESTful API Security

Term 3, 2019

By Mortada Al-Banna, CSE UNSW

Key Aspects of Software Security

- Integrity
- Confidentiality
- Availability

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

REST API Security...Does it matter?

REST API Security Matters

- 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)
- Major security breaches happened due to unsecure/unprotected API (e.g., Venmo, Equifax, Impreva)

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

REST APIs Authentication

API developers at least must deal with authentication and authorisation:

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

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

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

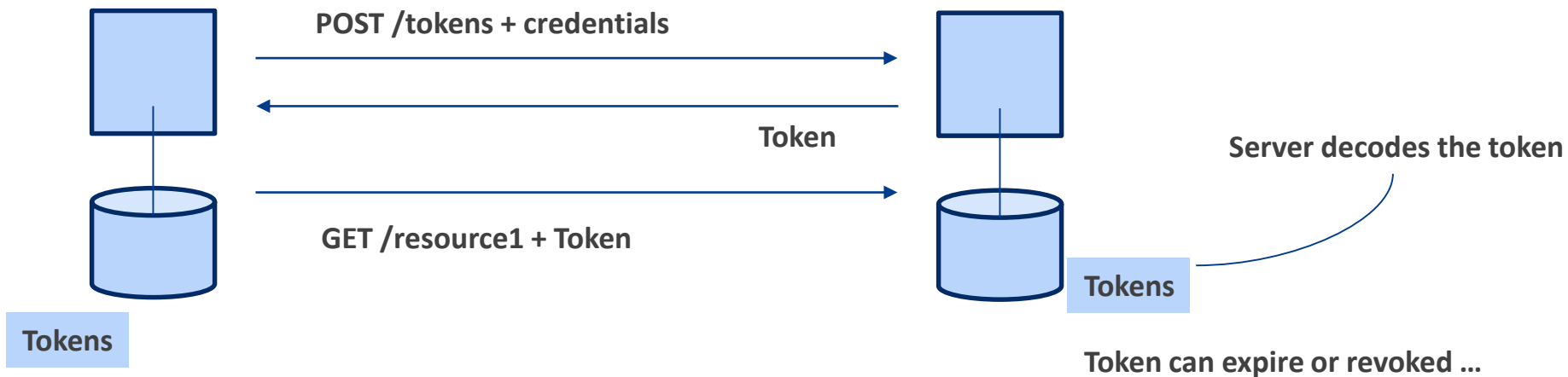
HTTP Basic Auth

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 ...



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

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

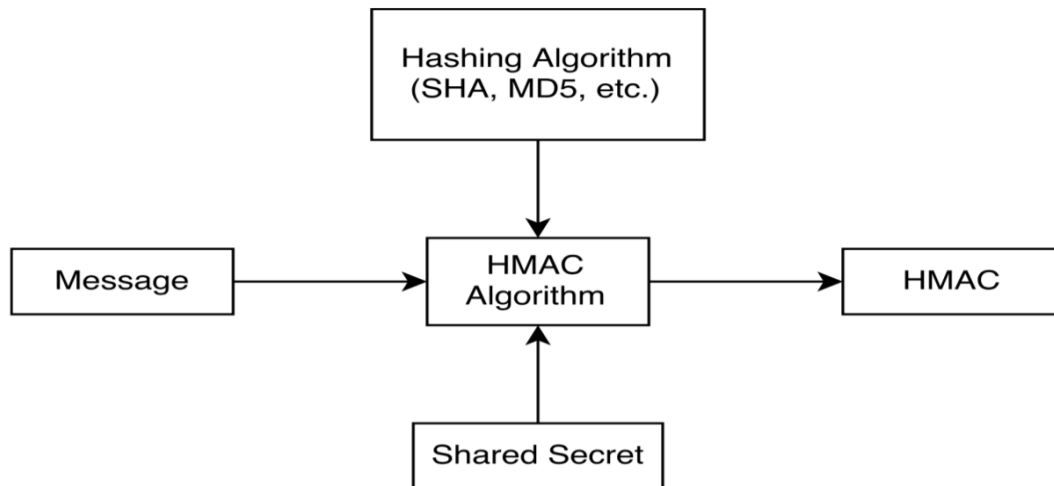
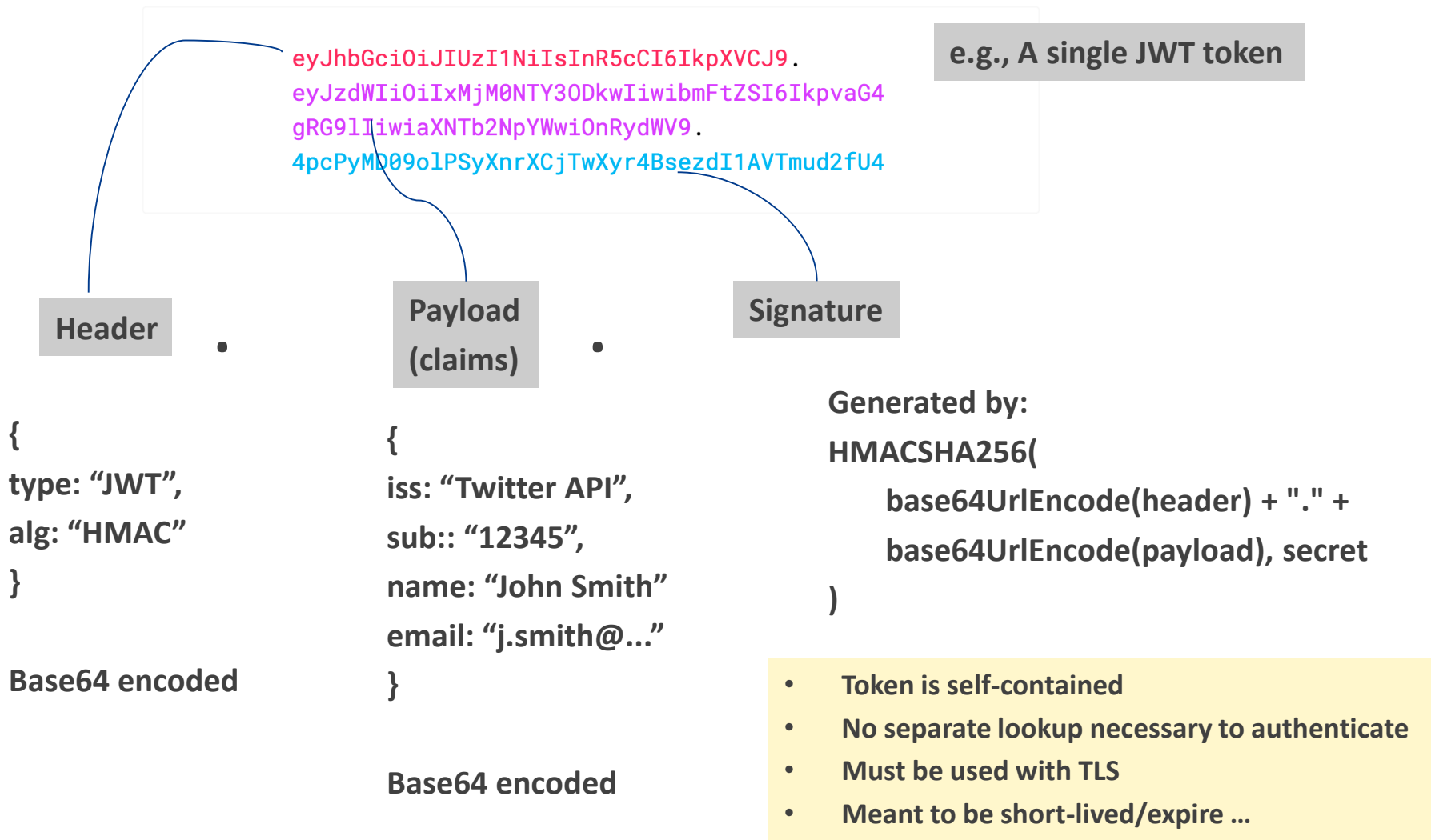


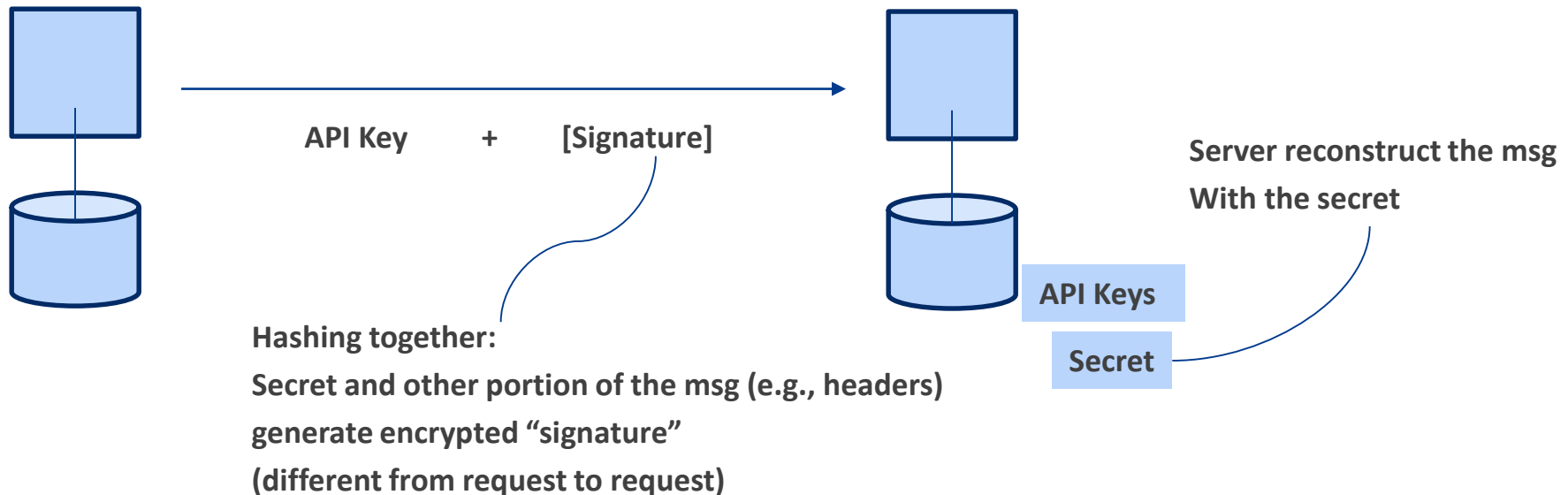
Figure 7.3: HMAC

Token-based method: what is in a token?



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
 - \approx user id and password, except it is meant to be authenticate the 'client application'



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:

Find Friends

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

[Upload Contact File](#)

- Essentially, an authorisation scheme for data access ...

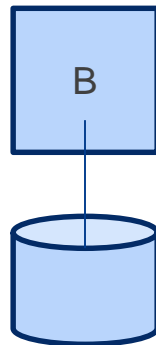
OAuth ([RFC-6749](#))

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?

Your valuable customer
(human, here ...)



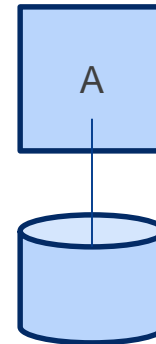
Some other service
Wanting your
customer's data
(e.g., email, address)



API requests

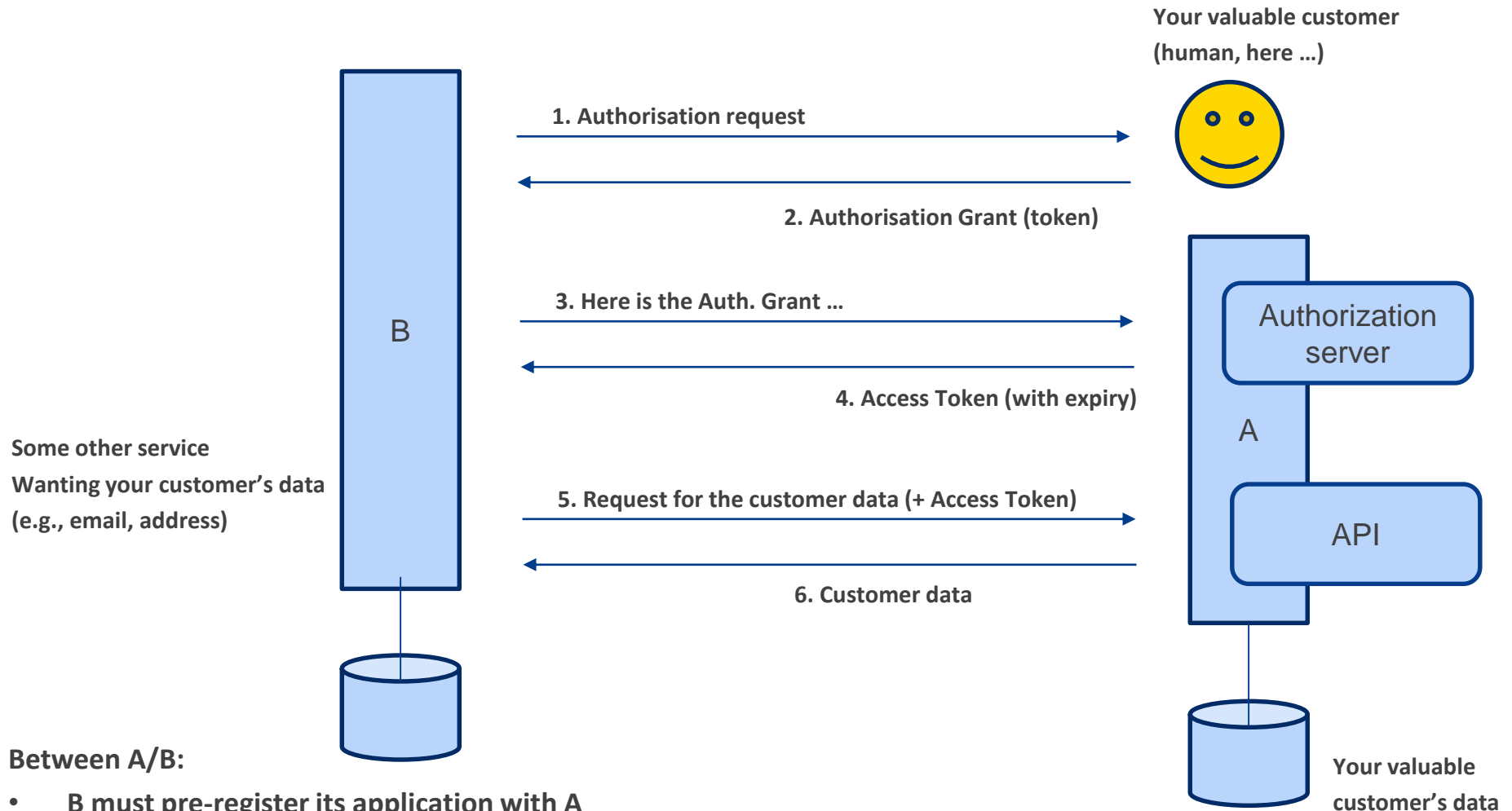


Customer data



Your valuable
customer's data

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.
- 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
- 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.
- Take care of log injection attacks by sanitising 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