Authentication
&
Authorization from First Principles

Kenneth Emeka Odoh

https://kenluck2001.github.io

Table of Contents

- Basics of Cryptography
- Authentication / Authorization
    - OAuth
    - SAML
- Use Cases
- Summary
- Conclusions

Basics of Cryptography

Cryptology consist of the following fields.

- Cryptography
- Cryptanalysis

Cryptography is the process for encrypting and decrypting messages.

Cryptanalysis is the process of recovering plaintext from the cryptotext without the decryption key.

The holy grail of cryptography is to make cryptanalysis very **computationally infeasible**.

- **Stenography** is the art of hiding message in medium that is not obvious. For example, hiding information in images {https://www.csmonitor.com/Science/2010/0630/How-Russian-spies-hid-secret-codes-in-online-photos}

- **Secret key cryptography**
    - The key cannot be made public without compromising security.
- **Public key cryptography**
    - Each user has two keys (private key and public key), if is very difficult to get the private key from the public key. The public key can be announced with compromising security.

Examples:

- p_t: plaintext
- E_k: encryption using the key, k
- D_k: decryption using the key, k
- c_t: cryptotext

$c_t = E_k(p_t)$

$p_t = Dk(ct)$

$D_k(E_k (p_t)) = p_t$

Shared key generation: **Diffie helman key exchange**

Message Signature

- Alice, A, wants to send a message, m, to Bob, B
- Create S_A as the hash of m.
- Alice encrypts the hashed message using her private key, dA

$$D_A (S_A) = S_A^{dA} \bmod n_A$$

- Send message with signature as ($E_B (D_A (S_A))$, $E_B(m)$) to bob
- Once bob receives the message. He verifies the signature to be sure that there are no changes in the messages.

This can provide benefits:

1. **Non-repudiation**
2. **Auditing** [4]

Protocol

- **Protocol** is the sequence of communication steps between entities.
  - This describes the message format and position in the sequence for message delivery and receipt between the participating entities.

Examples of Challenge-response protocol for identification

Schnorr’s identification protocol [chapter 19, [4]]

Zero knowledge proof

Authentication / Authorization

- Authentication is the art of proving your identity.
- Authorization is process of granting access to an authenticated party to allow access to a restricted resource.
- The server does not care about whom the user is but want to **verify** if the person has the **right credentials**.
- Multi-Factor authentication (MFA)

- JSON Web Signature (JWS) - rfc7515 (ietf.org).
- JSON Web Encryption (JWE) - rfc7516 (ietf.org).
- JSON Web Key (JWK) - rfc7517 (ietf.org).
- JSON Web Token (JWT) - rfc7519 (ietf.org).

HTTP Basic Authentication

Basic authentication works as follows:

1. Client sends a request to the server, the server returns a 401 and provides a way to authenticate.
2. On the client, a dialog will prompt the user for a username and password.
3. The client sends the user’s credentials to the server, the username and password are concatenated with a colon separator (username:password), base64-encoded, then added to the Authorization header like so:

Authorization: Basic base64(username:password) rfc2617 (ietf.org)

Issues with Basic Authentication

- key rotation
- Delegation
- Federation
- Storage of user credentials

Variants of Basic Authentication

- Basic
- Digest
- Bearer (for OAuth 2.0)
- HOBA (HTTP Origin-Bound Authentication, RFC 7486, draft)
- Mutual (Mutual Authentication Protocol, draft)

[1]

Identity Delegation

Helping a third-party to authenticate

- access a resource on your behalf.

Roles include:

- delegator: owns the resource (resource owner)
- delegate: want to access a service on behalf of the delegator.
- service provider (resource server): host the protected resource and validates the delegate.

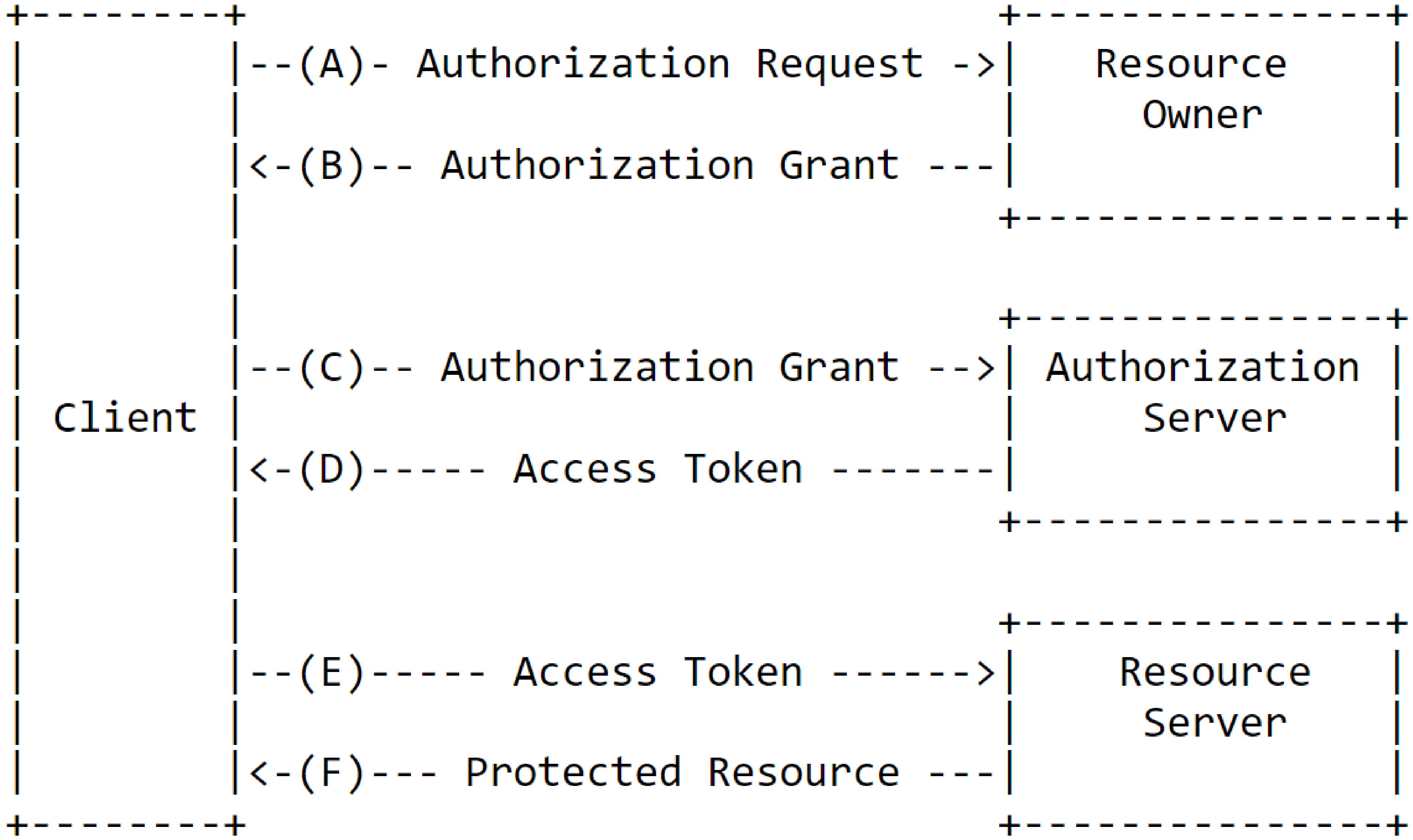
OAuth 2.0

- it is a **delegated authorization framework**. it allows scoped permissions to give restricted access to user without the need for password.
- It separates authentication from authorization.
- Oauth1.0a and Oauth2.0 are very different and backward-incompatible.

OAuth is not an authentication framework.

Participants in the protocol

- Client
- Resource owner
- Authorization server
- Resource server



```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

https://www.rfc-editor.org/rfc/rfc6749

Figure 1: Basic OAuth Protocol

Dissecting OAuth

- OAuth is **not used for authorization**.
- OAuth is also **not for authentication**.

Kinds of token

- Access Tokens: These are tokens that are presented to the client
- Refresh Tokens: These are used by the client to get a new access token from the Authorization Server.

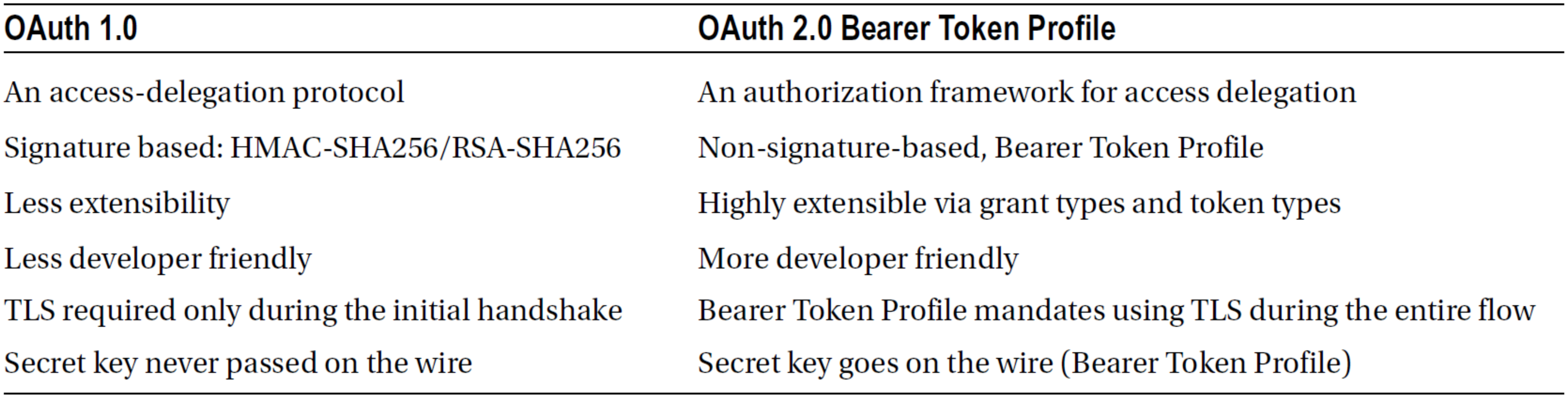
Profiles of token

- Bearer tokens
- Holder of Key (HoK) tokens

Token Format

- JWT token
- SAML token

[1, 3]



OAuth 1.0	OAuth 2.0 Bearer Token Profile
An access-delegation protocol	An authorization framework for access delegation
Signature based: HMAC-SHA256/RSA-SHA256	Non-signature-based, Bearer Token Profile
Less extensibility	Highly extensible via grant types and token types
Less developer friendly	More developer friendly
TLS required only during the initial handshake	Bearer Token Profile mandates using TLS during the entire flow
Secret key never passed on the wire	Secret key goes on the wire (Bearer Token Profile)

[2]

Mode of Operation (OAuth2 protocol)

1. **Client** requests authorization from **Resource owner**.
2. **Resource owner** authorizes **client** and delivers a **grant**.
3. Client presents **grant** to the authorization server to get a Token.
4. The **Token** is restricted to only access what the **Resource owner** authorized for the specific **Client**
5. Resources (APIs) validate the **Token** as having the proper and expected authorizations

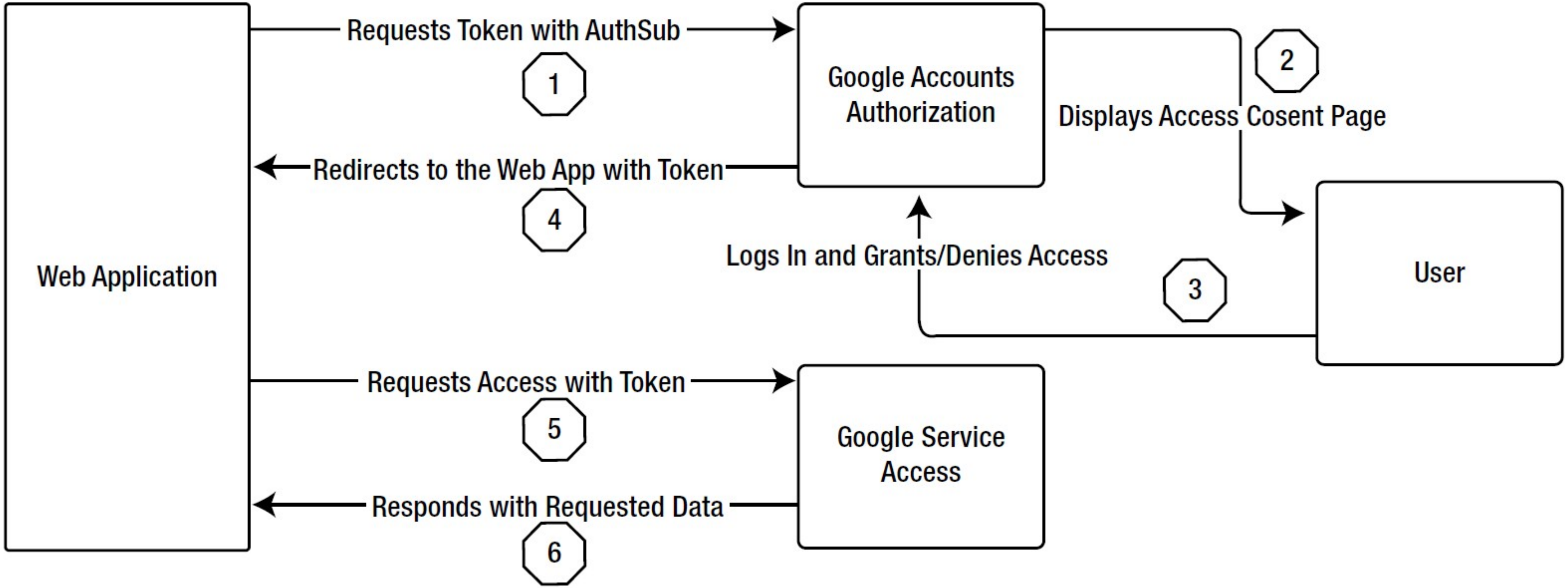
SAML

- Security Assertion Markup Language (SAML)
- it is an XML framework to allow identity and security info to be shared across domains.
- Assertion is a security token

rfc7522 (ietf.org)

Use Cases

Google AuthSub

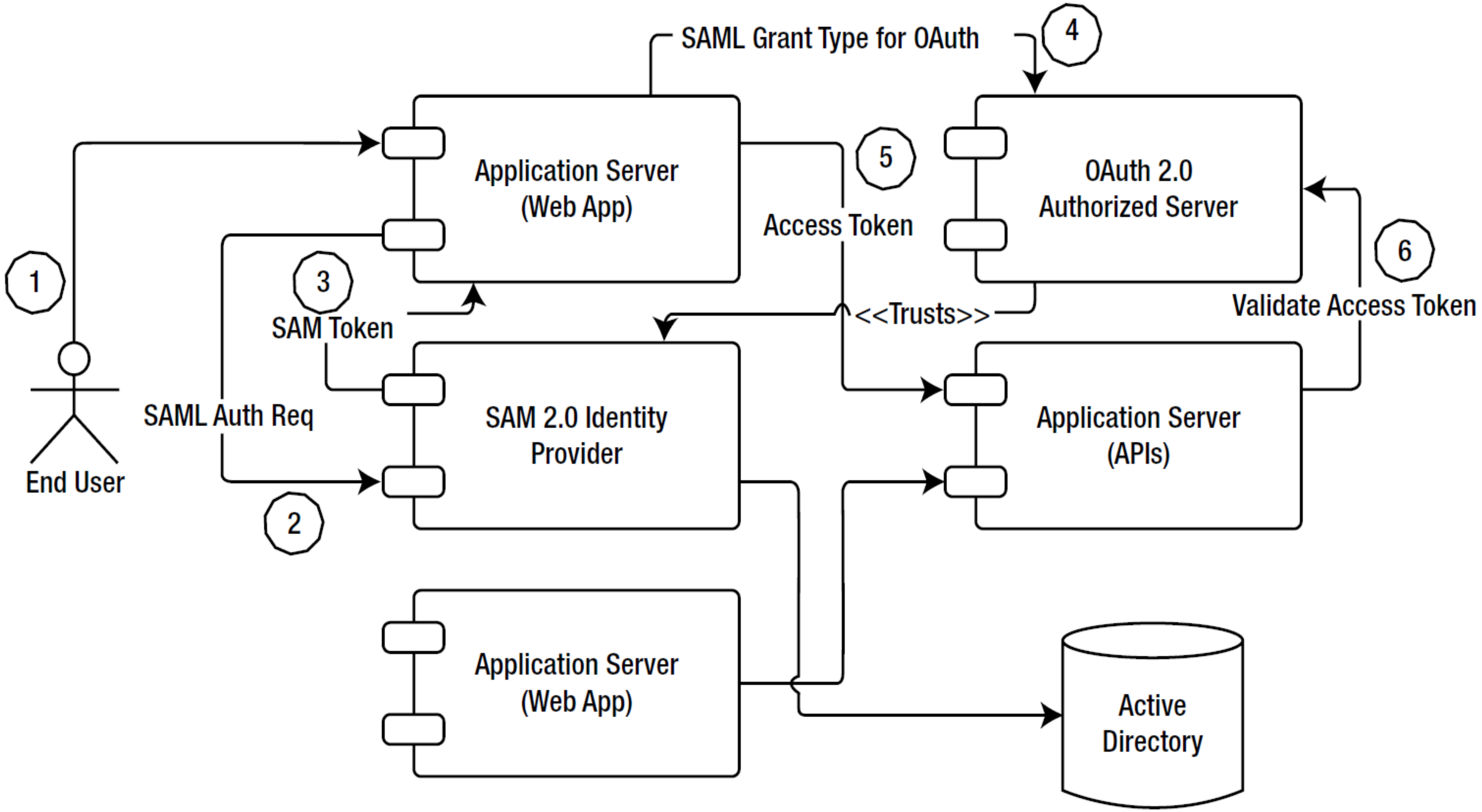


Web Application
Requests Token with AuthSub
1
Google Accounts Authorization
2
Displays Access Cosent Page
User
Logs In and Grants/Denies Access
3
Redirects to the Web App with Token
4
Requests Access with Token
5
Google Service Access
Responds with Requested Data
6

[2]

Figure 2: Google AuthSub Protocol

Single Sign-on with Delegated Access Pattern

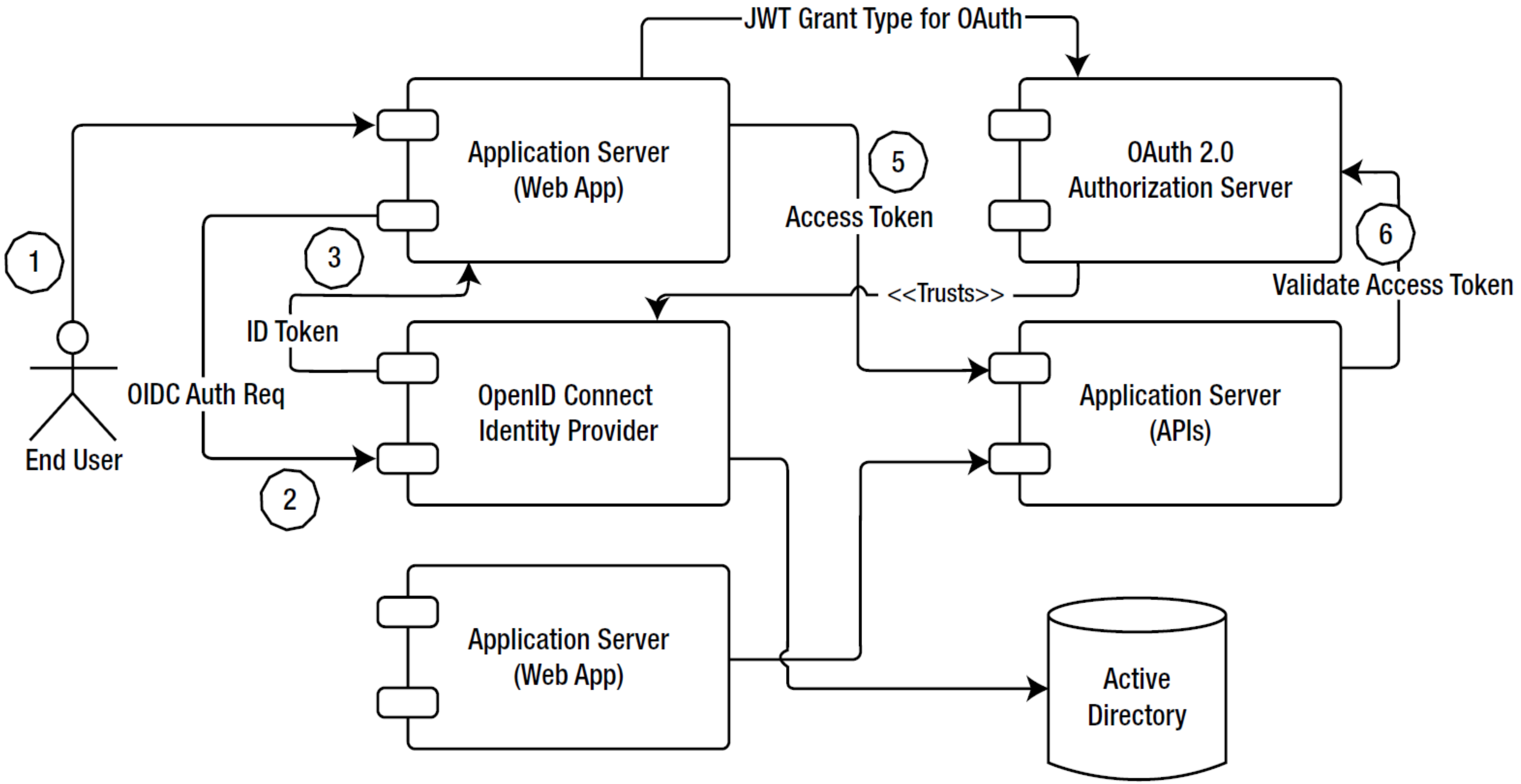


SAML Grant Type for OAuth
4
Application Server (Web App)
5
OAuth 2.0 Authorized Server
Access Token
6
1
3
SAM Token
<<Trusts>>
Validate Access Token
SAML Auth Req
SAM 2.0 Identity Provider
Application Server (APIs)
End User
2
Application Server (Web App)
Active Directory

[2]

Figure 3: SSO delegated access pattern

Delegated Access using JWT


JWT Grant Type for OAuth
Application Server (Web App)
OAuth 2.0 Authorization Server
5
Access Token
6
Validate Access Token
1
3
ID Token
<<Trusts>>
OpenID Connect Identity Provider
Application Server (APIs)
OIDC Auth Req
End User
2
Application Server (Web App)
Active Directory

[2]

Figure 4: delegated access using JWT

Summary

Securing your Resources

- Begin with a **threat** model
  - Identify the adversaries that you are protecting against, if it is a serious adversary be ready for shock e.g heartbleed bug.
  - How long should the information be secure.
- Identify the **trust** model.
- Identify the computation resources for encrypting and decrypting with minimal bottleneck.
- Reduce attack surface. Identify all attack vectors and handle the cases.
- Reduce severity of breaches with careful design. This fall into **crisis response**.
- Cryptography should be used in the mix of other techniques e.g secure coding, access control (permissions management) among others.

Design Principles of Authentication Systems

- Least privilege
- Fail-safe defaults
- Simple
- Validate access rights before granting resource
- No query about the user is an anti-pattern
- Authentication server must be performance and guarantee availability, or it becomes a centralized bottleneck for the entire user experience.

[2]

Conclusions

- Use only well-known standard for designing authentication protocols.
  - If new
    - Ensure it is discussed with the community in the RFC
- Security should not be an afterthought
- Security-first philosophy is ideal for building systems

References

1. API Security: A guide to building and securing APIs from the developer team at Okta.
2. Advanced API security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE.
3. API Security: A Collection of Articles
4. A Graduate Course in Applied Cryptography by Dan Boneh and Victor Shoup
5. Authy (https://medium.com/galvanize/fast-authorization-with-dynamodb-cd1f133437e3)