Web Application Architecture

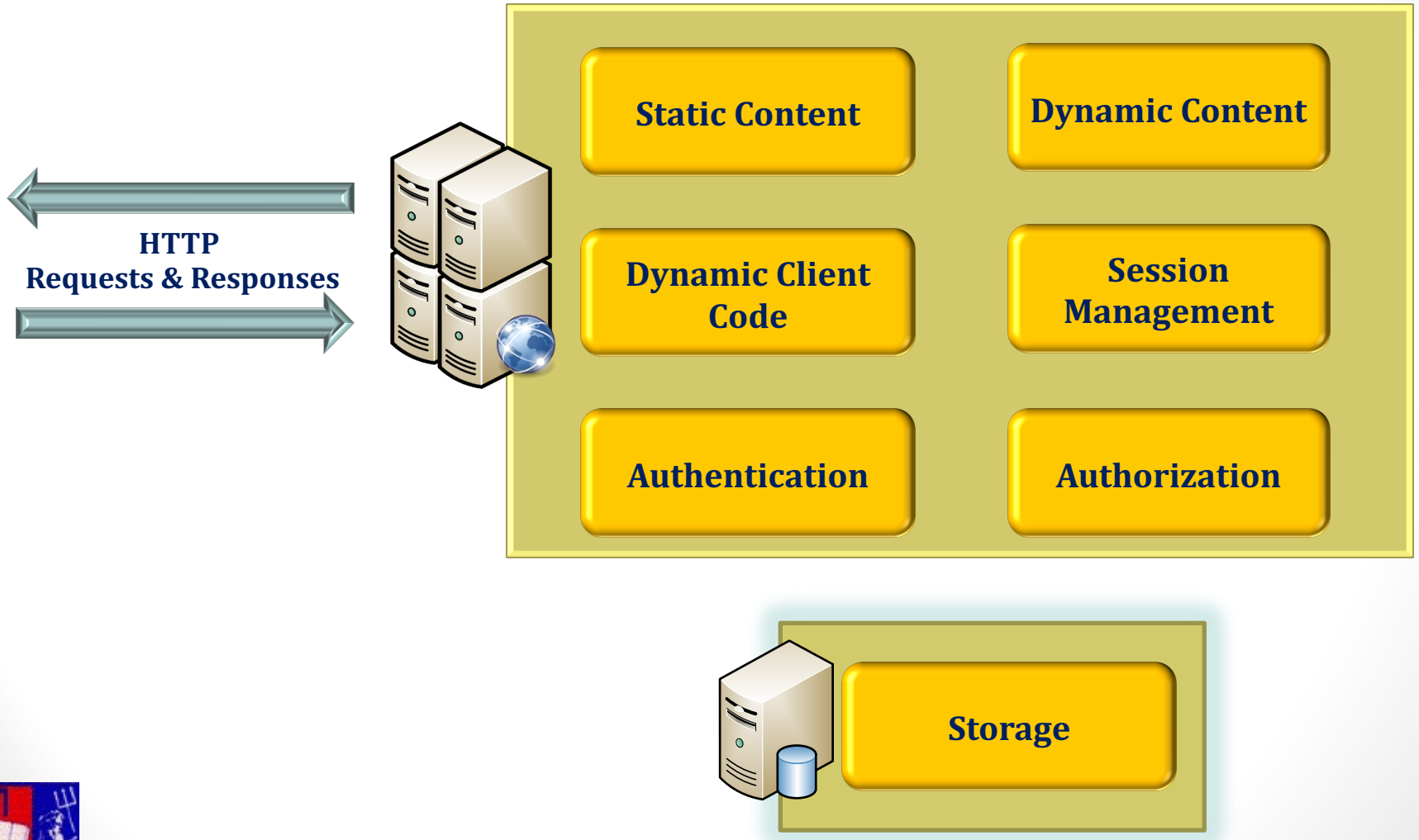
Τεχνολογιες Διαχειρισης Ασφαλειας (ΤεΔΑ) - ΠΜΣ Προηγμενα Συστηματα Πληροφορικης

Δρ. Καραντζιας Θανος

Topics

- High Level Web Architecture
- Static Content
- Dynamic Content
- Session Management
- Authentication
- Authorization
- Storage

High Level Architecture



Static Content

- Images (company logos, etc)
- Style-sheets
- Client libraries
- It can often be cached and pre-fetched,
- (Depending on the actual content) It can still require authentication and be subject to authorization rules, making it unfit for caching

Dynamic Client Code

- Complex and interactive web applications generate user-specific content
- It needs to be processed or displayed at the client side
- It is considered dynamic client code, since it is dynamically generated at the server-side, but executed at the client
- Dynamic client code typically contains sensitive information, and should not be cached

Dynamic Server Code

- The business logic of a web application is considered dynamic code
- It is responsible for retrieving information and processing requests, often in close conjunction with backend storage mechanisms
- The business logic of an application is typically closely coupled with authentication and authorization, and is almost always combined with static content

Session Management

- Web applications have the possibility to share state among requests (such as the authentication status of the user, or a shopping cart with items to purchase, etc.)
- Session management is a crucial component for any dynamic web application
- It is required by components such as the business logic, authentication or authorization

Session Management Solutions

With Authorization Header

- Session management can be implemented *on top of the Authorization header*
- The browser sends the user's credentials on every request
- The server keeps track of the user sending the request, implicitly defining a session
- It does not require the use of a session identifier, but it uses the credentials of the user to identify the correct server-side session state

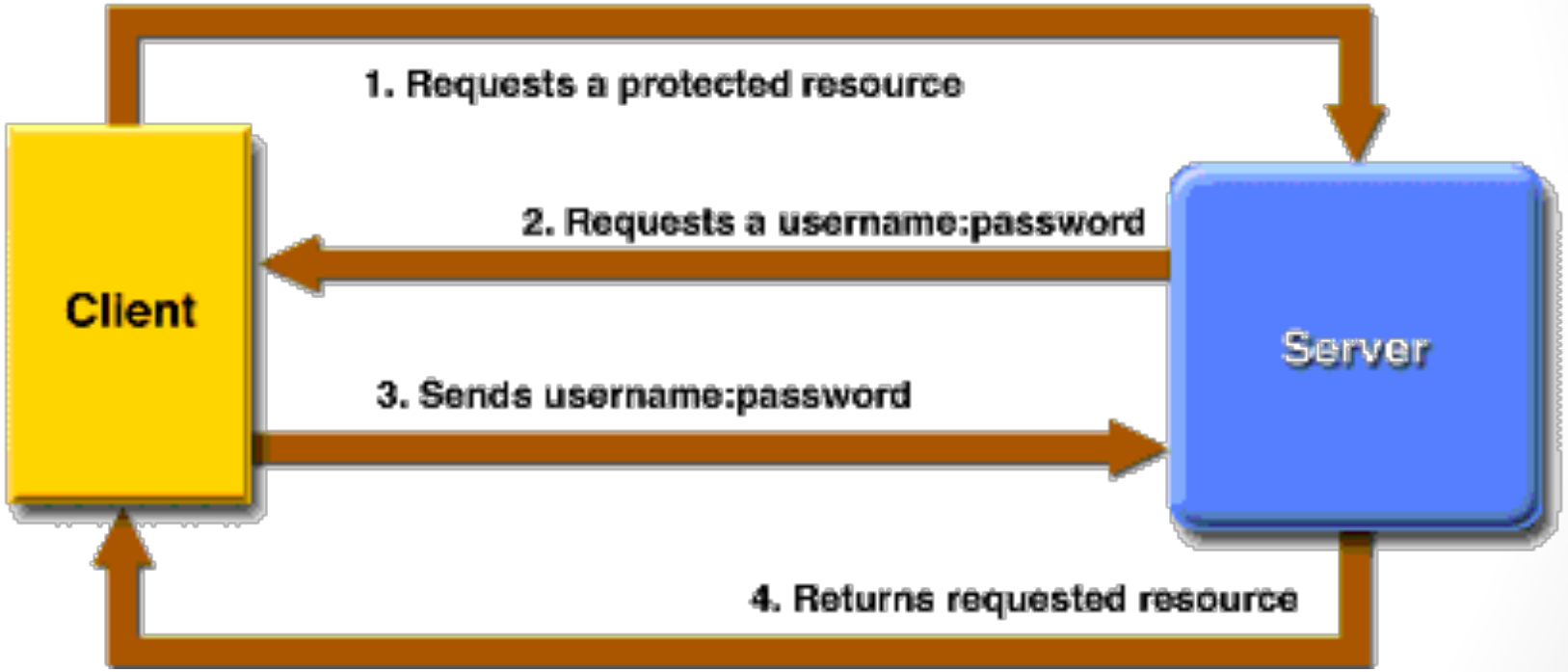
NOTE: The user's credentials have to be sent on every request

NOTE: This approach does not support anonymous sessions when a user is not yet authenticated (e.g., a shopping cart on a web shop)

NOTE: There is a lack of integration of the authentication prompt with the layout of the application

Session Management Solutions

With Authorization Header



Session Management Solutions

With Cookies

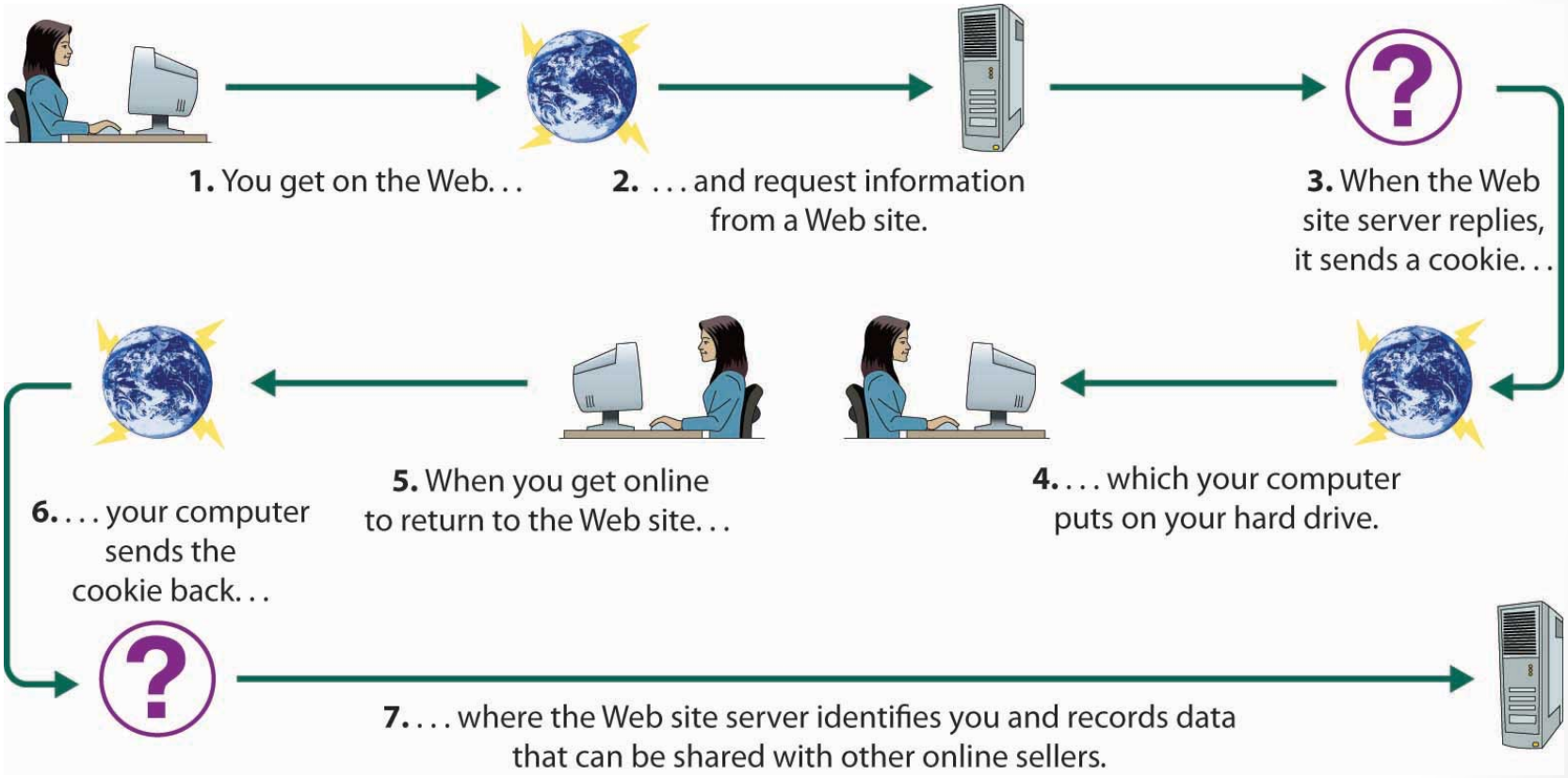
- When the server wants to initiate a session with the browser
  - it generates a new session identifier
  - It associates it with the server-side session object
  - It sends the session identifier to the browser using the Set-Cookie response header
- The server-side session object is typically referred to using a *Session Identifier (SID)*
- When the client presents a valid SID, the server automatically assumes that the request belongs to this session

NOTE: A SID should be sufficiently long, random and unique, in order to avoid collisions, guessing attacks or brute forcing.

NOTE: Most web development languages and frameworks offer out-of-the-box support for session management, and developers are strongly encouraged to use them instead of rolling out their own session management mechanisms

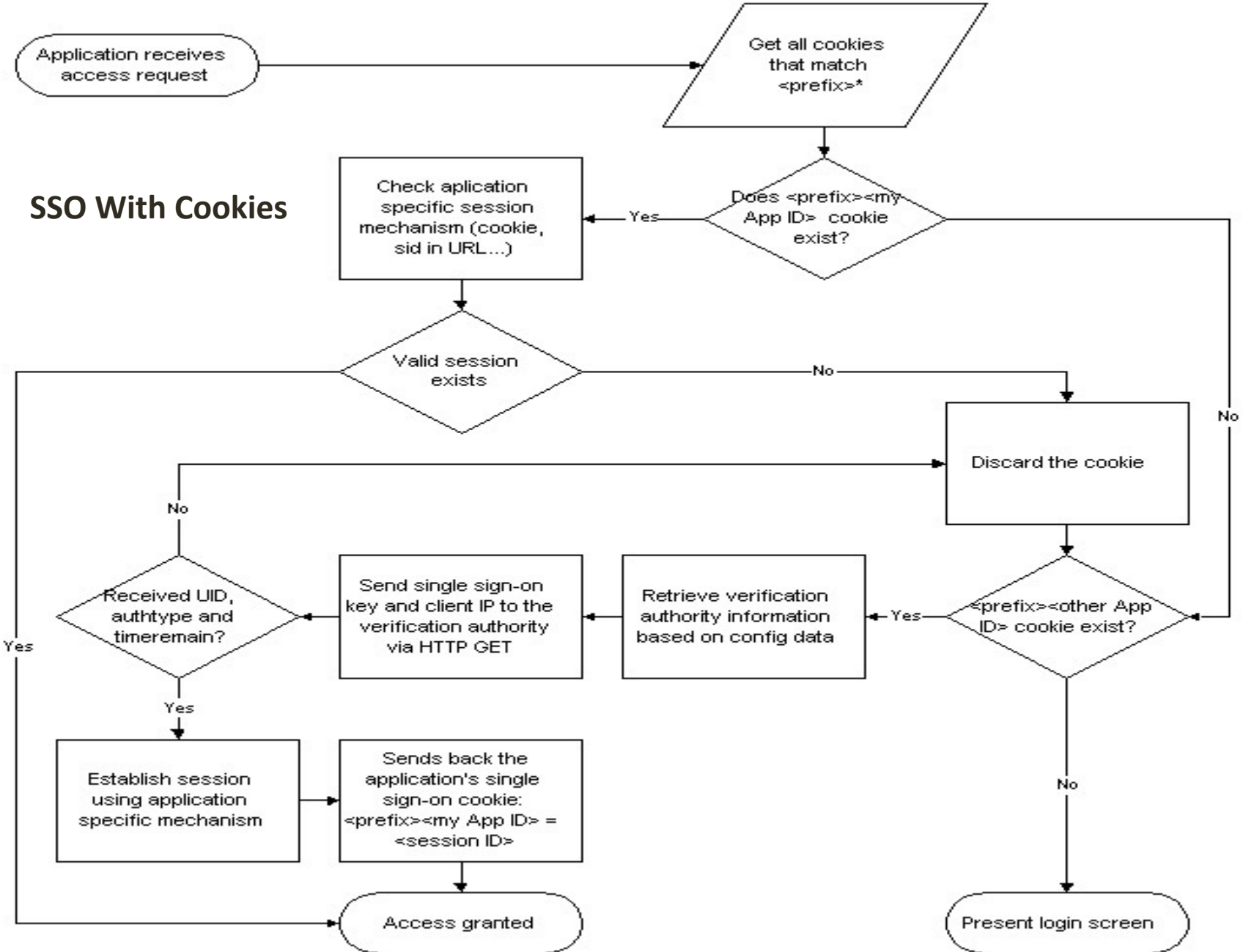
Session Management Solutions

With Cookies



Session Management Solutions

SSO With Cookies



Session Management Solutions

With URI Parameters

- The session identifier can be embedded as a parameter in the URI
- It enables the same session management techniques, with the delivery mechanism of the SID as the only difference

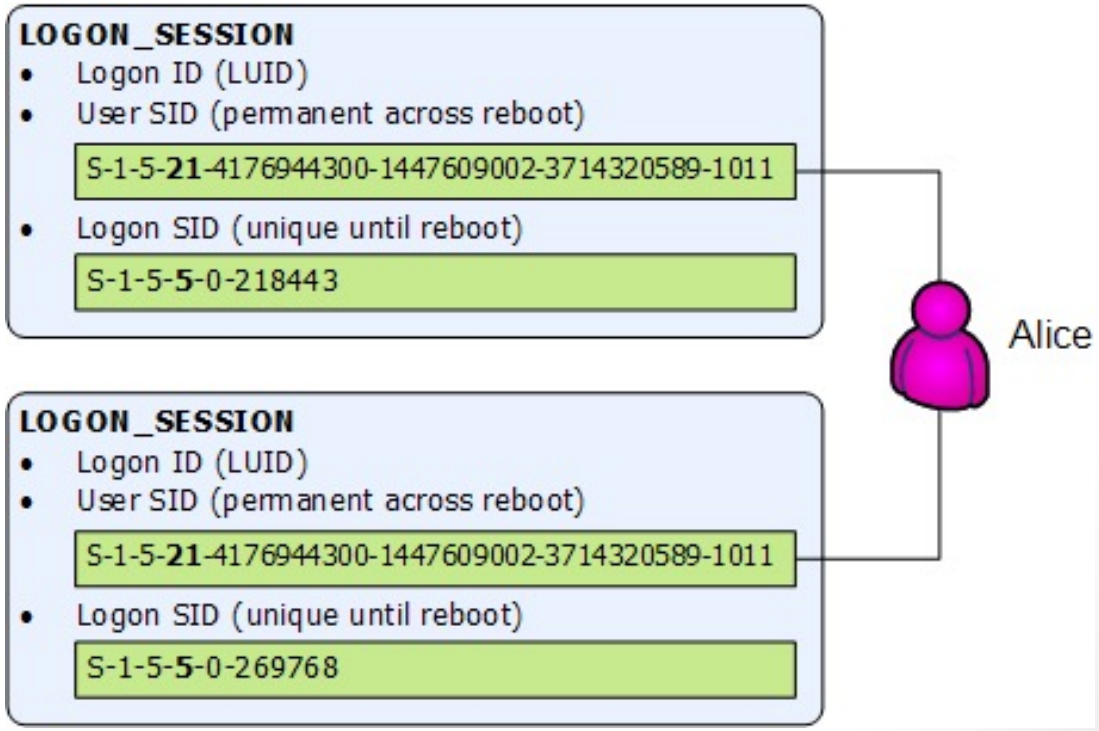
NOTE: The session identifier needs to be present in the URI, requiring the server to rewrite every URI in the response to include the correct SID. This URI rewriting is especially problematic for web applications that generate URIs at the client side, for JavaScript-based retrieval of information.

NOTE 2: Embedding a parameter in the URI can cause unintentional leaking of the SID through the Referrer header

Session Management Solutions

Windows Example

- Windows gives a SID to every logon session
- If the user logs out and logs back in again, he will be attributed a new Logon SID
- If the same user is logged locally and logged on remotely at the same, there will be a different Logon SID for each logon session



Authentication

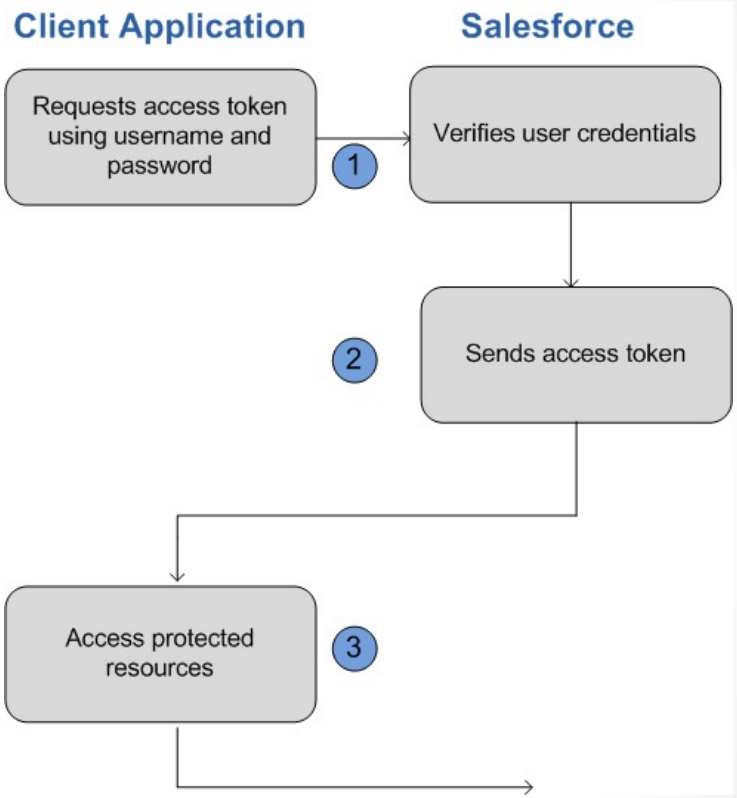
- Modern web applications offer access to large amounts of information (typically user-specific information, making authentication an important component within the web application)
- The authentication procedure allows a user to identify himself to the web application, enabling access to the protected features of the application

NOTE: *The stateless nature of HTTP couples authentication with session management, in order to maintain the authentication state across requests.*

Authentication Mechanisms

With a username and password

- Upon first registration with a web application, a username and password can be chosen or are given
- Future authentications depend on the knowledge of these credentials
- The username and password are stored by the web application (typically in a database)



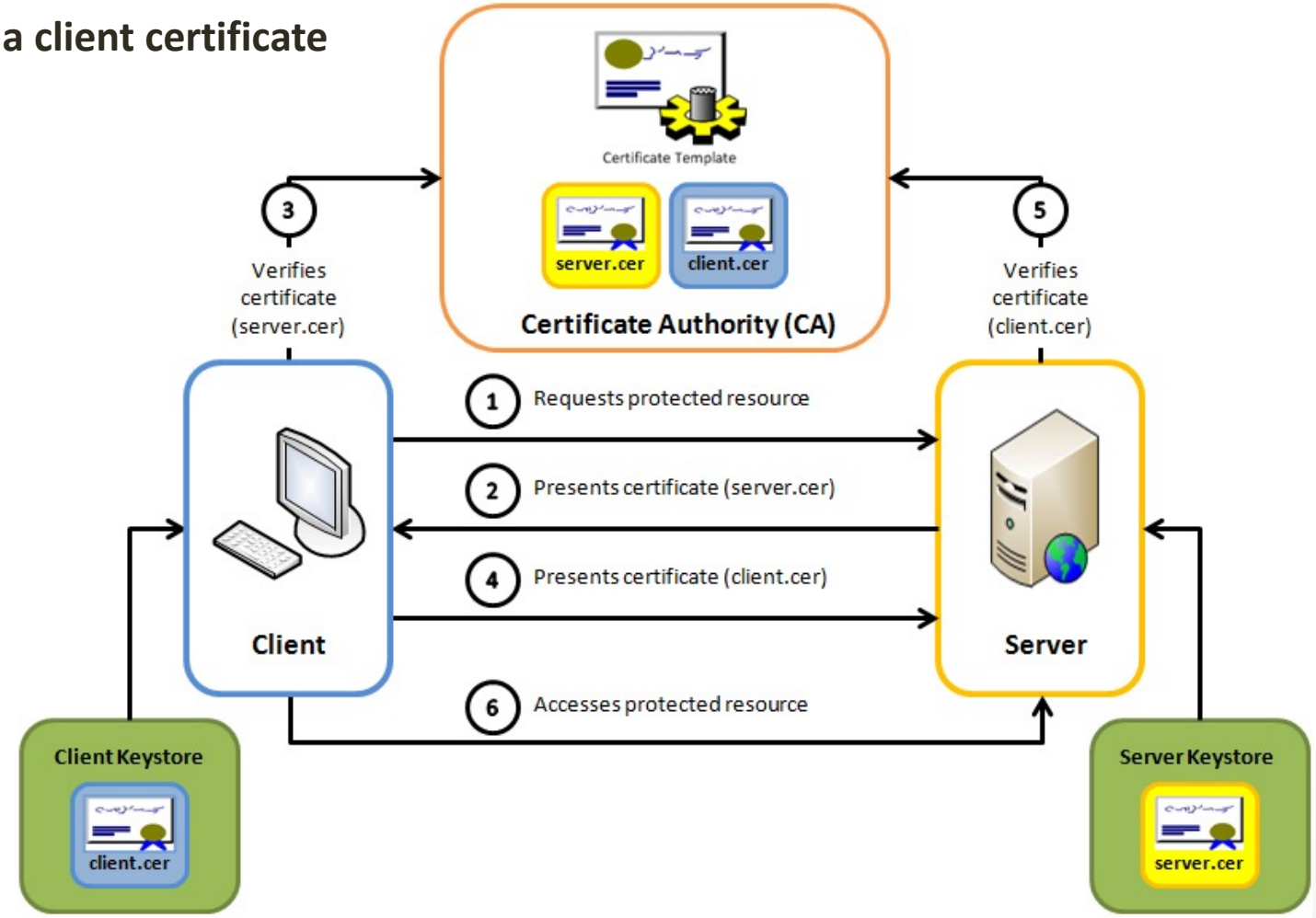
Authentication Mechanisms

With a client certificate

- Is used to create a secure (e.g. SSL/TLS) connection, where both client and server are authenticated with their certificates
- The server verifies whether the client certificate is valid (e.g., has a valid signature by an expected CA), and can extract user-specific information from the certificate
- Client certificates can be explicitly installed in the browser, or can be used in combination with electronic identity cards or other smartcards

Authentication Mechanisms

With a client certificate



Mutual SSL authentication / Certificate based mutual authentication

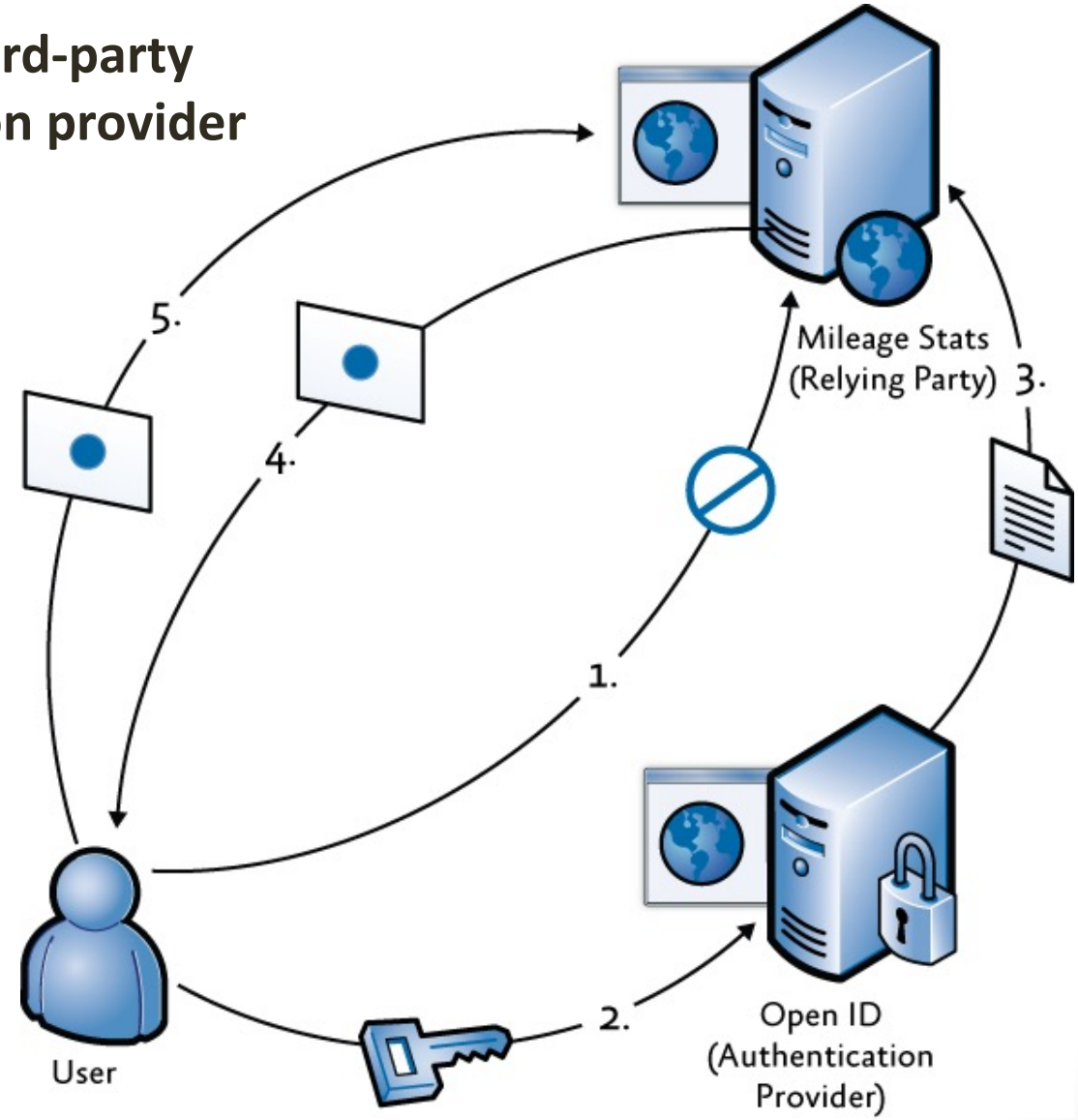
Authentication Mechanisms

Through a third-party authentication provider

- A web application simply directs the user to the authentication provider (e.g. OpenID, Google, Facebook)
- The provider takes care of the whole authentication process
- After a successful authentication, the application receives some evidence of the authentication, together with the user's details
- The exact details depend on the implementation and user's configuration, but typically include a username, a name, an email address, etc.

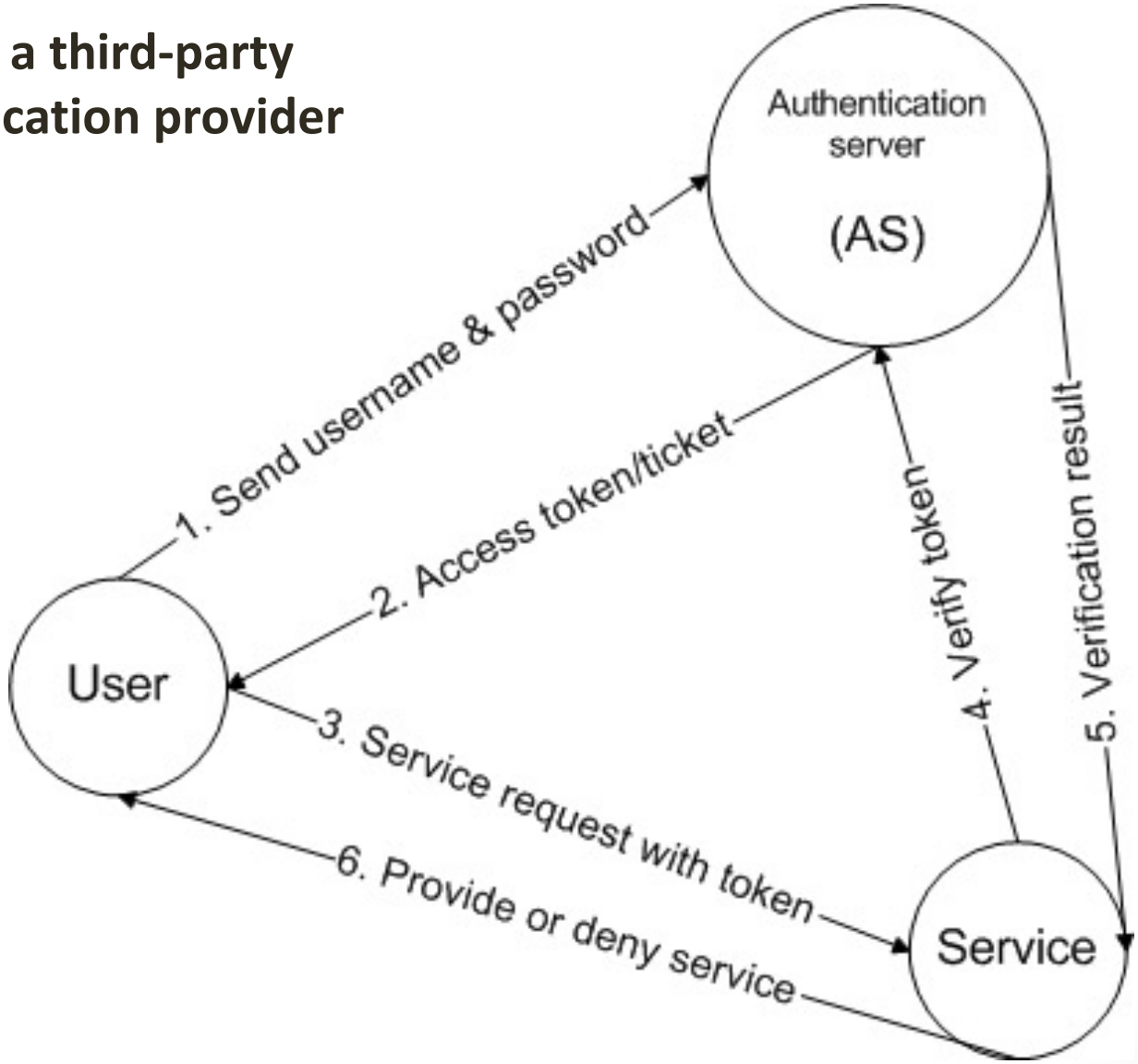
Authentication Mechanisms

Through a third-party authentication provider



Authentication Mechanisms

Through a third-party authentication provider



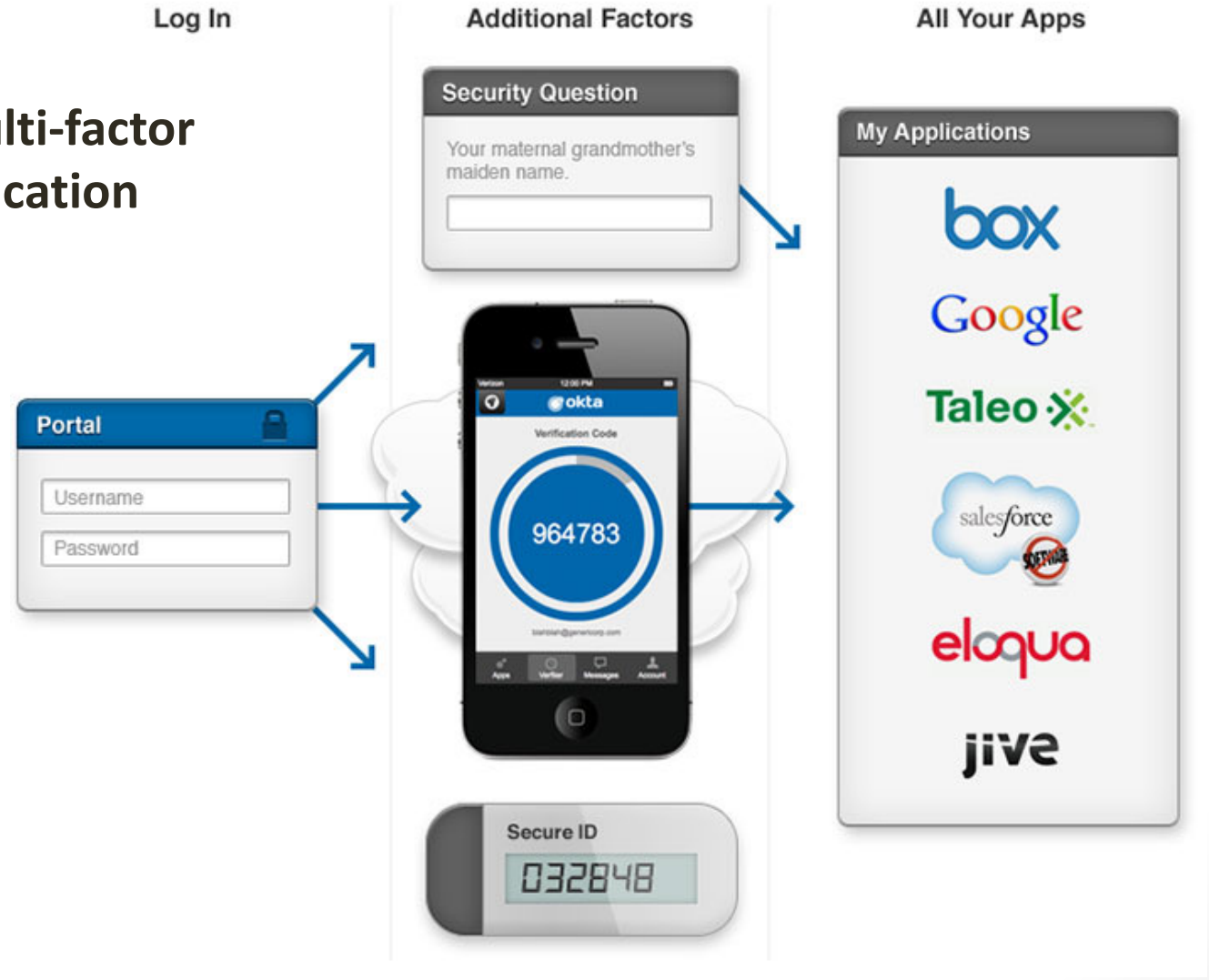
Authentication Mechanisms

With Multi-factor authentication

- Multi-factor authentication no longer depends on a single secret
- It requires multiple authentication factors to complete the process (e.g. two-factor authentication, with username and password authentication on one hand, and a token sent to your cellphone on the other hand)
- Preferably, each factor uses a different channel, hardening the process against a single way of stealing credentials

Authentication Mechanisms

With Multi-factor Authentication



Authentication Mechanisms

With Multi-factor Authentication

What is Multi Factor Authentication - (HAK)

1. Something you **Have**

Credit card, Tokens, Smart Card, device

2. Something you **Are**

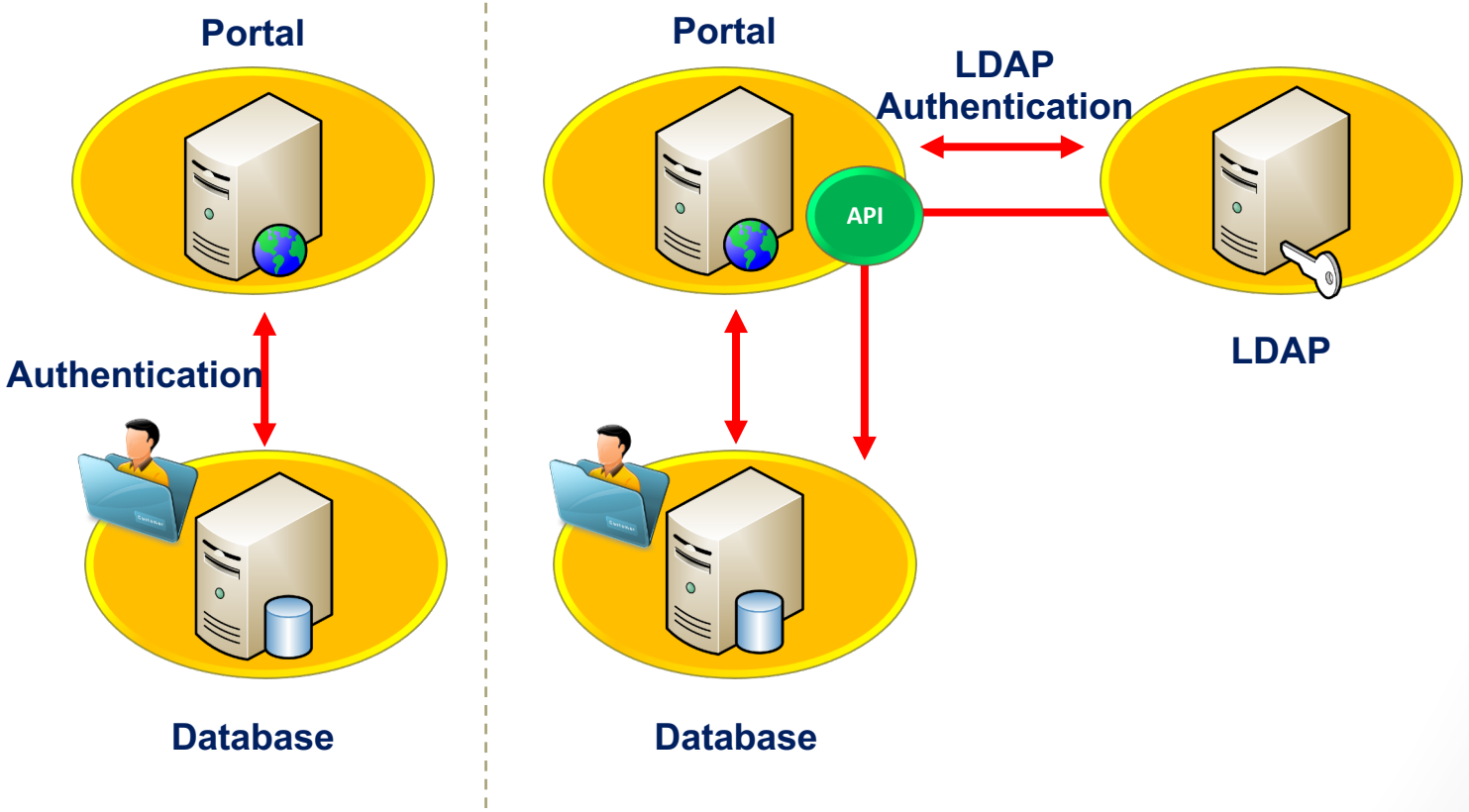
Unique characteristics using Biometrics

3. Something you **Know**

PINS, password, images

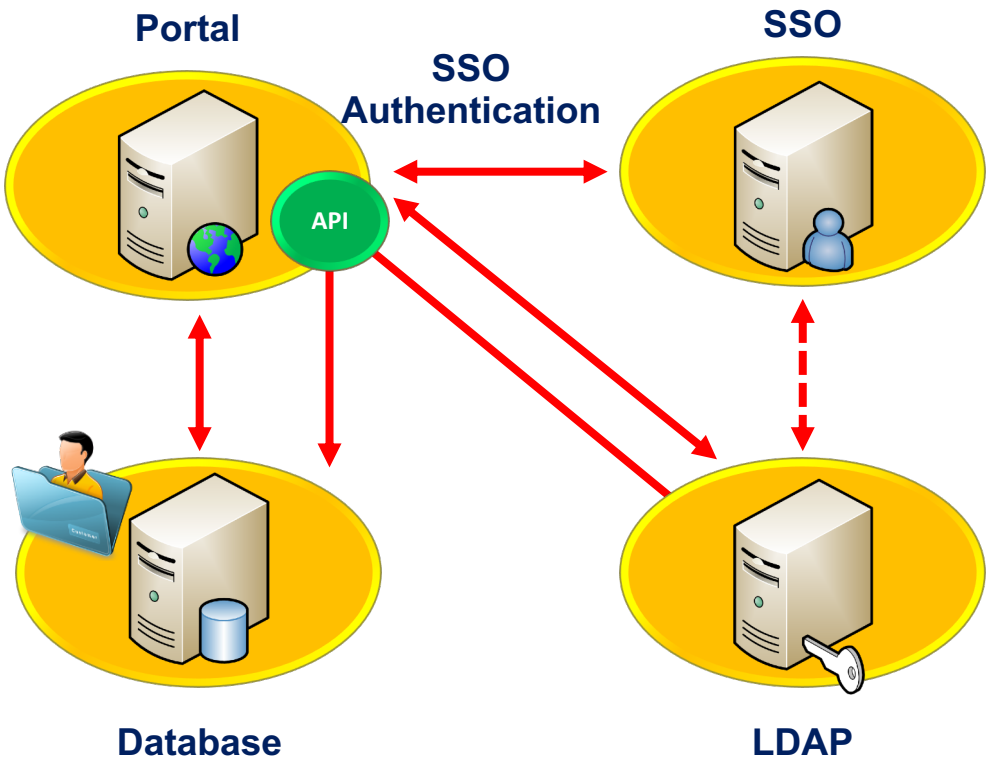
Authentication Models

Based on Portal API



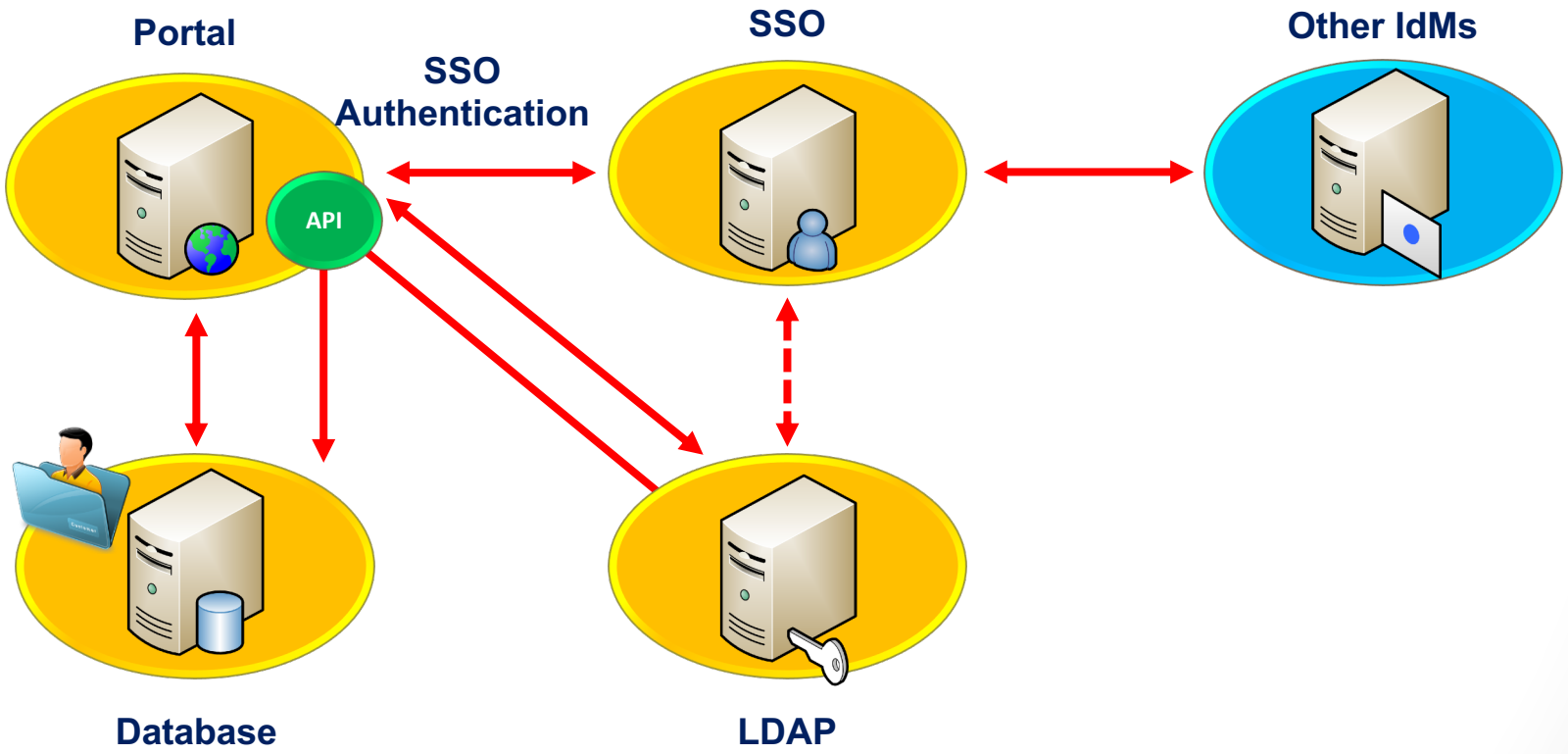
Authentication Models

Based on Portal API & SSO Server



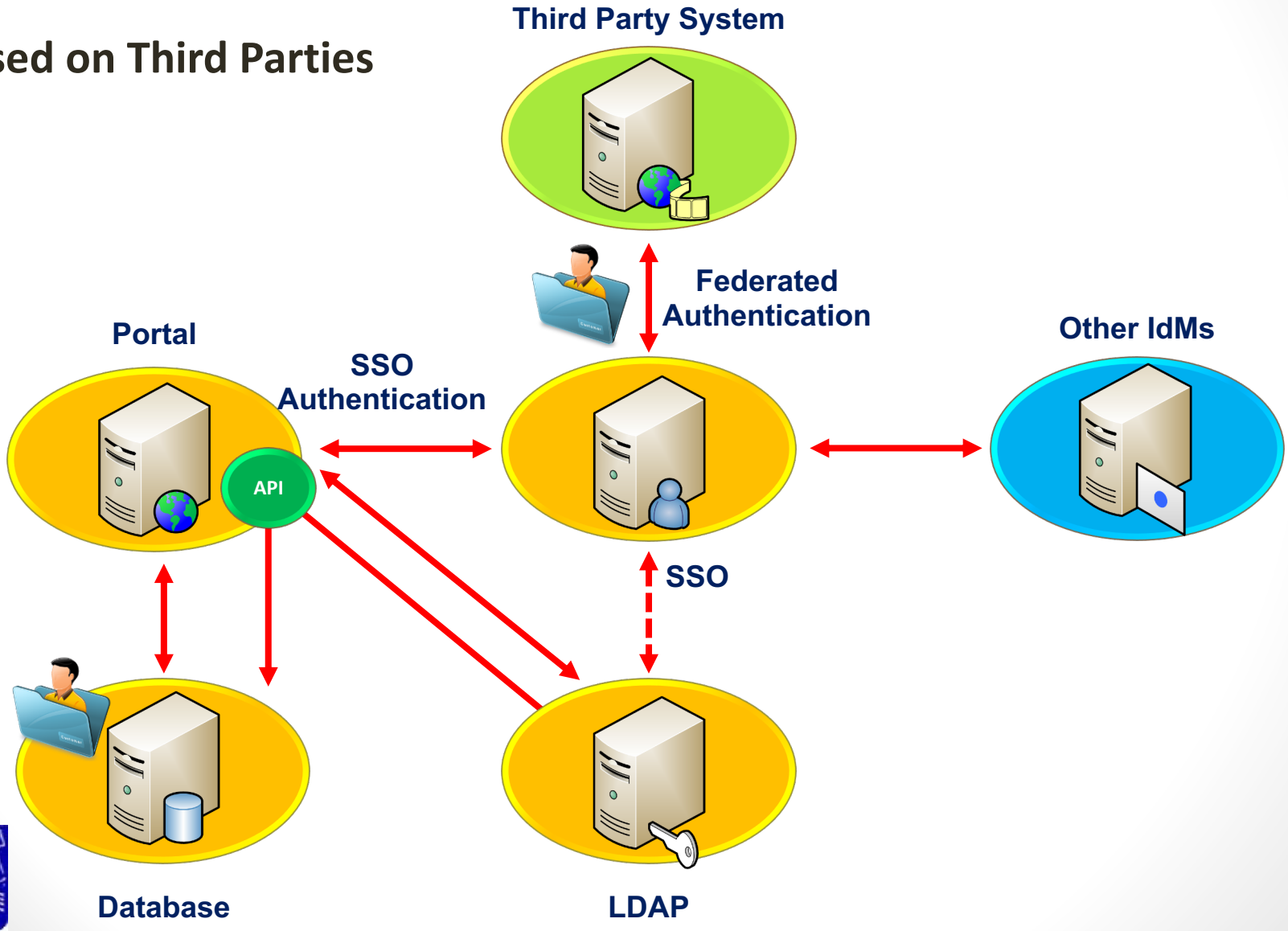
Authentication Models

Based on IDM Solutions

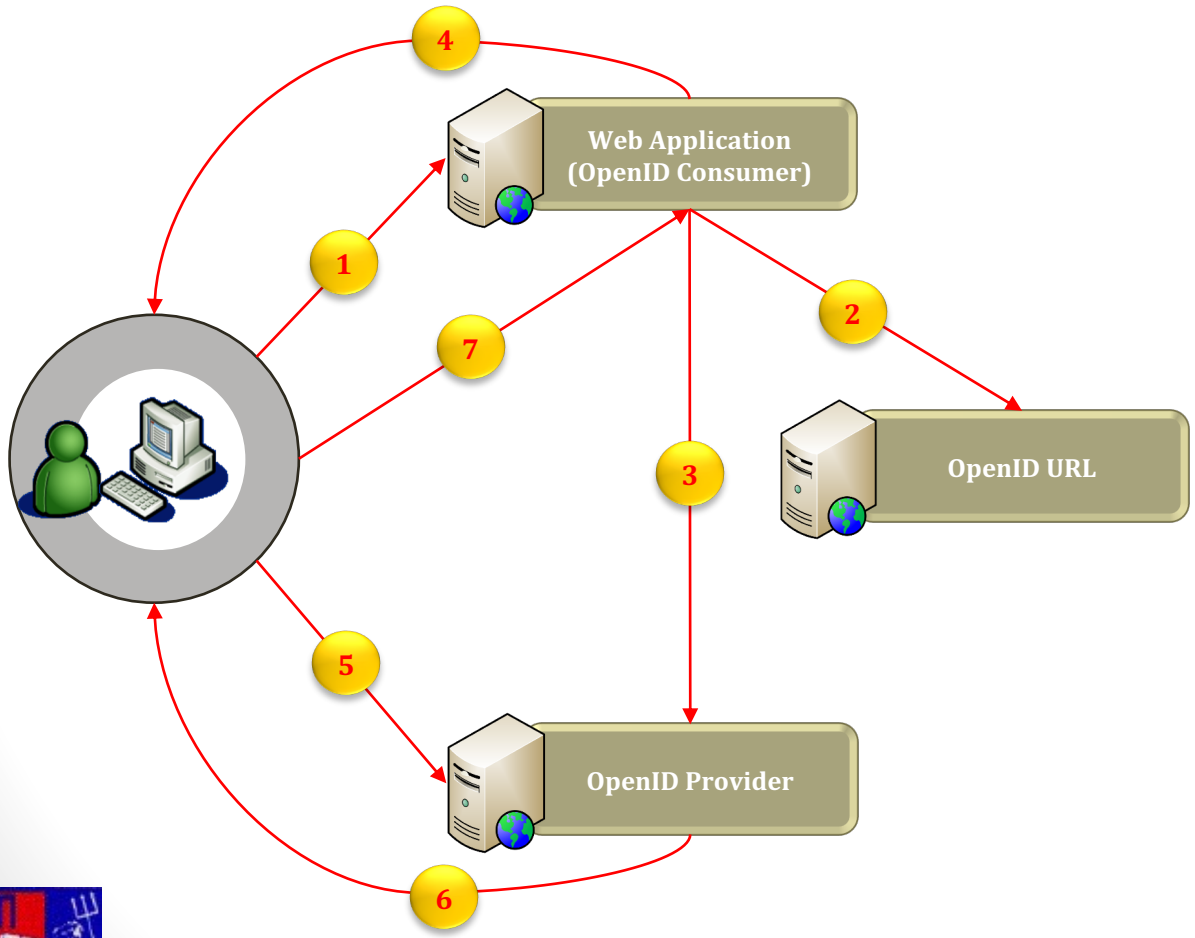


Authentication Models

Based on Third Parties



OpenID Protocol



1. User Posts OpenID URL
2. Discover Provider
3. Generated Shared Secret
4. Redirected to Provider
5. User Logs into Provider
6. Redirect to Consumer
7. Post result of Login & Simple Registration Info

Kerberos

Kerberos

- a protocol for authentication
- uses tickets to authenticate
- avoids storing passwords locally or sending them over the internet
- involves a trusted 3rd-party
- built on symmetric-key cryptography

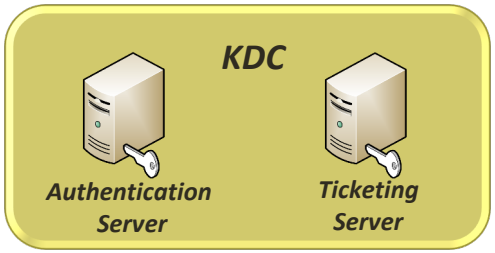
A user owns a **ticket** (proof of identity encrypted with a secret key for the particular service requested – on the local machine)

As long as it's valid, the user is able to access the requested service that is within a Kerberos realm

NOTE: Typically, this is used within corporate/internal environments.

Kerberos Realm

- Admins create realms (Kerberos realms) that will encompass all that is available to access
- A user may not have access to certain services or host machines that is defined within the policy management
- A realm defines what Kerberos manages in terms of who can access what
- User's machine, the Client, lives within this realm, as well as the service or host he wants to request and the ***Key Distribution Center (KDC)***.



Requesting Access

Three Interactions take place:

- User & Authentication Server
- User & Ticket Granting Server
- User & Service or host machine that he wants access to

NOTE 1: Each Interaction has two messages - one that the user can decrypt, and one that the user cannot

NOTE 2: The service/machine the user requests access to, **never** communicates directly with the KDC

NOTE 3: Secret keys are passwords plus info that are hashed

NOTE 4: All secret keys are stored in the KDC database (symmetric)

NOTE 5: The KDC itself is encrypted with a master key to add a layer of difficulty from stealing keys from the database

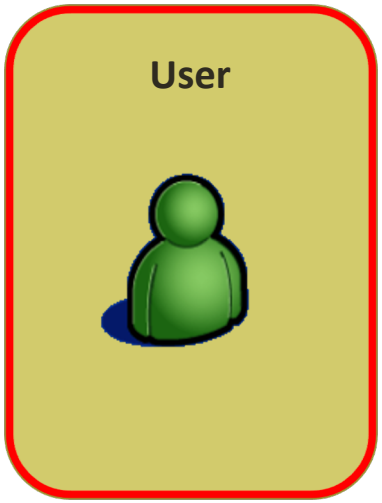
User & Authentication Server

- The user wants to access an HTTP Service
- First he must introduce himself to the Authentication Server
- He logs into his computer
- He initiates that introduction via a plaintext request for a ***Ticket Granting Ticket (TGT)***
- He send all this to the Authentication Server

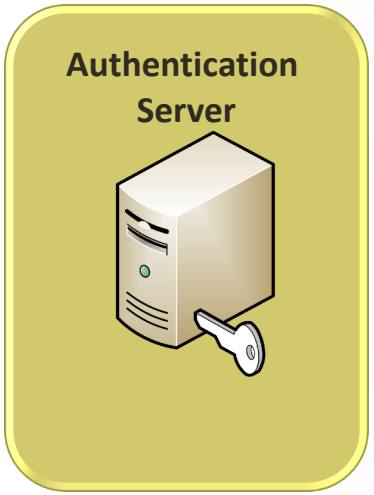
The plaintext message contains:

- Name/ID
- The name/ID of the requested service (in this case, service is the Ticket Granting Server)
- Network address (may be a list of IP addresses for multiple machines, or may be null if wanting to use on any machine)
- Requested lifetime for the validity of the TGT

User & Authentication Server



- Name/ID
- The name/ID of the requested service
- Network address Requested lifetime for the validity of the TGT



User & Authentication Server

- The Authentication Server will check if the user exists in the KDC database (no credentials are checked)
- If there are no errors, it will randomly generate a key called a session key for use between the user and the TGS
- The Authentication Server will then send two messages back to the user
- **Message 1** (encrypted with TGS Secret Key): Name/ID, the TGS name/ID, timestamp, user's network address, lifetime of the TGT, and TGS Session Key
- **Message 2** (encrypted with user's Secret Key): the TGS name/ID, timestamp, lifetime (same as above), and TGS Session Key

NOTE 1: The TGS Session Key is the shared key between the user and the TGS

NOTE 2: User's Secret Key is determined by prompting user's password, appending a salt (made up of user@REALMNAME.COM) and hashing the whole thing.

The user can use it for decrypting the second message in order to obtain the TGS Session Key. If the password is incorrect, then the user will not be able to decrypt the message.

NOTE 3: The user can not decrypt the TGT since he doesn't not know the TGS Secret Key. The encrypted TGT is stored within his credential cache

User & Authentication Server

User

MESSAGE 1
(encrypted with TGS Secret Key)

- Name/ID
- the TGS name/ID
- Timestamp
- user's network address,
- lifetime of the TGT
- TGS Session Key

MESSAGE 2
(encrypted with user's Secret Key)

- the TGS name/ID
- Timestamp
- Lifetime
- TGS Session Key

Authentication Server

User & Ticket Granting Server

- The user has the TGT that he can not read because he does not have the TGS Secret Key to decrypt it. He has however the TGS Session Key
- The user sends two messages
  - He first prepares the Authenticator, encrypted with the TGS Session Key, containing name/ID, and timestamp.
  - He then sends an unencrypted message that contains the requested HTTP Service name/ID he wants access to, and lifetime of the Ticket for the HTTP Service

along with the encrypted Authenticator and TGT to the Ticket Granting Server.

User & Ticket Granting Server

User

MESSAGE 1
(encrypted with TGS Session Key)

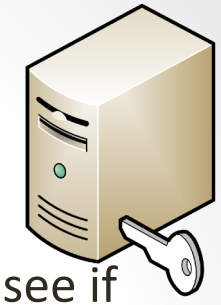
- Name/ID
- Timestamp

MESSAGE 2
(encrypted with user's Secret Key)

- the requested HTTP Service name/ID
- lifetime of the Ticket for the HTTP Service

Ticket Granting Server

User & Ticket Granting Server



- The Ticket Granting Server will first check the KDC database to see if the HTTP Service exists
- If so, the TGS decrypts the TGT with its Secret Key. Since the now-unencrypted TGT contains the TGS Session Key, the TGS can decrypt the Authenticator the user sent
- The TGS will then do the following:
  - compare user's client ID from the Authenticator to that of the TGT
  - compare the timestamp from the Authenticator to that of the TGT
  - check to see if the TGT is expired (the lifetime element),
  - check that the Authenticator is not already in the TGS's cache (for avoiding replay attacks),
  - if the network address in the original request is not null, compares the source's IP address to user's network address (or within the requested list) within the TGT.

User & Ticket Granting Server

- The Ticket Granting Server then randomly generates the HTTP Service Session Key, and prepares the HTTP Service ticket for the user that contains:
  - User's name/ID
  - HTTP Service name/ID
  - User's network address
  - timestamp
  - lifetime of the validity of the ticket
  - HTTP Service Session Key

and encrypts it with the HTTP Service Secret Key.

- Then the TGS sends User two messages.
  - ***Message 1:*** the encrypted HTTP Service Ticket
  - ***Message 2*** (encrypted with the TGS Session Key): HTTP Service name/ID, timestamp, lifetime of the validity of the ticket, and HTTP Service Session Key
- User's machine decrypts the latter message with the TGS Session Key that it cached earlier to obtain the HTTP Service Session Key.

NOTE 1: User's machine can not, however, decrypt the HTTP Service Ticket since it's encrypted with the HTTP Service Secret Key.

User & Ticket Granting Server

User

MESSAGE 1
(encrypted with HTTP Service Secret Key)

- User's name/ID
- HTTP Service name/ID
- User's network address
- timestamp
- lifetime of the validity of the ticket
- HTTP Service Session Key

MESSAGE 2
(encrypted with TGS Session Key)

- HTTP Service name/ID
- Timestamp
- lifetime of the validity of the ticket
- The HTTP Service Session Key

Ticket Granting Server

User & HTTP Service

- The user's machine prepares another Authenticator message that contains:
  - User's name/ID
  - timestamp

NOTE 1: This is encrypted with the HTTP Service Session Key.

- User's machine then sends the Authenticator and the still-encrypted HTTP Service Ticket received from the TGS.

User & HTTP Service

User

MESSAGE 1
(encrypted with HTTP Service Session Key)

- User's name/ID
- timestamp

MESSAGE 2
(encrypted with TGS Session Key)

- HTTP Service name/ID
- Timestamp
- lifetime of the validity of the ticket
- The HTTP Service Session Key

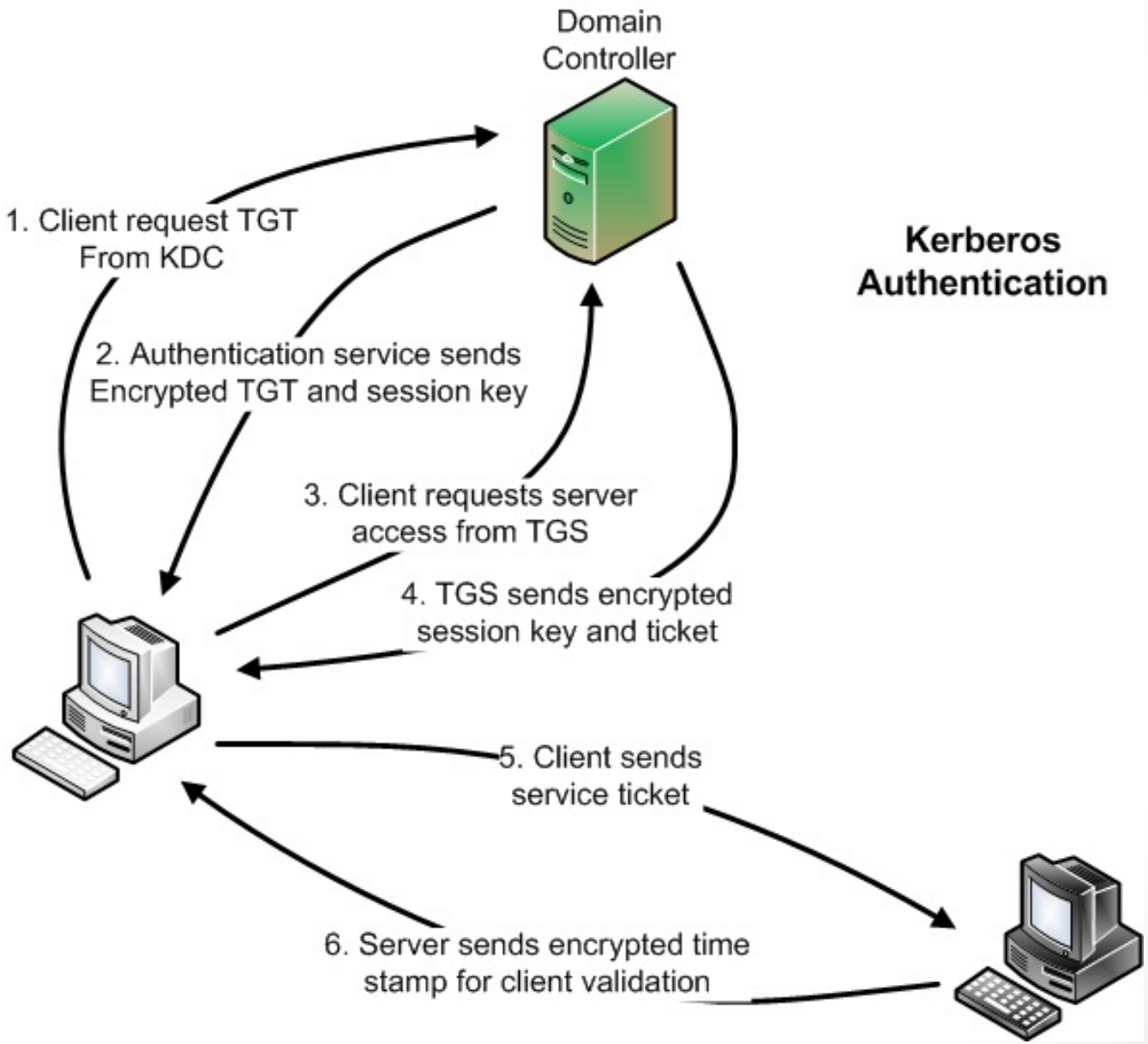
HTTP Service Server

User & HTTP Service

- The HTTP Service then decrypts the Ticket with its Secret Key to obtain the HTTP Service Session Key.
- It uses that Session Key to decrypt the Authenticator message the user sent.
- Similar to the TGS, the HTTP Server will then do the following:
  - compares user's client ID from the Authenticator to that of the Ticket
  - compares the timestamp from the Authenticator to that of the Ticket
  - checks to see if the Ticket is expired (the lifetime element)
  - checks that the Authenticator is not already in the HTTP Server's cache (for avoiding replay attacks)
  - if the network address in the original request is not null, compares the source's IP address to user's network address (or within the requested list) within the Ticket.
- The HTTP Service then sends an Authenticator message containing its ID and timestamp in order to confirm its identity to the user and is encrypted with the HTTP Service Session Key.
- User's machine reads the Authenticator message by decrypting with the cached HTTP Service Session Key, and knows that it has to receive a message with the HTTP Service's ID and timestamp.
- The user has been authenticated to use the HTTP Service

User & HTTP Service

Overview


Domain Controller
1. Client request TGT From KDC
2. Authentication service sends Encrypted TGT and session key
3. Client requests server access from TGS
4. TGS sends encrypted session key and ticket
5. Client sends service ticket
6. Server sends encrypted time stamp for client validation
Kerberos Authentication

Authorization

- The functionality of a web application is typically shared among a large number of users, making authorization crucial to ensure that a user has the correct permissions to access certain features or specific data
- Authorization checks need to be performed both on the available features, but also on the data used in these features
- It prevents an authorized action on unauthorized data objects (such as making a wire transfer from an account that does not belong to the user)

NOTE: Naturally*, authorization is closely coupled to authentication and session management.*

Storage

- Dynamic web applications typically require backend data storage, to keep track of users and their data
- The type of storage depends on the type of application
- Database storage is the most popular choice, closely followed by file-system storage

NOTE : *The backend storage is typically only accessible through the business logic, which acts as the frontend*

Ευχαριστώ
για την προσοχή σας!!

Επικοινωνία: karant@unipi.gr

Ενημέρωση: http://gunet2.cs.unipi.gr/eclass/