Eötvös Loránd University
Faculty of Informatics

An Improved Java-based Single Sign-on Solution

Péter Hudoba
Assistant professor at ELTE
Andreas Peter
Associate Professor at University of Twente
Szabó Áron
Security Consultant at E-GROUP

Zihan Ni
Computer Science

Budapest, 2020

Acknowledgement

First of all, I wish to express my deepest gratitude to my academic supervisor, Péter Hudoba, for guiding my graduation thesis with kindness and patience. He taught me a lot of specific research skills in the early stage. Then he spent much time reading through each draft and provided me with inspiring advice, which profoundly improved my understanding of academic writing. In this special period, COVID-19 has brought us many unexpected inconveniences, thanks so much for his understanding.
Second, I would like to pay my special regards to E-group and my industry supervisor Szabó Áron provided me a meaningful research direction and practical goals.
I am also deeply indebted to my coordinator Kiss Eszter and all the teachers in ELTE for their direct and indirect help.
Last, my thanks would go to my beloved family and boyfriend Yao for their continuous support and encouragement.

Contents

Chapter 1

Overview

1.1 Scenario

In the past two decades, the rapid development and popularization of the Internet and mobile Internet technologies have spawned a new digital economy. People are more accustomed to the benefits of information technology. On September 29, 2018, the electronic IDentification and Authentication Services regulation(eIDAS)[EU14] promulgated by the EU in 2014 came into effect. With the support of eIDAS, EU citizens can now use their electronic ID (eID) to conduct a series of credible cross-border activities:

- setting up a business in another Member State
- bidding to on line call for tender
- immigration procedures in the EU
- complete the registration of transnational education online
- complete tax declaration online
- online transactions sign electronic contracts
- online bank card opening, lending, etc.

But new digital methods also bring a lot of risks like online fraud, identity fraud, and private information disclosure. Against this background, the second edition of the Payment Services Amendment Act (PSD2)[EU19] is passed in EU in 2015. It focuses on the financial industry and the fight to reduce costs with also protecting consumers in the e-commerce industry by reducing fraud.

The cross-border activities inside EU (between Member States) based on these specifications (eIDAS and PSD2) became available recently for public service providers and also private service providers will be involved soon. Fortunately, similar specifications exist for connecting EU with other regions, such as China in Asia. These international specifications are maintained by GSMA (GSMA Mobile Connect) and applied by Mobile Operators[Con08a]. These specifications cover both user authentication and identification based on different assurance levels and exchange of KYC (Know-Your-Customer) attributes [Con08b].

These specifications are implemented by some vendors [Con08c]whose solutions are applied by Mobile Operators. E-Group is one of the registered vendors. E-Group implementation (as part of its IDX Identity Exchange Platform product) was born several years ago in the early stage of GSMA Mobile Connect specification, but now E-Group shall update its codes in order to start integration with e.g. GSMA Mobile Connect enabled Chinese Mobile Operators (such as China Unicom, China Telecom, China Mobile).

1.2 Goals

To make it more convenient for users to access E-group's platform without multiple logins after login in trusted systems or webpages within the company, E-group already built a Java-based single sign-on solution that supports various user authentication protocols.

Not long ago, E-group decided to experimentally add the authentication supported by GSMA Mobile Connect into the authentication part in the existed single sign-on solution. In addition to optimizing the company's internal login system, this also provides a reference basis for applying other services offered by other specifications in GSMA Mobile Connect in the future.

Our goal is to design a new system for single sign-on and multi-point use, which integrates GSMA Mobile Connect into authentication. Only authenticated users can access the E-group resource system. It not only facilitates user login and management but also effectively improves system security performance and ensures system resources are used efficiently.

Chapter 2

Single Sign-On

2.1 Single System Login

2.1.1 HTTP Protocol

Web application adopts client/server architecture, and HTTP as the communication protocol. HTTP is a stateless protocol. Each request from the client will be processed independently by the server, and will not be associated with the previous or subsequent requests. This process is illustrated in the following figure. There is no connection between the three request/response pairs.

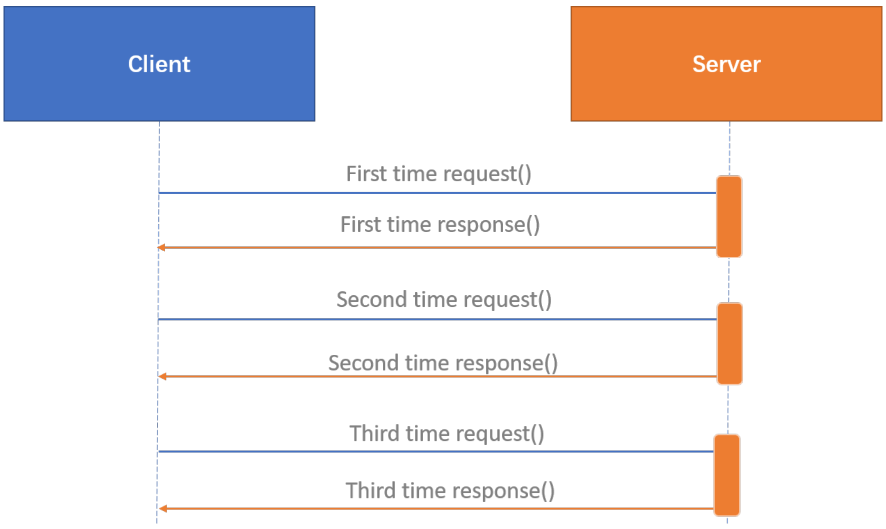


Figure 2.1: HTTP Protocol

But it also means that any user can access the server resources through the client. If you want to protect specific resources on this server, the client's requests must be limited by authentication: respond to the legitimate request and ignore the illegal request. In this case, the request status must be transparent during this period. Since the HTTP protocol is stateless, the process server and client maintains a state together, and this is called the session mechanism[Ana18].

2.1.2 Session

HTTP sessions is an industry standard feature that allows Web servers to maintain user identity and to store user-specific data during multiple request/response interactions between a client application and a Web application. HTTP sessions preserves:

- Information about the session itself (session identifier, creation time, time last accessed, etc.)
- Contextual information about the user (client login state, for example, plus whatever else the Web application needs to save)

Following is how the session mechanism works:

The client requests the server for the first time, the server creates a session and sends the session id as part of the response to the client. The client stores the session id and brings the session id in the subsequent second and third requests. The server obtains the session id in the request to know whether it is the same user. This process is shown in the figure below. Subsequent requests are associated with the first request.

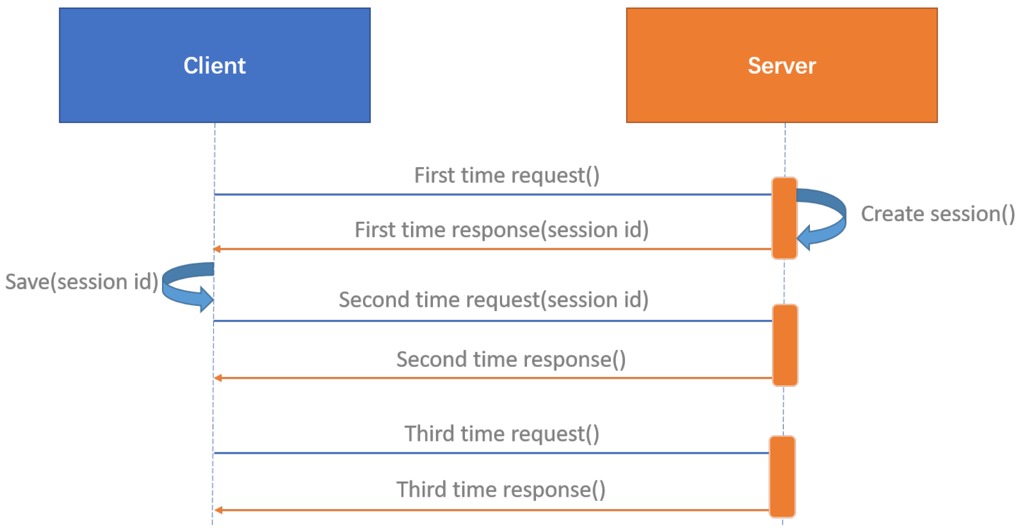


Figure 2.2: Session

2.1.3 Session-Management Methods

1. Based on Server-Session

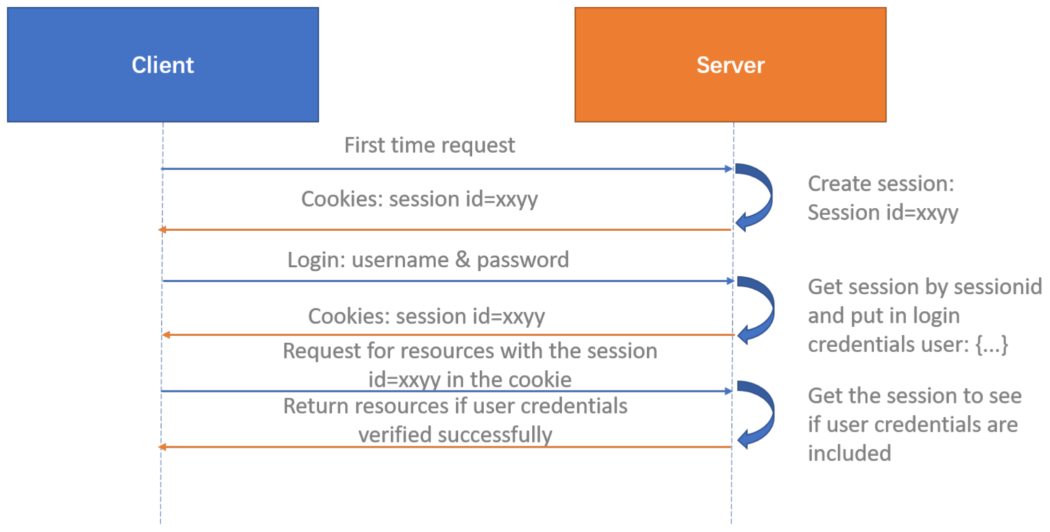


Figure 2.3: Server-Session Based SSO

Steps:

1. The server session is created by the server when the user accesses the application for the first time. The server assigns a unique session id to each session to ensure that each user has a different session.

2. The session id will be returned to the user's client through the cookie after it is created.

3. When the user sends a request to the server for the second time. Later, the session id will be passed back to the server through the cookie, so that the server can find the session corresponding to the user.

The session usually has a set expiration time, such as 2 hours. When it expires, the server will destroy the previous session and create a new session. But as long as the user sends a new request to the server within the expiration time, the server will usually extend his corresponding session's expiration time according to the current request time for another 2 hours.

When the user actively logs out, the login credentials in its session will be cleared. Therefore, before the user logs in or after logging out, or when the session object becomes invalid, it is impossible to obtain the required login credentials.Session is used to manage the login process only when there is validated login included. In this case, users can get the login credentials from this session if they have the corresponding session id.

Benefits:

Security is guaranteed.

The medium that maintains the session between the client and the server is always just a session id string, as long as the string is random enough, the attacker cannot easily impersonate to be others. Unless Cross-Site Request Forgery(CSRF) or session hijacking is possible.

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application[Kir].
Through CSRF, attacker can simulate user actions on one website (the target site) from another website (the attacking site) [Bla07].
Session hijacking, sometimes also known as cookie hijacking is the exploitation of a valid computer session—sometimes also called a session key—to gain unauthorized access to information or services in a computer system. In particular, it is used to refer to the theft of a magic cookie used to authenticate a user to a remote server. It has particular relevance to web developers, as the HTTP cookies used to maintain a session on many web sites can be easily stolen by an attacker using an intermediary computer or with access to the saved cookies on the victim's computer. After successfully stealing appropriate session cookies an adversary might use the Pass the Cookie technique to perform session hijacking[tec].

Under these circumstances, the attacker can to perform actions of the attacker's choosing using the user's authenticated session. Even if the impersonation succeeds, the attacker can only get the protected resources when there is a valid login credentials in the session if the user logged successfully.

Disadvantages:

1. The session is stored in the web server, so when the user has more online information simultaneously, the session information will occupy a lot of memory.

2. When the application is deployed in a cluster, you will encounter the problem of sharing sessions between multiple web servers. Since the session is created by a single server, the server that handles the user request highly possible is not the server which created the session.

3. When multiple applications want to share a session, cross-domain problems will also be encountered. Different applications may deploy different hosts, and cookies cross-domain may be a problem.

Solutions:

1. To deal with the first and second problem, it is a good choice to use an intermediate server such as Redis to manage the session operation like addition, delete or modification. It can reduce the burden on the web server, and also solve the problem of sharing sessions between different web servers.

2. Also, a configured load balancer can help transfer user information from one service to other servers which can deal with the second problem.

3. For the last problem, possible solution is to implement cross-domain access for session id in cookie, which can be achieved but could be troublesome because developers have to configure both front-end and back-end.

In this way, the session id is attached as a parameter in each request so the server can naturally parse the parameters to obtain it. Then the server can use the session id to determine whether it comes from the same session.

Obviously, request with parameter is not a reliable method. There is another way that the client maintains this session id by itself. Let the client automatically sends the session id every time an HTTP request is sent. This is why we need cookie here. An HTTP cookie (also simply called cookie) is a small piece of data sent from a website and stored on the user's computer by the user's client while the user is browsing. The data is stored in the form of "key/value". When the client sends an HTTP request, the cookie information is automatically attached.

2. Cookies-Based

Since the former method will increase the burden on the server and the complexity of the architecture, someone later came up with a solution to directly keep the user's login credentials at the client side. When the user successfully logs in, login credentials will be written into the cookie with a set validity period. Further requests directly verify the cookie to see whether the login credential exists and check the validation of the credentials.

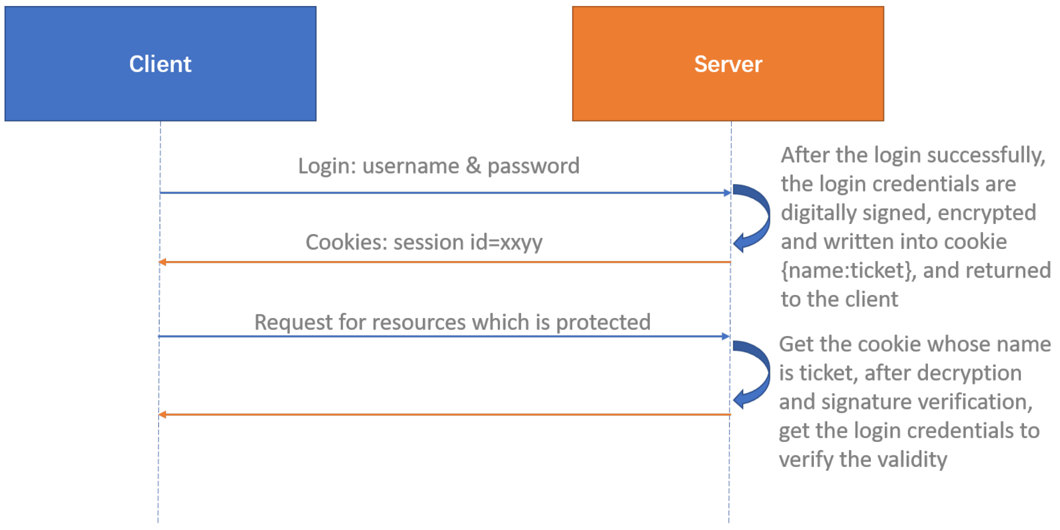

Client
Server
Login: username & password
Cookies: session id=xxyy
Request for resources which is protected
After the login successfully, the login credentials are digitally signed, encrypted and written into cookie {name:ticket}, and returned to the client
Get the cookie whose name is ticket, after decryption and signature verification, get the login credentials to verify the validity

Figure 2.4: Cookies-Based SSO

Steps:

1. The user initiates a login request, and the server verifies whether the user meets the login conditions based on the identity information such as the incoming user password. If so, a login credential is created based on the user information.

2. The server signs the login credential created in the previous step and then encrypts it with a symmetric encryption algorithm. Put the string into a cookie. The name of the cookie must be fixed (such as ticket) because later you have to get the cookie value based on this name.

3. When the user sends further requests, the server obtains the relevant cookie value according to the cookie name of the login credentials stored in the previous step. Do decryption first, and then perform digital signature authentication. If these two steps succeed, you can get the original login credentials; if any of of them fail, it means that the login credentials are illegal.

4. Compare the expiration time of this credential with the current time to determine whether the credential has expired. If it expires, the user needs to log in again; if it does not expire, the request is allowed to continue.

Benefits:

It realizes the statelessness of the server. The server only needs to be responsible for creating and verifying the login cookie, without maintaining the user's state information.

The problem of user session sharing - the second problem in server-session based, it can also be solved here:

- If it is the same application deployed in cluster, the code for verifying the login credentials are also the same. No matter which server processes the user requests, they can always obtain the login credentials in the cookie for verification.
- If they are different applications deployed in cluster, as long as each application contains the same login logic, then they can also easily share the session.

But in this case, the key file or key string used in the digital signature and encryption and decryption needs to be shared among different applications. In short, the algorithm needs to be completely consistent.

Since this method stores the login credentials directly at the client side and requires cookies to be passed around, its shortcomings are also obvious:

Disadvantages:

1. Cookies have a size limit and cannot store too much data. If there are too many messages stored in the login credentials or the string after the encryption signature is too long, it will cause other problems
2. There might also be cross-domain issues.
3. There is a risk of the attacker impersonate user by cookie poisoning.

Cookie Poisoning is the act of manipulating or forging session cookies for the purpose of bypassing security measures and achieving impersonation and breach of privacy. By forging these cookies, an attacker can impersonate a valid client, and thus gain information and perform actions on behalf of the victim. Or attackers can use forged cookies to trick a server into accepting a new version of the original intercepted cookie with modified values.

3. Token-based

In terms of process and implementation, this method is not much different from the cookie-based method. Except for the token is attached in all the further requests in the form of a URL parameter or HTTP header in token-based mechanism instead of parsing cookies in the cookies-based mechanism. Then the server can directly obtain the token from the

HTTP header or URL for verification after receiving the request:

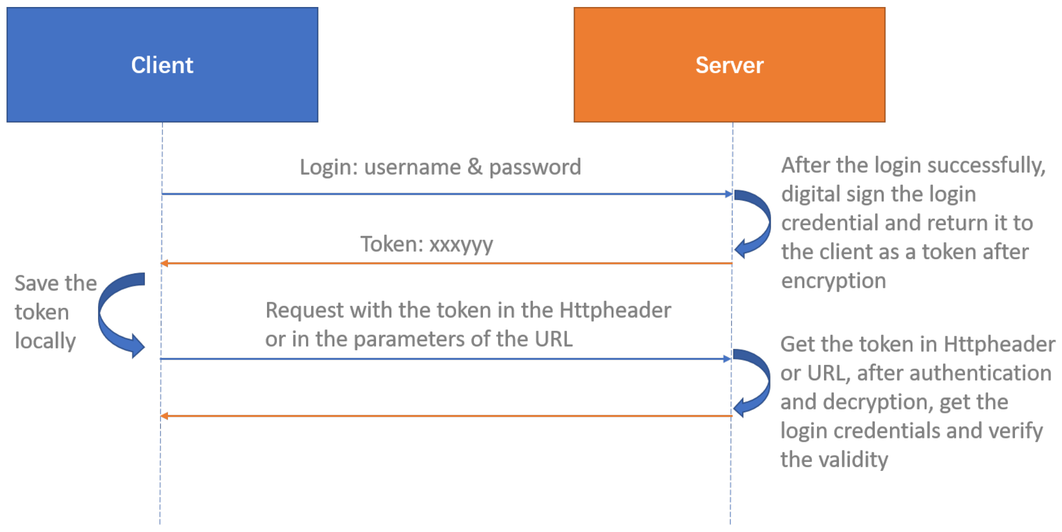

Client
Server
Login: username & password
After the login successfully, digital sign the login credential and return it to the client as a token after encryption
Token: xxxyyy
Save the token locally
Request with the token in the Httpheader or in the parameters of the URL
Get the token in Httpheader or URL, after authentication and decryption, get the login credentials and verify the validity

Figure 2.5: Token-Based SSO

Steps:

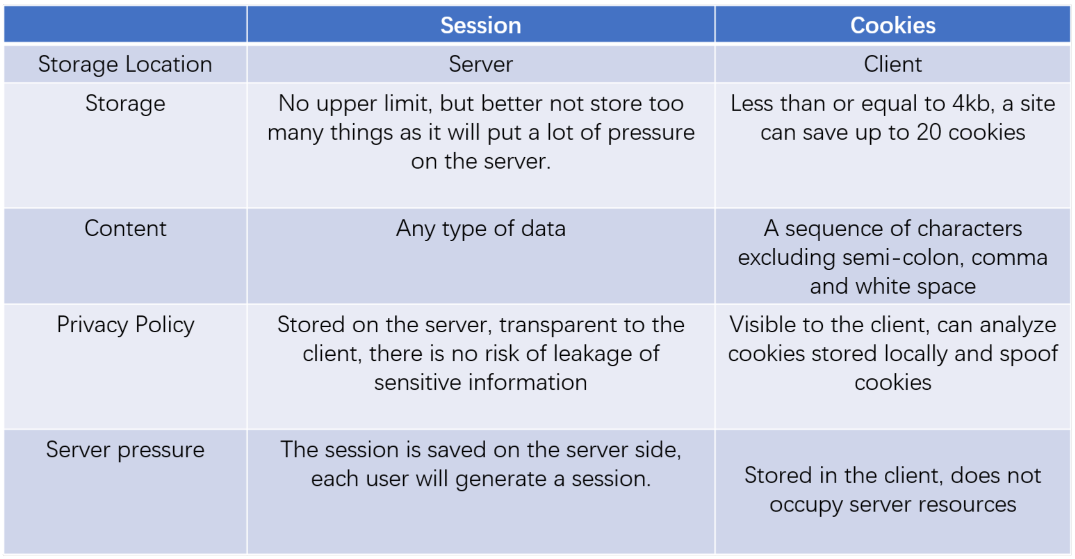
1. When the client requests the service for the first time, the server authenticates the user.
2. Pass the authentication, encrypt the user information to form a token, and return it to the client as the login credential.
3. Each subsequent request, the client carries an authenticated token. The service decrypts the token to determine whether it is valid.

In this way, the token is not passed through the cookie, but every time it is requested, the token is actively added to the HTTP header or in the parameters of the URL.

Benefits:

- The client request does not depend on the server information, and any multiple requests do not need to access the same service
- The server cluster and state are transparent to the client
- The server can be arbitrarily migrated and scaled
- Reduce server storage pressure

4. Comparision



	Session	Cookies
Storage Location	Server	Client
Storage	No upper limit, but better not store too many things as it will put a lot of pressure on the server.	Less than or equal to 4kb, a site can save up to 20 cookies
Content	Any type of data	A sequence of characters excluding semi-colon, comma and white space
Privacy Policy	Stored on the server, transparent to the client, there is no risk of leakage of sensitive information	Visible to the client, can analyze cookies stored locally and spoof cookies
Server pressure	The session is saved on the server side, each user will generate a session.	Stored in the client, does not occupy server resources

Figure 2.6: Session and Cookies

The advantages of token-based over session-based:

1. Stateless. The server does not need to store the token, only need to verify whether the token information is correct. But the session needs to be stored on the server.

2. Each request has a signature to prevent monitoring and replay attacks, while the session must rely on the link layer to ensure communication security.

3. The token has a natural advantage in CSRF defense. In server-session based session-management, the authentication data (session id in cookies) is automatically carried by the client and sent to the server. With this feature, the attacker can perform a CSRF attack. But here, the token is added to the request as dynamic parameters that will not be easily leaked.

The Advantages of token-based over cookies-based:

1. Support cross-domain access with easily attach the token in the HTTP header.

2. More suitable for mobile terminals (Android, iOS, applets, etc.). Native platforms like this do not support cookies sharing well. But token can be used as long as the client can store it. Hence, the token also has advantages on the mobile terminal.

2.2 The Complexity of Multi-systems

The web system has evolved from a single long-term system to an application group comprising multiple systems. With so many systems, users would be frustrated if they have to log in or log out individually, as shown in the following figure.

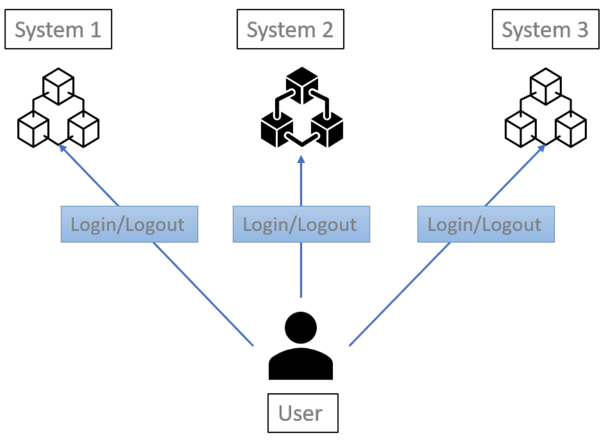


Figure 2.7: Log In and Out Individually

The web system is developed from a single system to an application cluster composed of multiple systems. The system, not the user, should bear the complexity. No matter how complicated the web system is, it is a unified whole for the user. That is to say, the user's access to the entire application cluster of the web system is the same as that of a single system. One time login/logout will be sufficient.

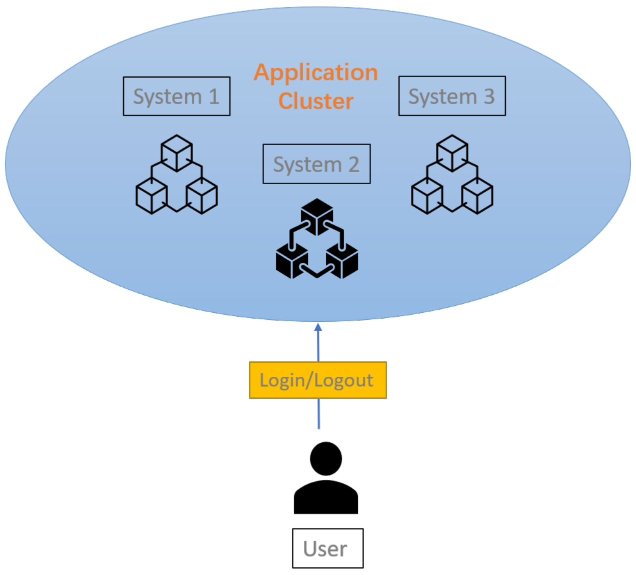


Figure 2.8: Log In and Out Individually

Although the single system login solution is perfect, it is no longer applicable to the multi-system application cluster. Because the core of the single system login solution is the cookie, which carries the session id and maintains the session state between the client and the server. But cookies are limited. This limitation is the domain of the cookie (usually corresponding to the domain name of the website). When the client sends an HTTP request, it automatically carries the cookie matching the domain, and not all cookies.

In this case, why not unify the domain names of all subsystems under a top-level domain name in the application cluster, such as "*.sso.com," and then set their cookie domain to "sso.com"? Theoretically, it works. Many multi-system logins in the early time used this way of sharing cookies with the domain name.

However, viability does not mean it is good. There are many limitations to the way of cookies sharing. First, the domain name of the application cluster must be unified. Second, the technology used by each system in the application cluster(at least the web server) must be the same. Otherwise, the key value of the cookie is different, so the session cannot be maintained in this way. Also, cookies sharing does not support cross-language platform.

Therefore, we need a brand-new login method to realize the login of the multi-system application cluster, which is the single sign-on solution we are going to talk about in the next section.

2.3 What is Single Sign-on

The definition of single sign-on(SSO) is that you only need to log in once to access other mutually trusted application systems in multiple application systems.

Compared to single-system login, SSO requires an independent authentication center. Only the authentication center can accept the user's username, password, and other security information used to authenticate the identity. Other systems do not provide the login service but only accept indirect authorization from the certification center.

The authentication center works differently based on the session-management method it applies.The authentication center works differently based on the session-management method it applies (more detail in Chapter 2.2). Compared to traditional authentication with session id or cookies, token mechanism stands out because of these great advantages:

- Stateless and scalable

Tokens stored on the client are stateless and extensible. Based on this stateless and non-storage session information, the load balancer can transfer user information from one service to the other servers.

- Security

Sending a token instead of sending a cookie in the request can prevent CSRF. Even if the client uses cookies to store tokens, cookies are only a storage mechanism and not for authentication. Reduce session operation since there is no information in the session.

2.4 SSO with Token Authentication

2.4.1 Log In

In token mechanism, indirect permission is achieved through tokens. After the SSO authentication center authenticates the user's identity successfully, an authorization token is created. The token is sent to the subsystem which sends request as a parameter. Authorization can be used to create a local session. The local session login method is the same as the single system login method.

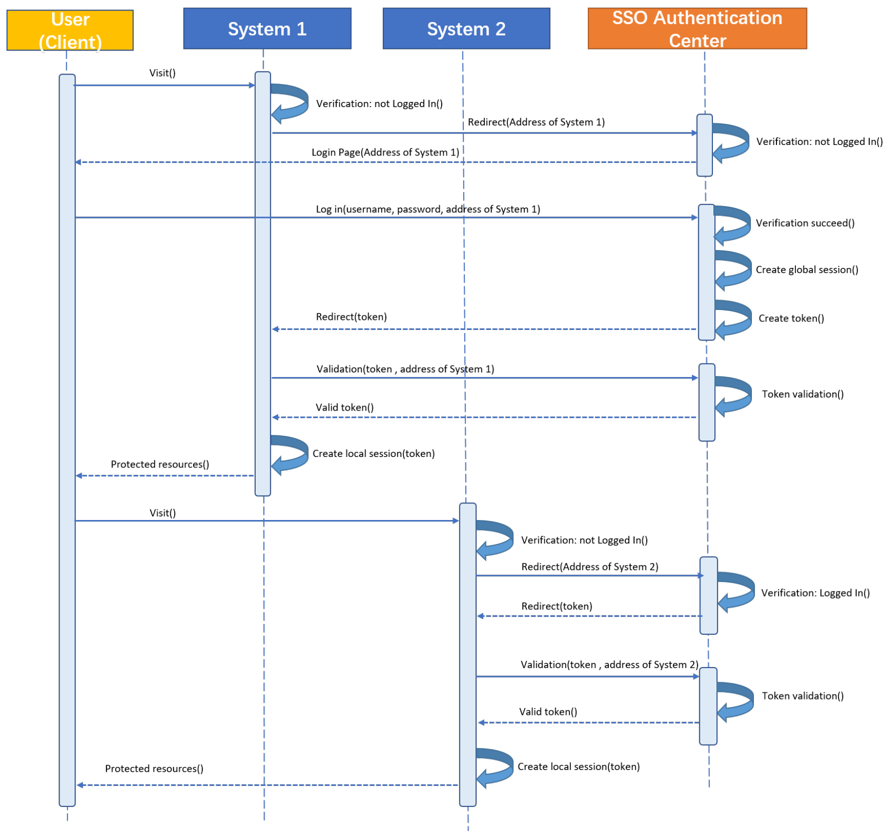

User (Client)
System 1
System 2
SSO Authentication Center
Visit()
Verification: not Logged In()
Redirect(Address of System 1)
Verification: not Logged In()
Login Page(Address of System 1)
Log in(username, password, address of System 1)
Verification succeed()
Create global session()
Create token()
Redirect(token)
Validation(token , address of System 1)
Token validation()
Valid token()
Create local session(token)
Protected resources()
Visit()
Verification: not Logged In()
Redirect(Address of System 2)
Verification: Logged In()
Redirect(token)
Validation(token , address of System 2)
Token validation()
Valid token()
Create local session(token)
Protected resources()

Figure 2.9: Single Sign-On Overview

A brief description to the above figure:

1. The user wants to access the protected resources on System 1. When System 1 finds that the user is not logged in, it jumps to the SSO authentication center and takes its address as a parameter.

2. The SSO authentication center found that the user was not logged in, and returns the user to the login page.

3. The user enters the username and password to submit a login application.

4. The SSO authentication center verifies the user information, creates a session between the user and the SSO authentication center, called a global session, and creates an authorization token.

5. The SSO authentication center jumps back to the initial request address with the token (System 1).

6. System 1 gets the token and visits the SSO authentication center to verify whether the token is valid.
7. The SSO authentication center verifies the token and returns it.
8. System 1 uses this token to create a session with the user, called a local session, and returns the protected resource.
9. The user wants to access to the protected resources of System 2
10. System 2 finds that the user is not logged in, jumps to the SSO authentication center, and takes its address as a parameter.
11. The SSO authentication center finds that the user has logged in, jumps back to the address of system 2, and attaches the token.
12. System 2 gets the token and goes to the SSO authentication center to verify whether the token is valid.
13. The SSO authentication center verifies the token and returns it. The registration system 2.
14. System 2 uses the token to create a local session with the user and return the protected resource.

After the user log-in is successful, a session will be established with the SSO authentication center and each subsystem. The session established between the user and the SSO authentication center is called a global session, and the session established between the user and each subsystem is called a local session. After the local session is established, the user's request to access the protected resources of the subsystem will no longer pass the SSO authentication center[Ana18].

The global session and the local session have the following constraints:

- The local session exists; the global session must exist.
- The global session exists; the local session does not necessarily exist.
- The global session is deleted; the local sessions must be deleted.

2.4.2 Log Out

Single sign-on naturally also requires single log-out. Log out in one subsystem, and all subsystem sessions will be deleted, as shown in the following figure.

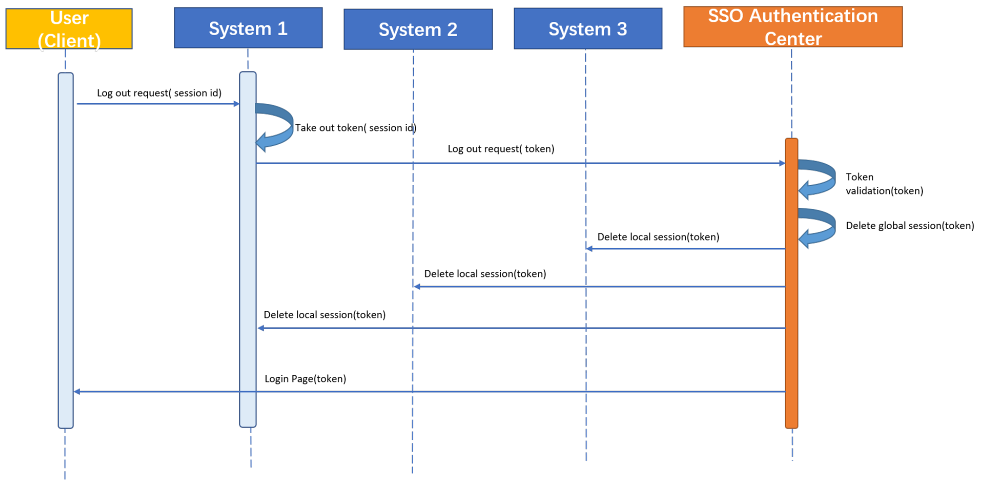

User (Client)
System 1
System 2
System 3
SSO Authentication Center
Log out request(session id)
Take out token(session id)
Log out request(token)
Token validation(token)
Delete global session(token)
Delete local session(token)
Delete local session(token)
Delete local session(token)
Login Page(token)

Figure 2.10: Log Out

The following is a brief description of the above figure:

1. The user initiates a logout request to System 1.
2. System 1 gets the token based on the session id established between the user and System 1 and initiates a logout request to the SSO authentication center.
3. The SSO authentication center verifies that the token is valid, delete the global session, and takes all system addresses registered with this token.
4. The SSO authentication center initiates a logout request to all systems.
5. Each registration system receives the cancellation request of the SSO authentication center and deletes the local session.
6. SSO authentication center redirects the user to the login page.

2.4.3 Token Types

Generally speaking, there are three main types of tokens:

Customized token: Developer's customized token based on business logic

JWT: JSON Web Token, a token specification defined in RFC 7519

Oauth2.0: An authorization specification defined in RFC 6750, but this is not actually a token.

2.5 Implement SSO with JWT

2.5.1 What is JWT

A JSON Web Token(JWT) is a JSON style lightweight authorization and identity authentication specification that can realize stateless and distributed Web application authorization. It is a specific implementation of token in Java.

The information in the JWT can be verified and trusted because it is digitally signed using a shared secret key or a public & private key pair.

When the sender creates the JWT, the sender will have:

- the information is written using a defined structure/encoding method;
- signed the JWT.

When the receiver gets the JWT, the receiver should:

- verify the JWT, by verifying the signature;
- decode the information in the JWT[Conc].

JWT Data Format

The JWT generally has three parts separated with dots, e.g.:

```
eyJ0eXAiOiJKV1QiLCJhbGciOIJIUzI1NiJ9.
eyJpc3MiOiJKb2huIERvZSIsImlhdCi6bnVsbCwiZXhwIjoxNTEzMTcxMzcwLCJhdWQiO
iIiLCJzdWIiOiIifQ.
Bxp8_MNg_kzFYat4n8OGOXpdP2ZPM9PWBTREp1W4Imo
```

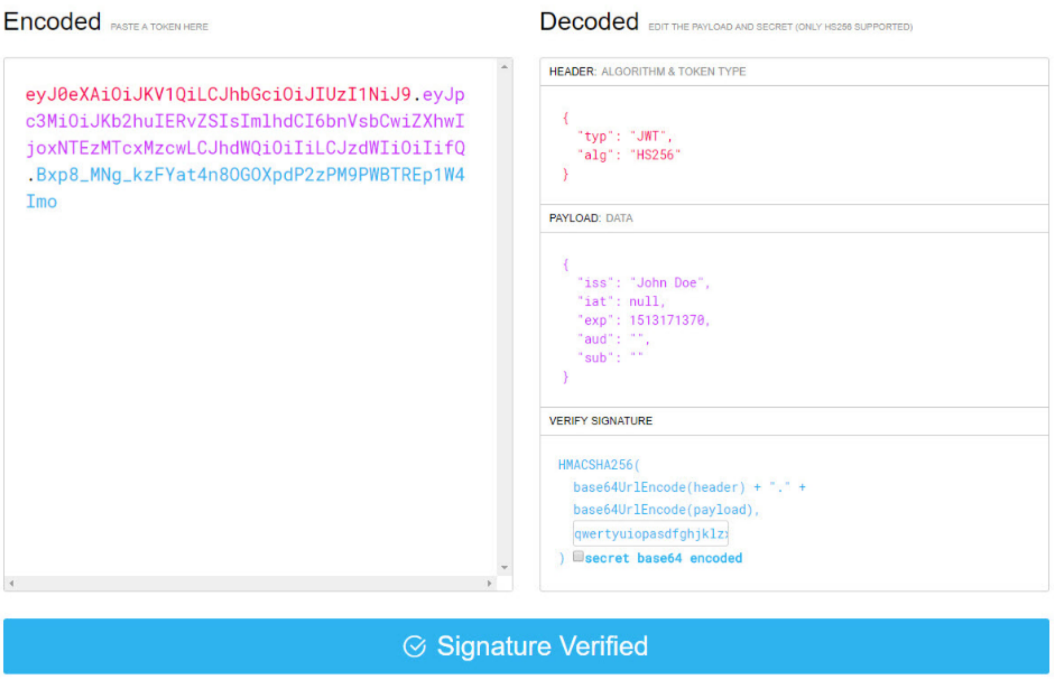
The three parts of JWT:

Header is a JSON object that consists of two keys:

- "typ" or type, is hardcoded and set as “JWT” that represents JSON Web Token;
- "alg" or algorithm, is the hashing algorithm the sender uses to sign the JWT.

Payload is a JSON object stores valid information called claims.

Signature is the encoded header and the encoded payload, encrypted according to the algorithm defined by Header and private key.


Encoded
PASTE A TOKEN HERE
Decoded
EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA
VERIFY SIGNATURE
Signature Verified

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJp
c3MiOiJKb2huIERvZSIsImlhdCI6bnVsbCwiZXhwI
joxNTEzMTcxMzcwLCJhdWQiOiIiLCJzdWIiOiIifQ
.Bxp8_MNg_kzFYat4n8OGOXpdP2zPM9PWBTREp1W4
Imo
```

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

```
{
  "iss": "John Doe",
  "iat": null,
  "exp": 1513171370,
  "aud": "",
  "sub": ""
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  qwertyuiopasdfghjklz
) secret base64 encoded
```

Figure 2.11: Example of encoded and decoded JWT

The header and payload of JWT are Base64URL1 encoded. This means that any information in these sections can be easily decoded using a specific application.
There are two ways to sign a JWT:

- symmetric, with the help of a shared secret key;
- asymmetric, with the help of a public&private key pair.

The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

Since the signature already includes the hash of the header and the payload, if the information in any one of three parts is tampered or edited the signature along with the tampered message will never match, and the JWT becomes invalid and should not be trusted[Conc].

Benefits:

- Small size (a string). So the transmission speed is fast in all kinds of methods. It can be transmitted via HTTP header (recommended)/URL/POST parameters.
- Rigorously structured. It contains all user-related authentication messages(in the payload), such as user-accessible routing, access validity period, etc. The server does

not need to connect to the database to verify the validity of the information. The payload also supports application customization.

- Support cross-domain verification.
- The signature is used to verify that the sender of the JWT is who it says it is and that the message wasn't changed along the way.

Security:

Since the signature already includes the hash of the header and the payload, if the information in any one of three parts is tampered or edited, the signature along with the tampered message will never match, and the JWT becomes invalid and should not be trusted.

2.5.2 Implement SSO with JWT

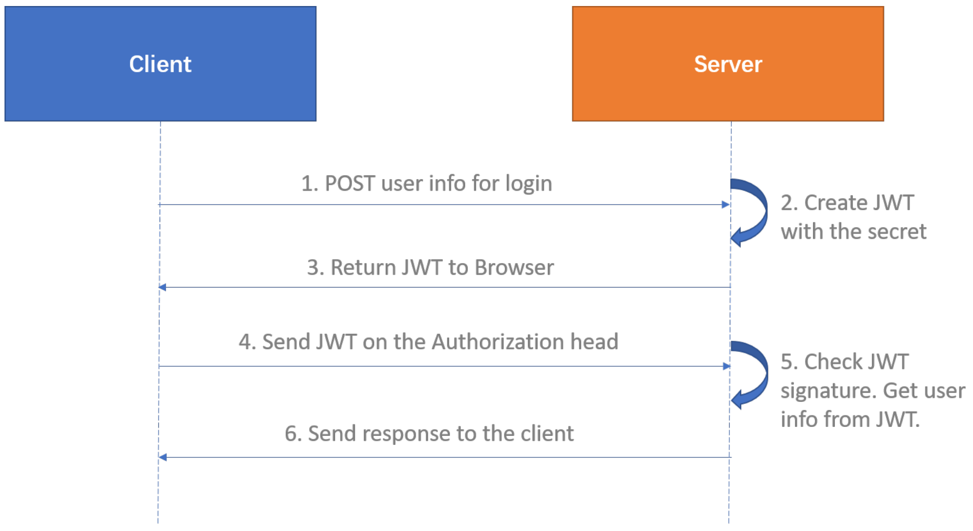

Client
Server
1. POST user info for login
2. Create JWT with the secret
3. Return JWT to Browser
4. Send JWT on the Authorization head
5. Check JWT signature. Get user info from JWT.
6. Send response to the client

Figure 2.12: SSO with JWT

1. User login
2. Authentication of the service, after passing, generate a token according to the secret
3. Return the generated token to the client
4. The user carries the token with each request
5. The server uses the public key to interpret the JWT signature, and after judging that the signature is valid, obtains user information from the payload

6. Process the request and return the response result

Since JWT already contains the user's identity information and will be carried with every request, the service does not need to save the user information or even send a query to the database.

2.5.3 Advantages on Using JWT in SSO

In addition to the advantages mentioned above, JWT has the following attractions compared to traditional server-side authentication.
1. Fully relies on stateless API, in line with RESTful principles (stateless HTTP)

State, Stateful API, Stateless API and REST
State: The status of the request is the relevant information saved during the interaction between the client and the server. The client's status is stored in page/request/session/application or global scope, and the server generally exists in the session.
Stateful API: The server saves the client's request state. Then it finds the previously interacted information in its session scope through the sessionid passed by the client and replies.
Stateless API: Statelessness is a very important principle of RESTful architecture. Each request of the stateless API is independent. It requires the client to save all the required authentication information. Each request must bring its own state and submit data including cookies and other states in the form of URL.
REST(Representational state transfer) is a software architectural style that defines a set of constraints to be used for creating Web services. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

The design of JWT fits the stateless principle: after the user logs in, the server will return tokens and the client store them locally. After this, access request to the server must bring the tokens to obtain access to related routes, services and resources. In single sign-on, JWT is used (using the HTTP header with Bearer attribute + token) to support cross-domain operations.
2. Easy in distributed management of static resources

In traditional session authentication, the server side must save the session id, which is used to verify the cookie passed from the user. But the session id will only be stored on one server, so it can only be answered by one server. Even if other servers are idle, they cannot answer and the advantages of distributed servers cannot be fully utilized. JWT relies on storing authentication information locally on the client-side, so that any server can respond. In this case, the server's resources can be better used.

3. Easy Generation and verification

There is no need to use a specific authentication scheme. As long as you have the authentication information required to generate tokens, you can call the corresponding interface to generate tokens from anywhere. Without tedious and coupled verification operations, it can be described as one-time generation and permanent use.

4. Support for native mobile apps than cookies

Cookies can be stored within a webview in mobile apps which is similar to the way they are stored in a browser. Each app has its own private space on the device . Therefore sharing cookies with each other mobile app or with the device's mobile web browser is not possible. So, native mobile applications do not support cookies and sessions well enough, but they token work fine in all this process(more details in 2.1.3 Cookies-Based).

2.6 Protocols in SSO

SSO is a general term for a class of solutions. The three main examples in single sign-on are **OpenID, OAuth, and SAML**. Before talking about the similarities and differences between the three protocols., we first give a few common features and quite important concepts about them.

Authentication VS Authorisation
Authentication: identity authentication, hereinafter referred to as authentication;
Authorisation: resource access authorization.
The role of authentication is to recognize that you can access the system, to identify whether the visitor is a legitimate user; and authorization is used to determine which resources you have access to.

Authorization Server/Identity Provider(IDP)
The service responsible for authentication is called AuthorizationServer or IdentityProvider, hereinafter referred to as IDP.

Service Provider(SP)/Resource Server
The service responsible for providing resources (API calls) is called ResourceServer or ServiceProvider, hereinafter referred to as SP.

2.6.1 SAML 2.0

Security Assertion Markup Language 2.0 (SAML 2.0) is a version of the SAML standard. It is an XML-based protocol that uses security tokens containing assertions which can be used for both authentication and authorization. The so-called security assertions are collections of statements about authentication, authorization, and user attributes (such as the user's validity or address, etc.)[Wikb].

When verifying the identity of a user, the service provider (SP) sends a SAML authentication request to IDP. The request will specify the authentication method settings in XML format. IDP will return a SAML request response after authentication of the user's identity, and also return an assertion(SAML token) in XML format to indicate the user's identity and related attributes. After the SP receives the SAML assertion, it verifies whether the source of the message is a trusted IDP, and then parses the XML to obtain authentication information after passing the verification.

Following is how SAML 2.0 works,

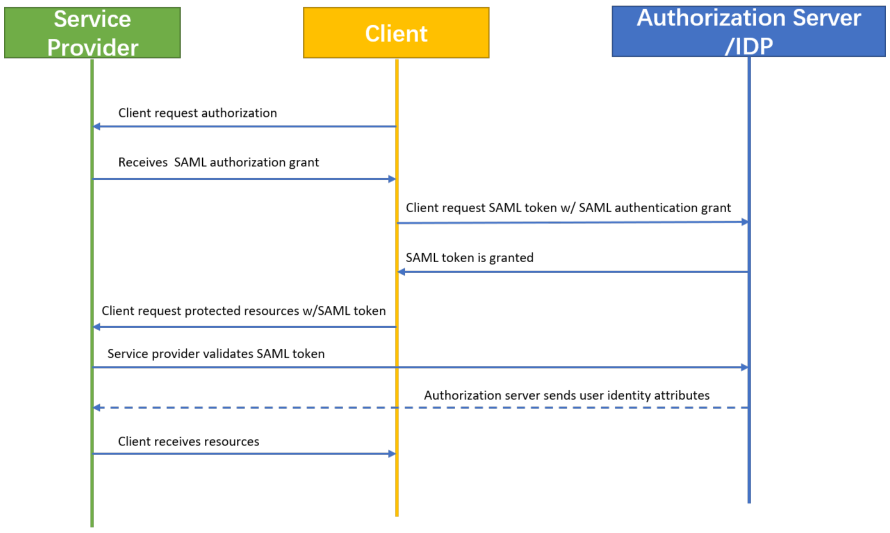

Service Provider
Client
Authorization Server /IDP
Client request authorization
Receives SAML authorization grant
Client request SAML token w/ SAML authentication grant
SAML token is granted
Client request protected resources w/SAML token
Service provider validates SAML token
Authorization server sends user identity attributes
Client receives resources

Figure 2.13: SAML 2.0

1. Users who have not yet logged in open a client to access your website (SP), which provides services but is not responsible for user authentication.

2. So the SP sends a SAML authentication request to IDP, and at the same time, the SP redirects the user's client to IDP.

3. After verifying that the request from the SP is correct, IDP presents a login form in the client to let the user fill in the username and password to log in.

4. Once the user logs in successfully, IDP will generate a SAML token containing user information (username or password). The IDP returns the token to the SP and redirects the user to the SP.

After the user logs in successfully in IDP, IDP needs to redirect the user to the SP site again. This step usually has two methods:

- HTTP redirection: This is not recommended because the length of the redirected URL is limited and cannot carry longer information, such as SAML Token.
- HTTP POST request: This is a more common practice. When the user logs in, a form is rendered, and the user clicks to submit a POST request to the SP. Or you can use JavaScript to issue a POST request to the SP.

SAML Token Structure

SAML token is also known as SAML Assertion, which is essentially an XML node contains a packet of security information:

```
<saml:Assertion ...>
..
</saml:Assertion>
```

SAML assertions are usually transferred from identity providers to service providers. Assertions contain statements that service providers use to make access-control decisions. Three types of statements are provided by SAML:

1. Authentication statements
2. Attribute statements
3. Authorization decision statements

Authentication statements assert to the service provider that the principal did indeed authenticate with the identity provider at a particular time using a particular method of authentication. Other information about the authenticated principal may be disclosed in an authentication statement[Wikb].

The SAML assertion must use a digital signature to ensure its integrity and non-repudiation. There is no mandatory requirement for SAML assertion to be encrypted as its privacy is provided at the transport layer using HTTPS in most scenarios. If the SAML assertion contains specific sensitive user information that encryption is an extra level of security needed here.

If the application is based on the Web, then the above solution work. But if you are developing a mobile app for iOS or Android, then the question arises:

1. The user opens the application on the mobile phone, and the user needs to be authenticated by IDP.
2. The application jumps to the browser. After the login authentication is completed, the token needs to be returned to the mobile application through HTTP POST.

Although the URL of POST can pull up the application, but the mobile application cannot parse the content of POST, and we cannot read the SAML Token.In any case, SAML 2.0 is not applicable to the current cross-platform scenario.

2.6.2 OAuth 2.0

OAuth 2.0 is a standard protocol for authorization, which provides a proxy access mechanism. That is to say, an application (which can be called a client) can request the resource server to obtain the resources belonging to the user or perform operations that meet the user's permissions without knowing the user's credentials like the username and password. OAuth 2.0 implements the above functions by issuing tokens to third-party applications through IDP for exchanging resources.

Let us first briefly go through the flow of OAuth2.0 under SSO.

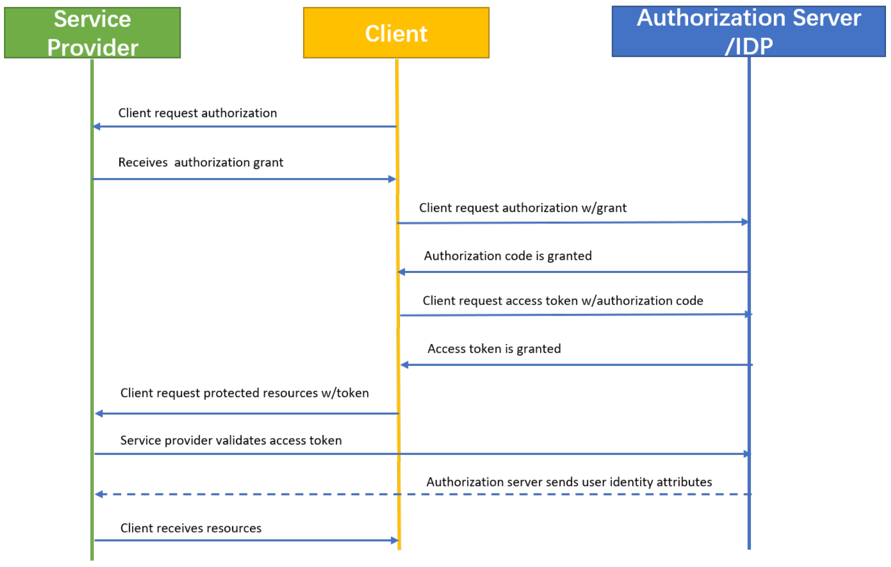

Service Provider
Client
Authorization Server /IDP
Client request authorization
Receives authorization grant
Client request authorization w/grant
Authorization code is granted
Client request access token w/authorization code
Access token is granted
Client request protected resources w/token
Service provider validates access token
Authorization server sends user identity attributes
Client receives resources

Figure 2.14: OAuth 2.0

1. The user wants to access the resources on the SP through the client (which can be a browser or a mobile application), but the SP tells the user that authentication is required and then redirects the user to IDP.
2. IDP asks the user if the SP can access the user information. If the user agrees, IDP returns an authorization code to the client.
3. The client gets the authorization code and exchanges an access token with IDP, and holds the access token to request resources from the SP.
4. After receiving the request, the SP uses the attached token to verify the identity of the user with IDP. After confirming that the identity is correct, the SP releases relevant resources to the client.

OAuth 2.0 also solves the problem that the mobile application cannot parse the content of POST or read the SAML Token under the SAML 2.0:

- On the one hand, the way that user is redirected from IDP to the client through URL redirection in Oauth 2.0. Custom schemes is allowed in URL, so the application can be pulled up even on the mobile phone;
- On the other hand, since IDP transmits the authorization code to the client instead of XML information, the code can be easily attached to the redirect URL.

2.6.3 OpenID

OpenID is an authentication standard, and many accounts on the Internet support Open ID such as Google, Yahoo, PayPal, and so on.

To use OpenID, users must first obtain an OpenID account (such as a Google account) at the OpenID (IDP). Users can use OpenID accounts to log in to any service application (relying party, RP) that accepts OpenID authentication. The OpenID protocol standard is to provide a framework for communication between IDP and RP.

The latest version of OpenID is OpenID Connect. OIDC is short for OpenID Connect.It builds an identity layer on OAuth 2.0, which is an identity authentication standard protocol based on OAuth 2.0 protocol.

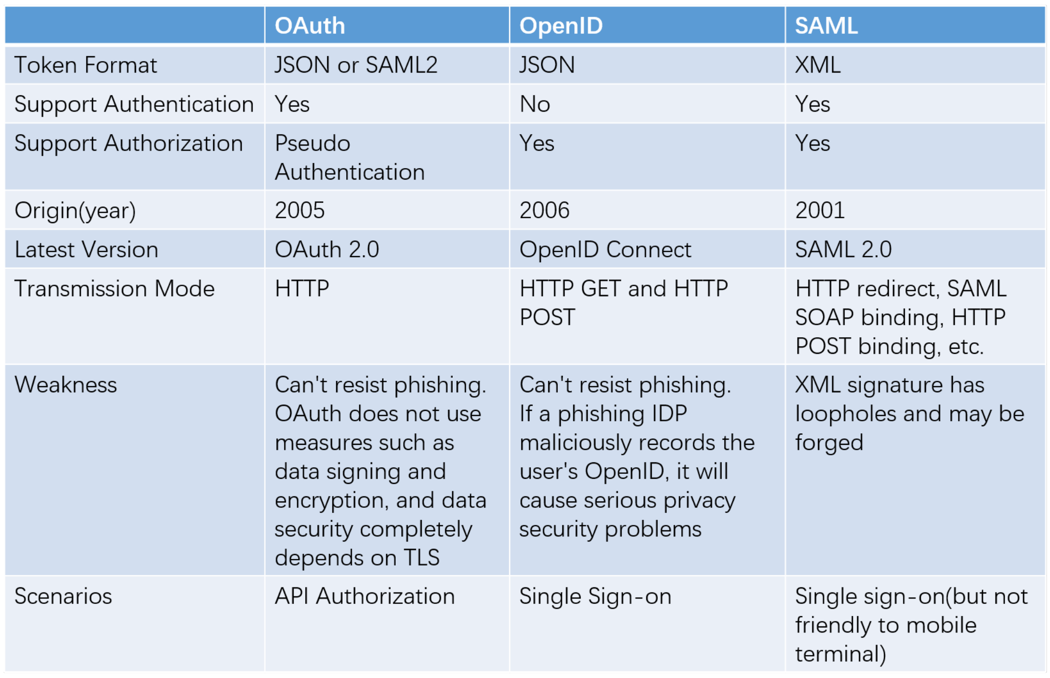
We all know that OAuth 2.0 is an authorization protocol, which cannot provide a complete identity authentication function. OIDC uses the OAuth 2.0 authorization server to provide user authentication for third-party clients, and passes the corresponding identity authentication information to the client. And it can be applied to various types of clients (such as server applications, mobile apps, JS applications), and is fully compatible with

OAuth 2.0, which means that after you build an OIDC service, it can also be used as an OAuth 2.0 service.

The Difference between OpenID Connect and Oauth 2.0

- OpenID: Only used for authentication, allowing you to log in to multiple websites with the same account. It just endorses your legal identity. When you log in to a site with your Facebook account, the site does not have access to your data on Facebook.
- OAuth 2.0: Used for authorization, allowing the authorized party to access the user data of the authorized party.

2.6.4 Comparison



	OAuth	OpenID	SAML
Token Format	JSON or SAML2	JSON	XML
Support Authentication	Yes	No	Yes
Support Authorization	Pseudo Authentication	Yes	Yes
Origin(year)	2005	2006	2001
Latest Version	OAuth 2.0	OpenID Connect	SAML 2.0
Transmission Mode	HTTP	HTTP GET and HTTP POST	HTTP redirect, SAML SOAP binding, HTTP POST binding, etc.
Weakness	Can't resist phishing. OAuth does not use measures such as data signing and encryption, and data security completely depends on TLS	Can't resist phishing. If a phishing IDP maliciously records the user's OpenID, it will cause serious privacy security problems	XML signature has loopholes and may be forged
Scenarios	API Authorization	Single Sign-on	Single sign-on(but not friendly to mobile terminal)

Figure 2.15: Comparision of Three SSO Protocols

Chapter 3

Mobile Connect

3.1 What is Mobile Connect

Mobile Connect is a new authentication standard promoted by mobile operators worldwide, which provides a convenient and safe personal solution for identity verification, authorization, and property sharing. It is a portfolio of mobile-based secure universal authentication, authorization, identity and attributes solutions.

Mobile Connect is provided by the mobile operators from global side. They work together to deliver services via standardized technical interfaces which is based on OpenID Connect and OAuth2.0.

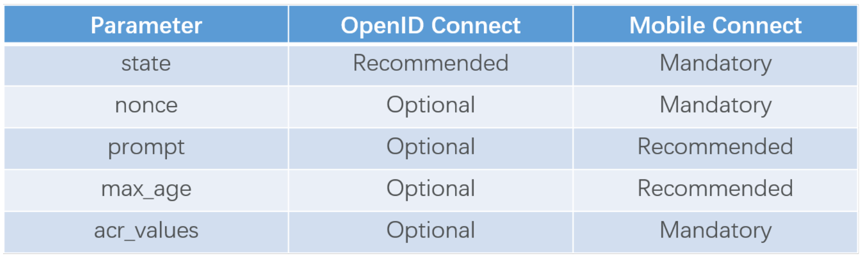
Why choose OpenID Connect?[Conf]

OpenID Connect has been adopted by Mobile Connect as the base protocol and framework because of its openness and robustness. OpenID Connect has the following advantages:

- It works on almost any device that has a web browser with access to the Internet;
- It is not specific to any operating system;
- There is a set of specifications that many developers are already familiar with. The specification is not proprietary and is currently publicly available;
- It is designed to be easy to use, reliable and secure.

The Difference between Mobile Connect and OpenID Connect

Globally speaking, the Mobile Connect API is based on the same standards and attributes defined by the OpenID Connect/OAuth 2.0 specifications. See Mobile Connect profile for more information.



Parameter	OpenID Connect	Mobile Connect
state	Recommended	Mandatory
nonce	Optional	Mandatory
prompt	Optional	Recommended
max_age	Optional	Recommended
acr_values	Optional	Mandatory

Figure 3.1: Different Requirements on Parameters

The differences are that certain parameters defined by OpenID Connect to be optional have been defined within Mobile Connect as mandatory as they are required to support the security aspects of Mobile Connect necessary to support the different authenticators as we can see on Figure 3.1[Dev].

3.2 Mobile Connect Work Flow

GSMA provides API exchanges to enable users to use the Discovery API and Authentication API for their identity authentication process with their Mobile Network Operator .

Mobile Network Operator (MNO) is a telecommunications service provider that offers services of wireless voice and data communication for its subscribed mobile users[Conb].

- The Discovery API : implemented by GSMA API Exchange platform: it enables your application to recognise the mobile network being used and whether Mobile Connect is available for that network. It also provides your application with the various URLs for the Mobile Connect service corresponding with the user's network.
- The Mobile Connect API: as known as Authentication API, implemented by MNO which allows the users to authenticate themselves using their Mobile Connect user account[Cona].

Mobile Connect provides a number of SDKs to help with integrating Mobile Connect or we can use the APIs directly.Here we explain how the application intergrate with Java-Server-Side SDK works.

Following is the overview of integrating Mobile Connect with SDK[Cone]:

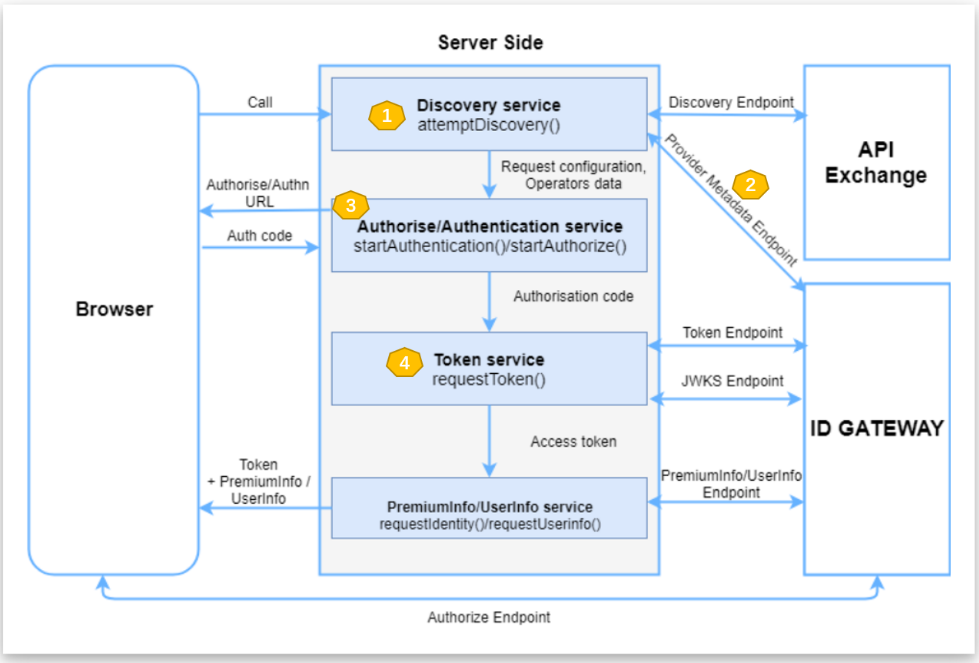

Server Side
Browser
Call
Authorise/Authn URL
Auth code
Token + PremiumInfo / UserInfo
1 Discovery service attemptDiscovery()
Request configuration, Operators data
3 Authorise/Authentication service startAuthentication()/startAuthorize()
Authorisation code
4 Token service requestToken()
Access token
PremiumInfo/UserInfo service requestIdentity()/requestUserinfo()
Discovery Endpoint
2 Provider Metadata Endpoint
Token Endpoint
JWKS Endpoint
PremiumInfo/UserInfo Endpoint
API Exchange
ID GATEWAY
Authorize Endpoint

Figure 3.2: GSMA Mobile Connect Overview

1. The client side call for the Mobile Connect Discovery endpoint with user's data as parameters.
2. A successful call to the Mobile Connect Discovery endpoint returns the end user's Mobile Network Operator (MNO) and describes the Mobile Connect services that MNO supports, via a URI to the MNO's Provider Metadata.
    - The metadata describes the Identity Gateway endpoints (Mobile Connect services) your application or service can use and how those endpoints are configured – for example, the response types an endpoint can return, the subject identifier types supported, or the Identity Services encryption algorithms in use.
3. Application or service call the Identity Gateway endpoint to make authorisation call, providing the following additional parameters:
    - client_name – specifies the name of the application/service requesting authorisation. This value is taken from Options and must match the Application Short Name.

- context – specifies the reason for the authorisation request, and should be built from the data describing the transaction requiring authorisation. The context is displayed on the authenticating (mobile) device only.
- binding_message– specifies a reference string to display on the device from which the authorisation request was invoked, and on the authenticating (mobile) device, allowing the user to visually verify that the confirmation message originated from their transaction request.

After that session is redirected and gets the authentication code. Client side sends this code to the server side, which uses this code for requestToken() method performing.

4. A successful call to the Authorisation endpoint returns an id_token identifying the user, and an access token that grants your application permission to request their personal information.

3.3 Discovery Service

Mobile Connect with Discovery

The Discovery Service allows the application/service to identify the mobile network being used by the end-user and whether it supports Mobile Connect. In addition to this, it also provides the URLs that all kinds of Mobile Connect services corresponding to the end user's network. Discovery is the process of determining the Mobile Connect Identity Provider (i.e., operator ID gateway) of the end-user. A successful discovery will return a number of endpoints and the operator-specific credentials for the following authentication with this ID Gateway.

Every mobile operator has a Mobile Country Code (MCC) and Mobile Network Code (MNC) to identify themselves. The Discovery service will identify the MCC_MNC and return the details, along with the relevant operator's Mobile Connect API details, to an application. This enables the application to call the relevant Mobile Connect service.

Also, MSISDN may be used in MCC_MNC identification. MSISDN stands for Mobile Station International Subscriber Directory Number. A number uniquely identifying a mobile phone number internationally. It serves for the mapping of the telephone number to the SIM card in a mobile phone. This number includes a Country Code, a National Destination Code, and a Subscriber Number, and doesn't include the '+'. A Country Code together with a National Destination Code identify the end-user's Mobile Network Operator. The end-user's MSISDN is always associated with the end-user's Mobile Connect account[Conb].

The Discovery service has various methods of obtaining the MCC_MNC information.

1. A user is accessing your service from a web browser where you cannot get the MCC/MNC directly.
    - To obtain the user's MCC/MNC, forward the user to the Operator Selection User Interface. This allows the user to enter their MSISDN and the correct MCC/MNC will be identified and returned to you. With this information you will be able to invoke the Discovery API to obtain the details required to perform the Mobile Connect authorisation process.
2. A user accessing your service from a Smartphone Mobile Application where your application is able to retrieve the MCC_MNC from the SIM card.
    - After obtaining the MCC/MNC from the SIM, you can invoke the Discovery API to obtain the details required to perform the Mobile Connect authorisation process.
3. A user accessing your service where you know their MSISDN number.
    - Invoke Discovery API with the MSISDN to obtain the details required to perform the Mobile Connect authorisation process[Dev].

Mobile Connect without Discovery

If the Mobile Connect endpoint of the end-user is already known and the same to the necessary credentials, Discovery API is no longer required. If the application works with specific mobile network operators or single operator, they will provide you with the credentials and endpoints that should be used. The discovery service supports many different methods to retrieve end-user ID gateway details. Methods totally depend on the applied scenario.

3.4 Authentication and Get Tokens

We here use Mobile Connect Authenticate v1.1 which specifies a set of operations that allow developers to interact with operator ID Gateways to authenticate a user.
Authentication is a two-stage process:

1. Make an OIDC Authorize call to the Mobile Connect ID Gateway asking for the end-user to authenticate themselves. If this is successfully done, then the response can be used to build the Get Token call.

2. The Get Token call uses the code value received from the authorize request to request an ID Token. The customer identifier (PCR) can then be extracted from the token to identify the end-user.

3.4.1 Method Authorize

Requests that the operator ID Gateway authenticate an end-user and return a value which can be used to make the Get Token call.

Once the user has selected the Mobile Connect button the application should open up a pop-up window or web view. The application will load this window with the authorise request. A successful authorise request will result in a **302 Found** response from the operator ID Gateway where the Location header will point to the operator wait screen. This screen displays a message to the end-user that a message has been sent to their device using the operator's selected authenticator. (You can see this in the Sandbox where an SMS wait screen is displayed).

Once the end-user has authenticated themselves another 302 Found is returned pointing to the redirect URL passed in the initial request (and previously registered with the operator ID Gateway). The response will have a number of query parameters which are used to build the follow-up Get Token call.

3.4.2 Method Get Token

Returns a signed ID Token and an Access Token when passed the code value returned from a previous authorize request. For Authentication we only need the ID Token as this contains the end-user PCR. To access this the ID Token will need to be decoded, validated and verified.

Mobile Connect supports two tokens types – Identity (ID) Tokens and Access Tokens. Both of these are granted by an operator Identity Gateway (IDGW).

Access Token

Access Token Represents stateless permission to access the protected resource the web application's after been authorized.

The access token is returned when the call Get token is made by sending the authorization code. Whenever a web application wants to access a protected resource, it should send a token. Once the access token permission is verified, the access token permission will allow the web application to access the protected resource.

Format: The Access Token can be a string or a JWT.

ID Token

The ID token contains important security information about end-user authentication.

The end-user agrees to write their own information inside the token. The application / Web service that gets the token can verify its signature and then the information contained in the ID token will be trusted. The ID token must be in the JSON web token (JWT) format.

Accordingly, the following information will be included in the load of ID token in JWT format:

- sub: asserts the identity of the user, called subject in OpenID which contains a globally unique identifier called Pseudonymous Customer Reference (PCR)[Conb].

PCR (Pseudonymous Customer Reference) is a unique identifier used by Mobile Connect to reference a pairing between a specific end-user's account and a specific application/web service. By using PCR, personal data about the user is not neccessary to access the application / Web service. Applications / Web services store PCR for user accounts in their databases. PCR is unique for each application / Web service and user account combination. Other mobile connect enabled service providers will not be able to copy and use the PCR of another application in their system to associate with the same user.

- iss: specifies the issuing authority
- aud: is generated for a particular client
- iat: issue time
- exp: expiration time
- authtime: when and how the authentication happens
- nonce: associate the application/web service session with the ID Token
- amr: the authentication method used
- athash: has an access token hash value

3.5 A Guide of Using Mobile Connect

1. Register on the Mobile Connect Developer Portal

Register an account on the mobile connect developer portal and use the sandbox to create applications that will make discovery and mobile connect calls. After registering the application, three sets of discovery credentials are created. These credentials are used when testing the application implementation through the developer portal sandbox.

2. Sandbox and Sandbox Settings

The mobile connect sandbox is used to simulate the mobile connect discovery service and other services that need to use the mobile connect ID gateway.

You need to set the mobile phone number for testing in the sandbox, so that the sandbox can complete the test supporting complete authentication.

Sandbox supports two authenticators: SMS and passthrough. SMS authentication sends a text message with a URL to the user configured MSISDN. Passthrough authentication will automatically authenticate the end-user.

3. Intergrate Mobile Connect to your application

4. Test your application in the sandbox

Run the application and click the Mobile Connect button. Enter the phone number or passthrough number set in sandbox. Click next to see a screen prompt that a message has been sent to your device. Click the link in the received message to log in to the application. This will be done automatically if the test is performed using a passthrough number.

Chapter 4

An Improved Java-Based SSO Solution

4.1 Literature Survey

Single sign-on is one of the mechanism which make security more robust against the unauthorized access authentication. The relevant papers and the analysis of the existing approaches are discussed in this section.

In [CL11], the author implemented a secure single sign-on mechanism that is efficient, secure, and suitable for mobile devices in distributed computer networks to preserve user anonymity when possible attacks occur. It uses unitary tokens based on secure hash functions, nonce values and public key encryption techniques in access control.

With the same purpose, in [Pri13], they formalize the security model of single sign-on scheme that satisfies soundness, preserves credential privacy, meets user anonymity, and supports session key exchange. It make observations about how the security of this SSO scheme can be improved by presenting their features, functionality and benefits to analyze the security level.

[MA11] describes a new anti phishing SSO model based on mobile QR code. Apart from preventing phishing attacks this new model is also safe against man in the middle and reply attacks.

Authentication is served using three major requirements:

- What you know: a password, personal identification number, or recovery questions
- What you have: a smartcard, FIDO token, one-time password (OTP), Bluetooth device, Smart Watch, or some other authenticator

- Who you are: a biometric authenticator, such as a fingerprint or face recognition
- What you do and where you're at: location-based authentication using GPS, IP address, or Integrated Windows Authentication (IWA) and how you type (keystroke biometrics)

Single-Factor Authentication (SFA) is an identity verification process that requires the access-requesting party (can be a person, software or machine) to produce to the authenticating party a single identifier – single factor – that is linked to its identity. SFA is used by default in many systems because it is easy and cheap to implement. SFA considered less secure than Multi-factor authentication(MFA), especially when the identifier is a vulnerable password[Wika].Without an additional factor to your password to confirm your identity, all a malicious user needs is your password to gain access. If password used with one of the other authentication mechanisms, then it will be two-factor authentication. In addition, to have a strong authentication mechanism, clever combinations needed to come up as a benefit for the system[O'G04]. MFA uses several different factors to verify a person's identity and grant access to various software, systems, and data.

The work in [MMSW17] explains the basis of the authentication process and common authentication technologies, such as password-based authentication, biometric-based authentication, and combinations of these technologies.Also it discussed single-factor authentication, two-factor authentication and multi-factor authentication based on three factors.

To have an efficient and secure consolidated authentication model (CAM) to authenticate a user and a variety of devices that are still using the password-based authentication or Password Authentication Scheme (PAS), [KH12] analyzes the current Consolidated Authentication Models for both user and device authentication and securely available credential (SACRED) standards).

A dynamic approach for PAS is proposed in [RW12] . It uses One-Time Passwords (OTP) instead of static passwords based on user's password, the authenticating time, as well as a unique property that the user possesses at the moment of authentication. The proposed authentication improves upon two-factor authentication and other currently known authentication schemes, and effectively protect user's account against various attack.

In [WYX12], the author demonstrate that Chang and Lee's new SSO scheme is actually insecure because it cannot satisfy the privacy of credentials and the reliability of various authentications. A single sign-on SSO mechanism is proposed, which uses a trusted authorization center (TAC) to to send the token integrated with the private key and shared public key to the user in order to control authentication.

In the paper [DNT11], authors build on proxy signature schemes to introduce the first public key cryptographic approach to single sign-on frameworks which can be easily and efficiently implemented using standard cryptography APIs and libraries. The scheme provides a framework that handles both session state across multiple services and granular access control.

Celesti et al. [CTVP10] have propose a technical solution based on the IdP/SP model along with the SAML technology. The three-phase (Discovery, Match-making and Authentication) mechanism is built for cross-cloud Single Sign-On authentication. They also extended their work in their work in [CTVP11] by developing a CLoud Enabled Virtual EnviRonment (CLEVER) which focus on the authentication phase required for a secure interaction between different CLEVER domains.

Thomas et al. present [TDC15] to discuss the the existing single sign-on systems like OpenID and OAuth for authenticating web sites. For example, these web-based single sign-on (SSO) services such as Google Sign-In and Log In with Paypal are based on the OpenID Connect protocol. These protocols enable so-called relying parties(RPs) to delegate user authentication to so-called identity providers(IDPs). They provide a framework for communication between IDP and RP)which allows user to login in with authentication.

OpenID Connect is one of the most important Single Sign-On (SSO) protocols used for delegated authentication. In[FKS17], they carry out the first in-depth security analysis of OpenID Connect. Base on it, they carried out a detailed formal model of OpenID Connect, then precisely formalize and prove central security properties for it, including authentication, authorization, and session integrity properties.

OpenID Connect is used by companies like Amazon, Google, Microsoft, and PayPal. The work in [MMSW17] analyze the famous attacks on SSO protocols and adapt these on OpenID Connect. They categorize the described attacks into two classes: Single-Phase Attacks abusing a lack of a single security check and Cross-Phase Attacks requiring a complex attack setup and manipulating multiple messages distributed across the whole protocol workflow. With an evaluation of officially referenced OpenID Connect libraries, they found 75 percent of them vulnerable to at least one Single-Phase Attack and libraries are susceptible to Cross-Phase Attacks, Then they address the existing problems in a Practical Offensive Evaluation of Single Sign-On Services (PrOfESSOS) which introduces a generic approach to improve the security of OpenID Connect implementations by systematically detecting vulnerabilities.

4.2 Proposed Solution

4.2.1 Research Objectives

From the literature survey, we found that there are two main points to be improved in the existing SSO solutions:

1. Single-factor authentication cannot satisfy the privacy of credentials and the reliability of various authentications
2. Communication security during the authentication process

Our goal is to develops a improved single sign-on solution which satisfied the problems mentioned and test its results after implementation. In the following, we will list requirements and reasons.

1. Integrate GSMA Mobile Connect in authentication

- Mobile Connect uses "things I own", that is, mobile phones as the main authentication factor, and is less vulnerable to attacks than the "I know" method used by password-based mechanisms.
- Mobile Connect requires service providers and applications to register in advance. OpenID Connect Mobile Connect Profile configuration can only be accessed after authentication using pre-registered credentials. This adds an extra layer of security.
- The Mobile Connect identity verification system uses secure private mobile network operator channels instead of the public Internet, which prevents Le attackers from sending fraudulent messages or identity verification prompts to users.
- Mobile Connect can support secure multi-factor authentication and authorization: Mobile Connect can combine the user's mobile device associated with a unique mobile number via the SIM ('something I have') and PIN ('something I know') to verify and authenticate the user. As an alternative to a PIN, the biometrics capability intrinsic to many smartphones, such as a fingerprint ('something I am'), could be used as the second authentication factor[Cond].
- Most importantly, Mobile Connect only allows users who have mobile devices with subscription IDs (phone numbers) to authenticate.

2. Use JWT to provide secure communication between SP and IP.

JWT is easy to use and has low overhead. The server does not need to record user status information and can safely transmit information between various services.

4.2.2 Implementation

To integrate mobile connect login in JWT-based single sign-on solution. I proposed the following Design:

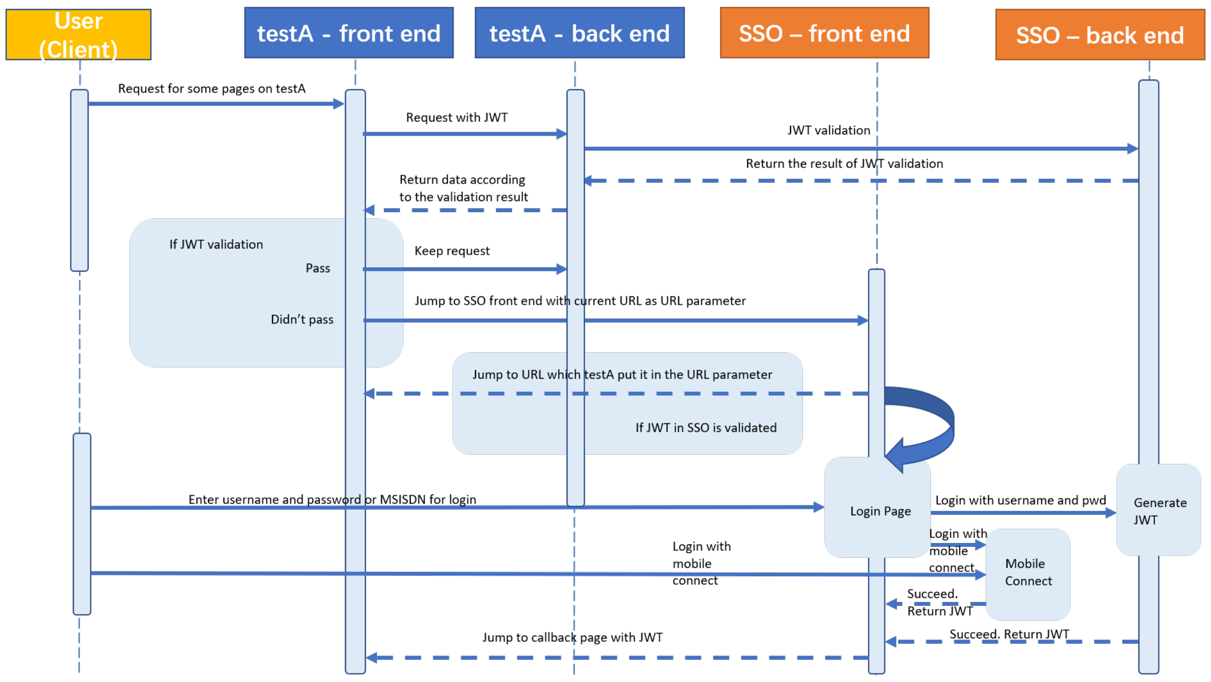

User (Client)
testA - front end
testA - back end
SSO – front end
SSO – back end
Request for some pages on testA
Request with JWT
JWT validation
Return the result of JWT validation
Return data according to the validation result
If JWT validation
Pass
Didn't pass
Keep request
Jump to SSO front end with current URL as URL parameter
Jump to URL which testA put it in the URL parameter
If JWT in SSO is validated
Enter username and password or MSISDN for login
Login Page
Login with username and pwd
Generate JWT
Login with mobile connect
Login with mobile connect
Mobile Connect
Succeed. Return JWT
Jump to callback page with JWT
Succeed. Return JWT

Figure 4.1: Work Flow of the SSO solution

First Time Request

- Request for protected information on testA will be redirected to login page on SSO server.
- Login with username and password or click 'Mobile Connect' button to authenticate with Mobile Connect.
- After login succeed, SSO will generate a JWT with HS256 algorithm or it will receive the ID token returned from Mobile Connect Authenticate service. Token will be saved in both global session and local session.
- Redirect to the initial request page on testA. TestA read the JWT from the local session and jump to SSO server for verification.

- Verification is determined by the token type. If it comes from username and password login, use HS256 to verify the JWT signature. If it is from Mobile Connect Authenticate, use RS256 to verify the signature.
- If JWT is validated, returned the protected resources.

Second Time Request

- Server receives the request for protected resource.
- Read the JWT stored in local session and jump to SSO server for verification.
- Verification the JWT signature according to its type.
- If JWT is validated, returned the protected resources.

Username and Password Login

- Retrieve the database according to the user name entered by the user to confirm whether the password is correct.

Details in Mobile Connect Login

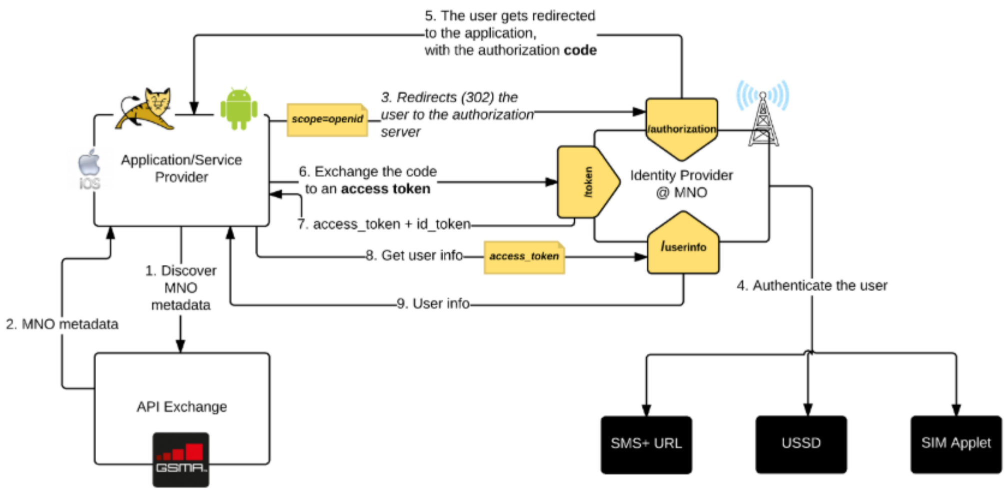

5. The user gets redirected to the application, with the authorization code
3. Redirects (302) the user to the authorization server
scope=openid
/authorization
Application/Service Provider
iOS
6. Exchange the code to an access token
/token
Identity Provider @ MNO
7. access_token + id_token
8. Get user info
access_token
/userinfo
1. Discover MNO metadata
9. User info
4. Authenticate the user
2. MNO metadata
API Exchange
GSMA
SMS+ URL
USSD
SIM Applet

Figure 4.2: Mobile Connect Authentication

- The user selects with discovery or without discovery mode, and enters the corresponding parameters (such as MSISDN or MCC_MNC) as required.

- In the discovery mode, the MNO ID gateway(IDGW) is searched first. After receiving the response, call the Mobile Connect ID Gateway asking for the end-user to authenticate themselves.
- In the without discovery mode, authentication starts directly.
- When the the end-user's authentication devices receives the authentication message from IDGW, clicking the link in the message will authenticate themselves.
- If the IDGW returns that the requested authentication status of success, an authentication code is returned. Send authentication code with context parameters to get the access_Token and ID token.

4.3 Analysis

Here we analyze our solution in the aspects of three participants in the our solution as well the deployment and security:

1. End users who are trying to access our application and have a mobile device with a subscription ID (phone number)

- For end users, the convenience offered by Mobile Connect is unmatched. People will forget the password, but it is difficult to forget their mobile phones - main part of mobile connect enable mobile devices. It spreads easily and works globally. GSMA Mobile Connect specification supports different types of authentication options, from a simple click "ok" to biometrics + PKI.
- User credentials are stored in the phone. Not in the service provider database, nor in the mobile network operator repository.
- Another big benefit is the global and federal nature. Mobile Connect users can log in to the online service at the other end of the globe with the identity that their home operator sent them. This is why E-group wants to integrate mobile connect.

2. Our application, here as a relying party or service provider

- The mobile connection makes it easy for visitors to register and use their mobile phones for authentication, which increase user conversion rates.
- Compared with widely-spread social media identities such as Google and Facebook, network services can trust the actual identity behind authentication more because the identity and possible attributes are based on subscription information from mobile network operators.

3. The Mobile Network Operator corresponds to the subscription ID of the end user

- The censored identity attributes are valuable for online services. Mobile Connect is a toolkit for mobile network operators to commercialize these assets. Attributes can help online services to better convert visitors.

4. Deployment

- The responsible party for the deployment of Mobile Connect will be the mobile network operator, as it will provide access to users. And it not only covers a single operator in a single market, but almost the most important operators in many countries.

5. Security

Here we will analyze the security feature of our solution against the attacks[MM17] by claims checking in JWT:

- Replay Attacks

  Replay attacks allow to reuse an ID token in order to authenticate the attacker as a victim. As a prerequisite, the attacker needs to get in possession of an old token and submit it to the Client. In our solution, the parameters in the body part of JWT prevent this kind of attack:

  iat (issued at) indicates the time on which the ID token is created

  exp (expires) indicates the latest time on which the ID token is valid.

  A Client implementation must if the current time is after iat but before exp.

- Signature Manipulation

  Signature Manipulation (SM) is an attack which targets the ID Token verification part of a Client. If the signature verification by a Client is not handled correctly, an attacker may be able to login as an arbitrary End-User of this application: To perform a SM attack an attacker has to act as an End-User only.

  There are different possibilities to achieve this goal:

  1. No Signature Validation at all. If the Client does not to validate the signature at all, the attacker can inject arbitrary content in the ID token.

  2. It is possible to create a valid JWT token by setting the **alg** parameter in the JWT header to none.

- Token Recipient Confusion

  Each ID token is intended to be used for a specific Client. This is indicated by the **aud** (audience) parameters. If this check is missing, an attacker can reuse tokens that are intended to be used on a different Client. As a result, the attacker will get access on the targeted Client in context of the victim.

Chapter 5

Demo

Based on the improved SSO solution we proposed in the last chapter, we will show a simple use case of the proof-of-concept implementation.

Request for protected user information on TestA

TestA: https://localhost:8081/userinfo

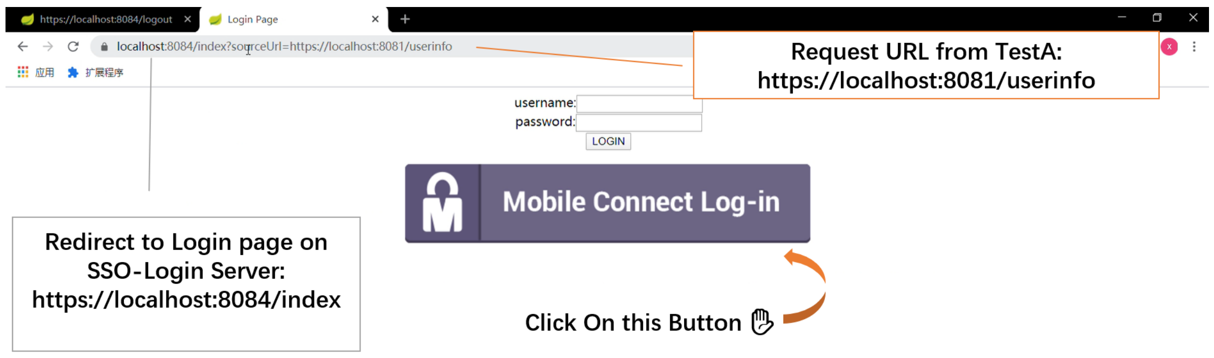
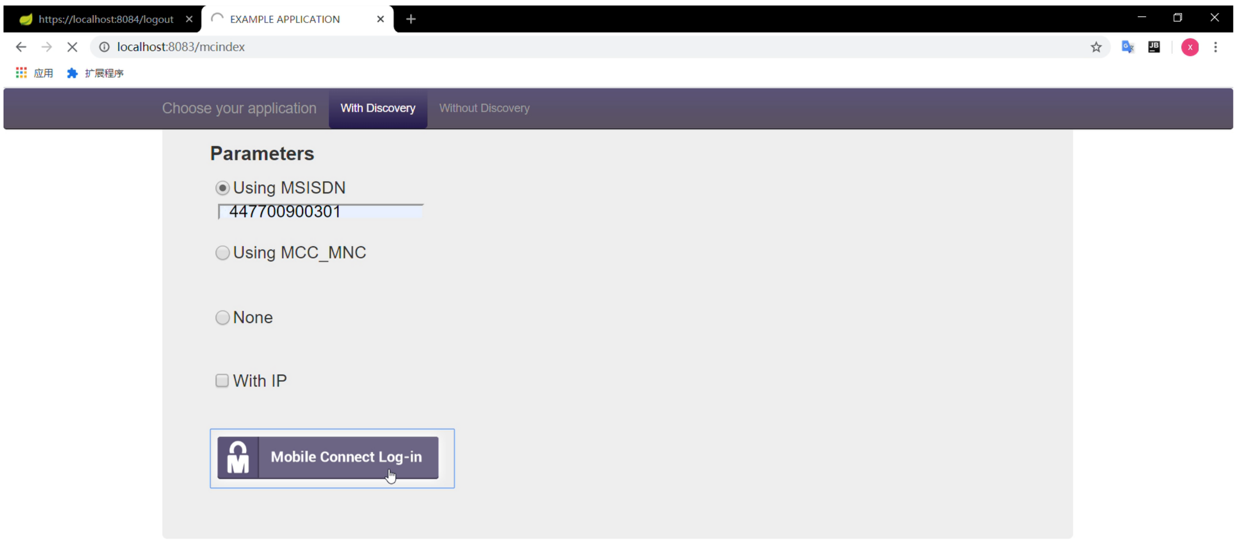

https://localhost:8084/logout
Login Page
localhost:8084/index?sourceUrl=https://localhost:8081/userinfo
应用
扩展程序
Request URL from TestA: https://localhost:8081/userinfo
username:
password:
LOGIN
Mobile Connect Log-in
Redirect to Login page on SSO-Login Server: https://localhost:8084/index
Click On this Button

Figure 5.1: Request Protected Information

Since this is the first time request, there are no tokens inside the session. Webpage is redirected to the login page on SSO-Login Server(https://localhost:8084/index).

Click on "Mobile Connect Log-in" button to proceed with Mobile Connect.



Parameters

(●) Using MSISDN
447700900301

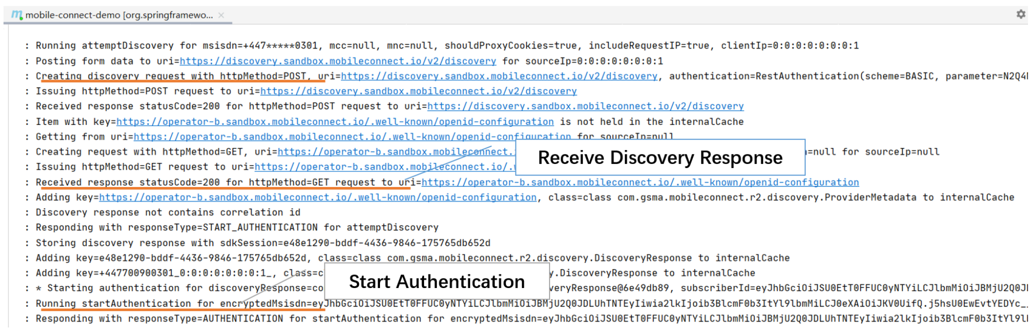
( ) Using MCC_MNC

( ) None

[ ] With IP

Figure 5.2: Login with Mobile Connect

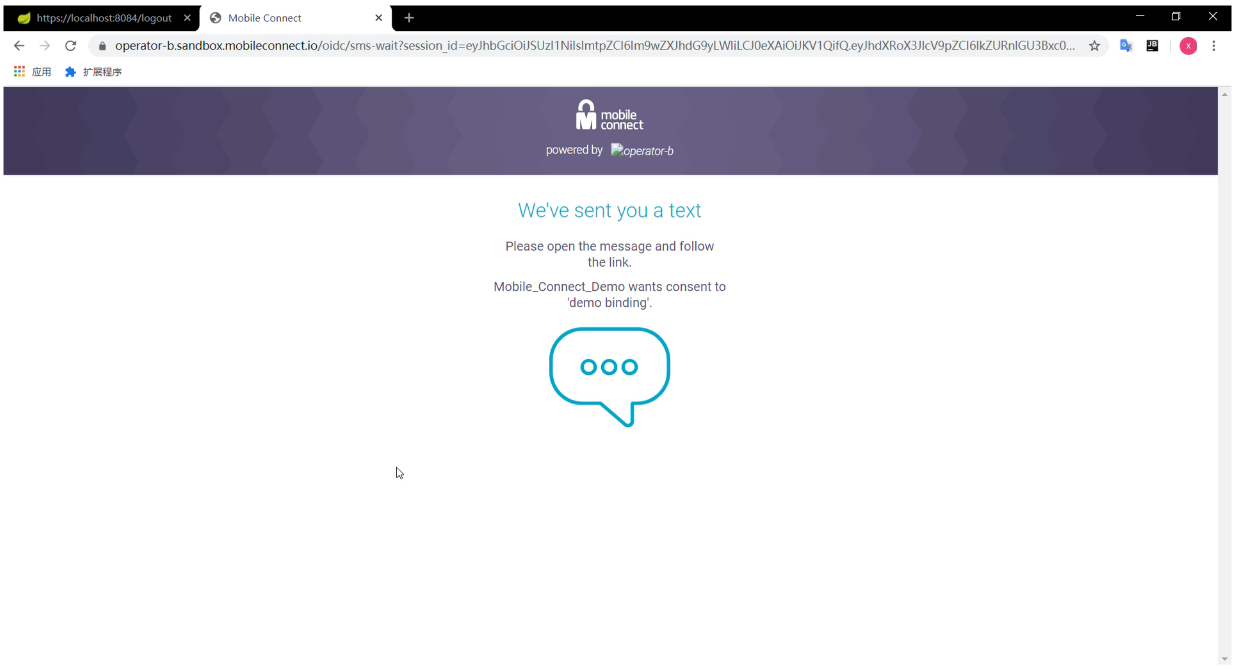
Enter the passthrough number in Mobile Connect Sandbox which is managed by operator-b: 447700900301. Start Mobile Connect with Discovery.


mobile-connect-demo [org.springframewo...
Receive Discovery Response
Start Authentication

```
: Running attemptDiscovery for msisdn=+447*****0301, mcc=null, mnc=null, shouldProxyCookies=true, includeRequestIP=true, clientIp=0:0:0:0:0:0:0:1
: Posting form data to uri=https://discovery.sandbox.mobileconnect.io/v2/discovery for sourceIp=0:0:0:0:0:0:0:1
: Creating discovery request with httpMethod=POST, uri=https://discovery.sandbox.mobileconnect.io/v2/discovery, authentication=RestAuthentication(scheme=BASIC, parameter=N2Q4
: Issuing httpMethod=POST request to uri=https://discovery.sandbox.mobileconnect.io/v2/discovery
: Received response statusCode=200 for httpMethod=POST request to uri=https://discovery.sandbox.mobileconnect.io/v2/discovery
: Item with key=https://operator-b.sandbox.mobileconnect.io/.well-known/openid-configuration is not held in the internalCache
: Getting from uri=https://operator-b.sandbox.mobileconnect.io/.well-known/openid-configuration for sourceIp=null
: Creating request with httpMethod=GET, uri=https://operator-b.sandbox.mobileconnect.                              =null for sourceIp=null
: Issuing httpMethod=GET request to uri=https://operator-b.sandbox.mobileconnect.io/.
: Received response statusCode=200 for httpMethod=GET request to uri=https://operator-b.sandbox.mobileconnect.io/.well-known/openid-configuration
: Adding key=https://operator-b.sandbox.mobileconnect.io/.well-known/openid-configuration, class=class com.gsma.mobileconnect.r2.discovery.ProviderMetadata to internalCache
: Discovery response not contains correlation id
: Responding with responseType=START_AUTHENTICATION for attemptDiscovery
: Storing discovery response with sdkSession=e48e1290-bddf-4436-9846-175765db652d
: Adding key=e48e1290-bddf-4436-9846-175765db652d, class=class com.gsma.mobileconnect.r2.discovery.DiscoveryResponse to internalCache
: Adding key=+447700900301_0:0:0:0:0:0:0:1_, class=c                    y.DiscoveryResponse to internalCache
: * Starting authentication for discoveryResponse=co                    veryResponse@6e49db89, subscriberId=eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJlbmMiOiJBMjU2Q0J
: Running startAuthentication for encryptedMsisdn=eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJlbmMiOiJBMjU2Q0JDLUhTNTEyIiwia2lkIjoib3BlcmF0b3ItYl9lbmMiLCJ0eXAiOiJKV0UifQ.j5hsU0EwEvtYEDYc_
: Responding with responseType=AUTHENTICATION for startAuthentication for encryptedMsisdn=eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJlbmMiOiJBMjU2Q0JDLUhTNTEyIiwia2lkIjoib3BlcmF0b3ItYl9l
```

Figure 5.3: Start Authentication after Receiving Discovery Response

Get Discovery response with the IDGW of operator-b. Starts authentication.


https://localhost:8084/logout
Mobile Connect
operator-b.sandbox.mobileconnect.io/oidc/sms-wait?session_id=eyJhbGciOiJSUzI1NiIsImtpZCI6Im9wZXJhdG9yLWIiLCJ0eXAiOiJKV1QifQ.eyJhdXRoX3JlcV9pZCI6IkZURnlGU3Bxc0...
应用
扩展程序
mobile connect
powered by operator-b

We've sent you a text

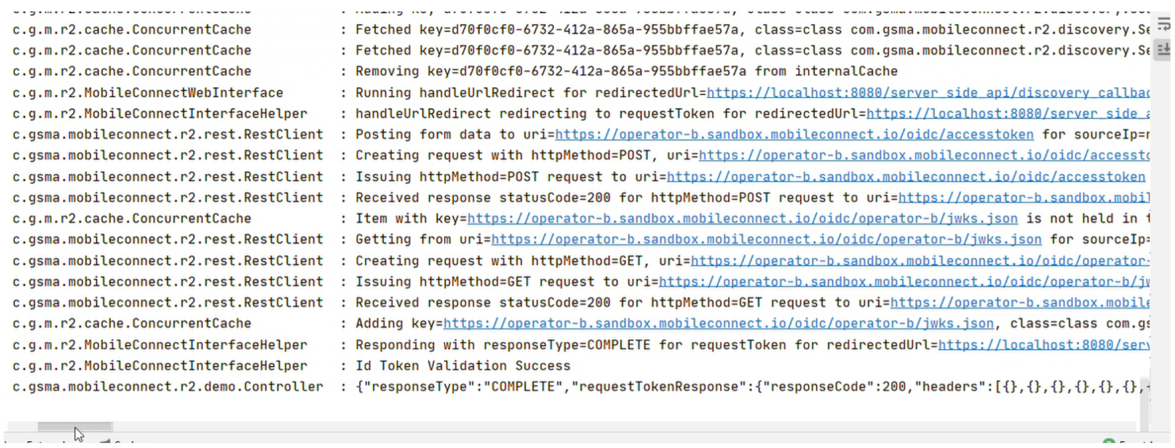
Please open the message and follow the link.

Mobile_Connect_Demo wants consent to 'demo binding'.

Figure 5.4: Authentication Message

Mobile Connect Operator sends message to devices.

Check the log on Mobile Connect side.

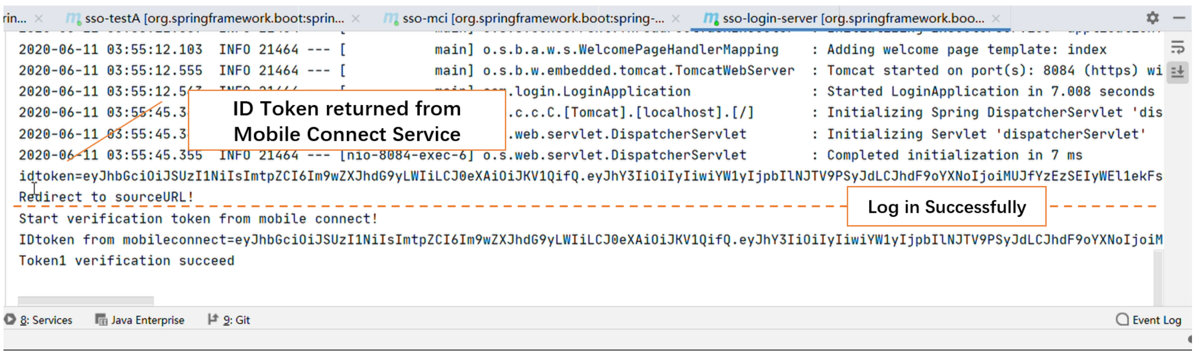


```
c.g.m.r2.cache.ConcurrentCache           : Fetched key=d70f0cf0-6732-412a-865a-955bbffae57a, class=class com.gsma.mobileconnect.r2.discovery.Se
c.g.m.r2.cache.ConcurrentCache           : Fetched key=d70f0cf0-6732-412a-865a-955bbffae57a, class=class com.gsma.mobileconnect.r2.discovery.Se
c.g.m.r2.cache.ConcurrentCache           : Removing key=d70f0cf0-6732-412a-865a-955bbffae57a from internalCache
c.g.m.r2.MobileConnectWebInterface       : Running handleUrlRedirect for redirectedUrl=https://localhost:8080/server_side_api/discovery_callbac
c.g.m.r2.MobileConnectInterfaceHelper    : handleUrlRedirect redirecting to requestToken for redirectedUrl=https://localhost:8080/server_side_a
c.gsma.mobileconnect.r2.rest.RestClient  : Posting form data to uri=https://operator-b.sandbox.mobileconnect.io/oidc/accesstoken for sourceIp=r
c.gsma.mobileconnect.r2.rest.RestClient  : Creating request with httpMethod=POST, uri=https://operator-b.sandbox.mobileconnect.io/oidc/accessto
c.gsma.mobileconnect.r2.rest.RestClient  : Issuing httpMethod=POST request to uri=https://operator-b.sandbox.mobileconnect.io/oidc/accesstoken
c.gsma.mobileconnect.r2.rest.RestClient  : Received response statusCode=200 for httpMethod=POST request to uri=https://operator-b.sandbox.mobil
c.g.m.r2.cache.ConcurrentCache           : Item with key=https://operator-b.sandbox.mobileconnect.io/oidc/operator-b/jwks.json is not held in i
c.gsma.mobileconnect.r2.rest.RestClient  : Getting from uri=https://operator-b.sandbox.mobileconnect.io/oidc/operator-b/jwks.json for sourceIp=
c.gsma.mobileconnect.r2.rest.RestClient  : Creating request with httpMethod=GET, uri=https://operator-b.sandbox.mobileconnect.io/oidc/operator-
c.gsma.mobileconnect.r2.rest.RestClient  : Issuing httpMethod=GET request to uri=https://operator-b.sandbox.mobileconnect.io/oidc/operator-b/jw
c.gsma.mobileconnect.r2.rest.RestClient  : Received response statusCode=200 for httpMethod=GET request to uri=https://operator-b.sandbox.mobile
c.g.m.r2.cache.ConcurrentCache           : Adding key=https://operator-b.sandbox.mobileconnect.io/oidc/operator-b/jwks.json, class=class com.gs
c.g.m.r2.MobileConnectInterfaceHelper    : Responding with responseType=COMPLETE for requestToken for redirectedUrl=https://localhost:8080/serv
c.g.m.r2.MobileConnectInterfaceHelper    : Id Token Validation Success
c.gsma.mobileconnect.r2.demo.Controller  : {"responseType":"COMPLETE","requestTokenResponse":{"responseCode":200,"headers":[{},{},{},{},{},{},
```

Figure 5.5: Request Tokens

After the end-user clicks on the URL in the sent message, authentication succeed, an authentication code is generated.Then, request for tokens with the authentication code. Mobile Connect Services validate the authentication state with authentication code, returned access token and ID token.

Check the log on SSO-Login Server.



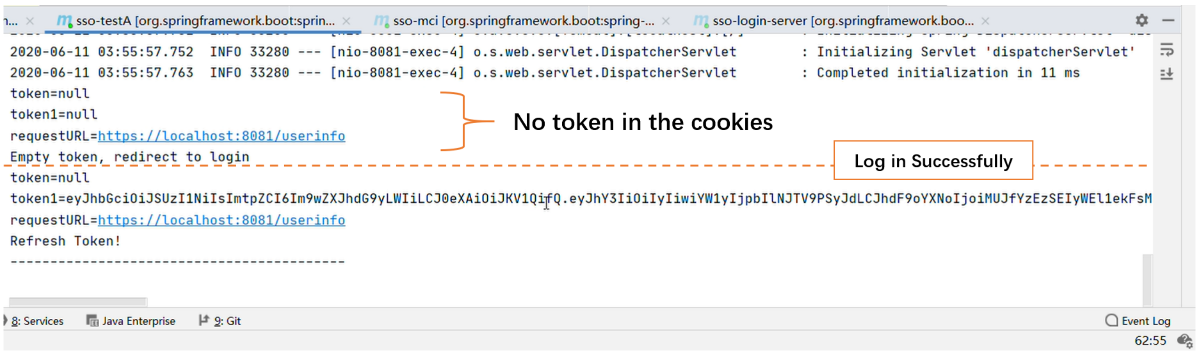
rin...
sso-testA [org.springframework.boot:sprin...
sso-mci [org.springframework.boot:spring-...
sso-login-server [org.springframework.boo...
ID Token returned from Mobile Connect Service
Log in Successfully
8: Services
Java Enterprise
9: Git
Event Log

```
2020-06-11 03:55:12.103  INFO 21464 --- [           main] o.s.b.a.w.s.WelcomePageHandlerMapping    : Adding welcome page template: index
2020-06-11 03:55:12.555  INFO 21464 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8084 (https) wi
2020-06-11 03:55:12.5                                       .login.LoginApplication                 : Started LoginApplication in 7.008 seconds
2020-06-11 03:55:45.3                                       .c.c.C.[Tomcat].[localhost].[/]         : Initializing Spring DispatcherServlet 'dis
2020-06-11 03:55:45.3                                       .web.servlet.DispatcherServlet          : Initializing Servlet 'dispatcherServlet'
2020-06-11 03:55:45.355  INFO 21464 --- [nio-8084-exec-6] o.s.web.servlet.DispatcherServlet        : Completed initialization in 7 ms
idtoken=eyJhbGciOiJSUzI1NiIsImtpZCI6Im9wZXJhdG9yLWIiLCJ0eXAiOiJKV1QifQ.eyJhY3IiOiIyIiwiYW1yIjpbIlNJTV9PSyJdLCJhdF9oYXNoIjoiMUJfYzEzSEIyWEl1ekFs
Redirect to sourceURL!
Start verification token from mobile connect!
IDtoken from mobileconnect=eyJhbGciOiJSUzI1NiIsImtpZCI6Im9wZXJhdG9yLWIiLCJ0eXAiOiJKV1QifQ.eyJhY3IiOiIyIiwiYW1yIjpbIlNJTV9PSyJdLCJhdF9oYXNoIjoiM
Token1 verification succeed
```

Figure 5.6: Request Tokens

ID token is received from Mobile Connect Service. Token is saved in both global session and local session, then redirect to the initial request page.

Check the log on TestA.

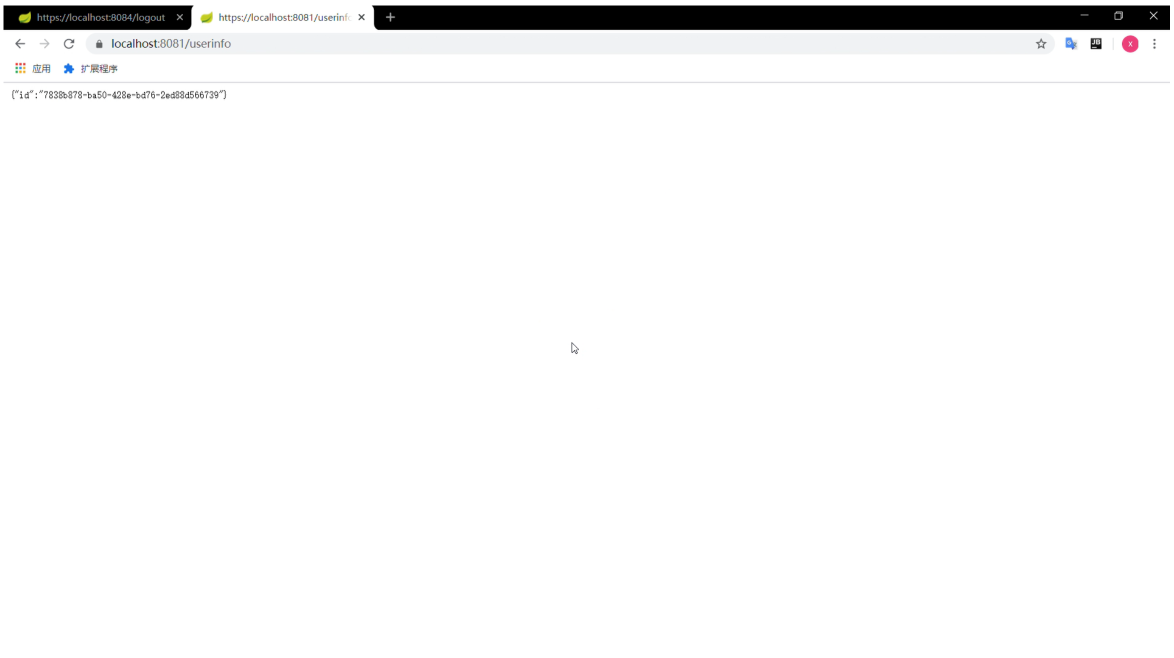


sso-testA [org.springframework.boot:sprin...
sso-mci [org.springframework.boot:spring-...
sso-login-server [org.springframework.boo...
No token in the cookies
Log in Successfully
8: Services
Java Enterprise
9: Git
Event Log
62:55

```
2020-06-11 03:55:57.752  INFO 33280 --- [nio-8081-exec-4] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2020-06-11 03:55:57.763  INFO 33280 --- [nio-8081-exec-4] o.s.web.servlet.DispatcherServlet        : Completed initialization in 11 ms
token=null
token1=null
requestURL=https://localhost:8081/userinfo
Empty token, redirect to login
token=null
token1=eyJhbGciOiJSUzI1NiIsImtpZCI6Im9wZXJhdG9yLWIiLCJ0eXAiOiJKV1QifQ.eyJhY3IiOiIyIiwiYW1yIjpbIlNJTV9PSyJdLCJhdF9oYXNoIjoiMUJfYzEzSEIyWEl1ekFsM
requestURL=https://localhost:8081/userinfo
Refresh Token!
------------------------------------------
```

Figure 5.7: Successfully Access Request Page

There were no tokens in the local session for the first time request. After login successfully by Mobile Connect, ID token is saved in both global session and local session.


https://localhost:8084/logout
https://localhost:8081/userinfo
localhost:8081/userinfo
应用
扩展程序

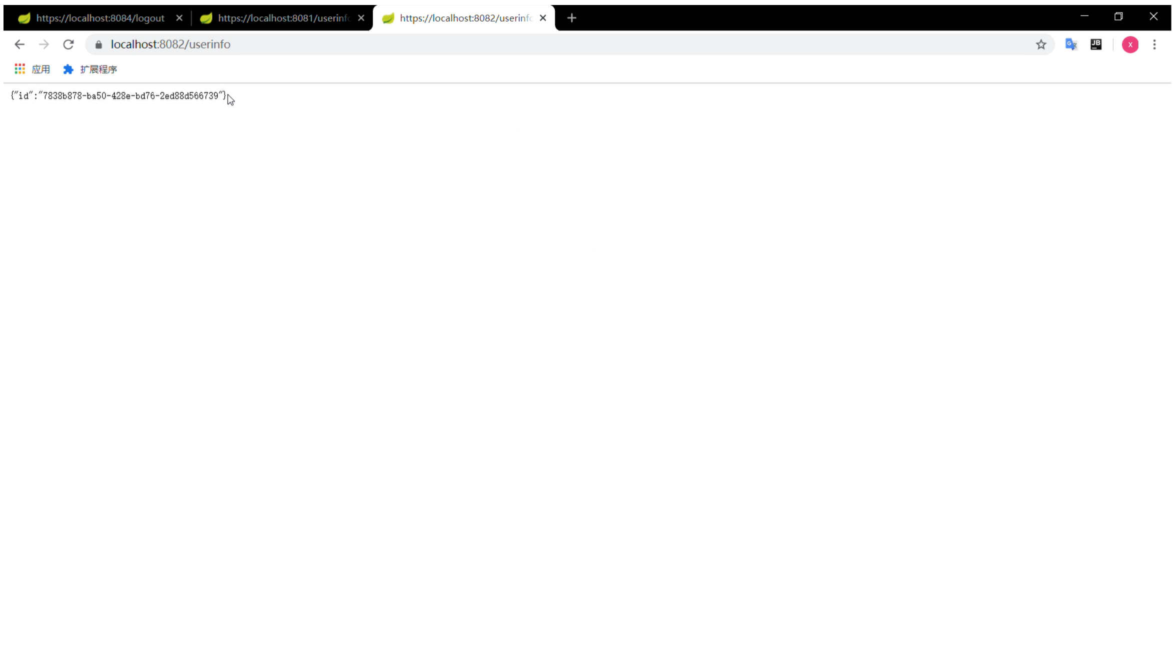
```
{"id":"7838b878-ba50-428e-bd76-2ed88d566739"}
```

Figure 5.8: Successfully Finish Initial Request

ID token is verified by SSO-Login Server, return protected user information.

Request for protected user information on TestB

TestB: https://localhost:8082/userinfo

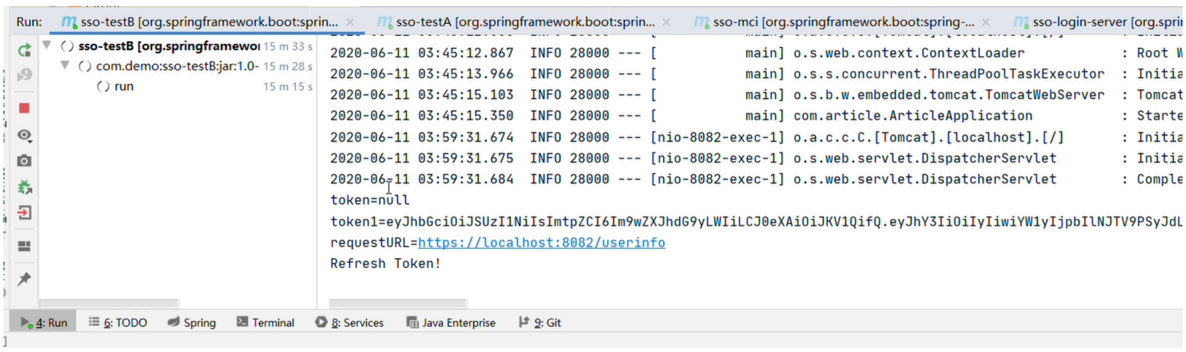

https://localhost:8084/logout
https://localhost:8081/userinfo
https://localhost:8082/userinfo
localhost:8082/userinfo
应用
扩展程序

```
{"id":"7838b878-ba50-428e-bd76-2ed88d566739"}
```

Figure 5.9: Successfully Access Request Page

Directly access to the protected user information without log in.

Check the log on TestB.


Run:
sso-testB [org.springframework.boot:sprin...
sso-testA [org.springframework.boot:sprin...
sso-mci [org.springframework.boot:spring-...
sso-login-server [org.sprin
sso-testB [org.springframewo 15 m 33 s
com.demo:sso-testB:jar:1.0- 15 m 28 s
run 15 m 15 s
4: Run
6: TODO
Spring
Terminal
8: Services
Java Enterprise
9: Git

```
2020-06-11 03:45:12.867  INFO 28000 --- [           main] o.s.web.context.ContextLoader            : Root W
2020-06-11 03:45:13.966  INFO 28000 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initia
2020-06-11 03:45:15.103  INFO 28000 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat
2020-06-11 03:45:15.350  INFO 28000 --- [           main] com.article.ArticleApplication           : Starte
2020-06-11 03:59:31.674  INFO 28000 --- [nio-8082-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initia
2020-06-11 03:59:31.675  INFO 28000 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet        : Initia
2020-06-11 03:59:31.684  INFO 28000 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet        : Comple
token=null
token1=eyJhbGciOiJSUzI1NiIsImtpZCI6Im9wZXJhdG9yLWIiLCJ0eXAiOiJKV1QifQ.eyJhY3IiOiIyIiwiYW1yIjpbIlNJTV9PSyJdL
requestURL=https://localhost:8082/userinfo
Refresh Token!
```

Figure 5.10: Successfully Access Request Page

TestB reads the token in the global session which was generated from the successfully login in previous step and redirect to SSO server for token verification.

The protected user information is returned because the token is validated.

Chapter 6

Conclusion

In this paper, we propose an improved Java-based single sign-on solution. According to the current situation on E-group and their requirements for updating existed single sign-on system, we conducted an in-depth study on GSMA Mobile Connect. Based on a summary of existing solutions, we integrated traditional login and Mobile Connect authentication into a JWT-based SSO. Communication security is guaranteed from the secure transmission of information by round-trip redirection between systems. At the same time, the Mobile Connect authentication satisfies the privacy of credentials and the reliability of identity authentication.

Chapter 7

Further Work

Our solution has only initially completed the integration of Mobile Connect authentication into single sign-on. GSMA Mobile Connect provides us a complete variety of authentication services, but we just used some of them to complete the identity verification.
In addition to authentication through end-user's mobile devices, GSMA also provides a unique sign-up service. Only the end-user is required to simply consent for their mobile operator to share core information about them with MNO. Besides, it offers an extra fraud prevention service that provides information on the pairing between a user's mobile phone account.
The further goal is to delve into GSMA Mobile Connect and thoroughly apply it to our system. We will have a more secure and user-friendly cross-border business platform that combines mobile devices, mobile network operators, and service providers with authentication. authorization, identity or attribute sharing and verification.

Bibliography

[Ana18] Ankur Anand. Building A Simple Single Sign On(SSO) Server And Solution From Scratch In Node.js. . https://codeburst.io/building-a-simple-single-sign-on-sso-server-and-solution-from-scratch-in-node 2018.

[Bla07] Jeremiah Blatz. Csrf: Attack and defense. *McAfee® Foundstone® Professional Services, White Paper*, 2007.

[CL11] Chin-Chen Chang and Chia-Yin Lee. A secure single sign-on mechanism for distributed computer networks. *IEEE Transactions on Industrial Electronics*, 59(1):629–637, 2011.

[Cona] GSMA Mobile Connect. About Mobile Connect. https://developer.mobileconnect.io/about.

[Conb] GSMA Mobile Connect. Glossary. https://developer.mobileconnect.io/glossary.

[Conc] GSMA Mobile Connect. Json Web Token . https://developer.mobileconnect.io/json-web-token-jwt.

[Cond] GSMA Mobile Connect. Mobile Connect: Mobile high-security authentication. https://mobileconnect.io/wp-content/uploads/2019/02/MC_high-security-authentication_Sep-16.pdf.

[Cone] GSMA Mobile Connect. Mobile Connect SDK Overview. https://developer.mobileconnect.io/java-sdk-overview.

[Conf] GSMA Mobile Connect. OpenID Connect. https://developer.mobileconnect.io/content/openid-connect.

[Con08a] GSMA Mobile Connect. KYC Match. https://www.gsma.com/identity/mobile-connect-deployment-map, 2008. [Online; accessed 19-July-2008].

[Con08b] GSMA Mobile Connect. KYC Match. `https://developer.mobileconnect.io/mobile-connect-di-api#tag/KYC-MATCH`, 2008. [Online; accessed 19-July-2008].

[Con08c] GSMA Mobile Connect. KYC Match. `https://www.gsma.com/identity/mobile-connect-vendors`, 2008. [Online; accessed 19-July-2008].

[CTVP10] Antonio Celesti, Francesco Tusa, Massimo Villari, and Antonio Puliafito. Three-phase cross-cloud federation model: The cloud sso authentication. In *2010 Second International Conference on Advances in Future Internet*, pages 94–101. IEEE, 2010.

[CTVP11] Antonio Celesti, Francesco Tusa, Massimo Villari, and Antonio Puliafito. Federation establishment between clever clouds through a saml sso authentication profile. *Int. J. on Adv. in Internet Tech*, 4(1), 2011.

[Dev] Orange Developer. Mobile Connect- Orange. `https://developer.orange.com/tech_guide/mobile-connect/`.

[DNT11] Bernardo Machado David, Anderson CA Nascimento, and Rafael Tonicelli. A framework for secure single sign-on. *IACR Cryptology ePrint Archive*, 2011:246, 2011.

[EU14] EU. eIDAS. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv%3AOJ.L_.2014.257.01.0073.01.ENG`, 2014.

[EU19] EU. European Union (Payment Services) . `http://www.irishstatutebook.ie/eli/2019/si/255/made/en/print`, 2019.

[FKS17] D. Fett, R. Küsters, and G. Schmitz. The web sso standard openid connect: In-depth formal security analysis and security guidelines. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 189–202, 2017.

[KH12] Jaejung Kim and Seng-phil Hong. A consolidated authentication model in cloud computing environments. *International Journal of Multimedia and Ubiquitous Engineering*, 7(3):151–160, 2012.

[Kir] KirstenS. Cross Site Request Forgery (CSRF) . `https://owasp.org/www-community/attacks/csrf`.

[MA11] Syamantak Mukhopadhyay and David Argles. An anti-phishing mechanism for single sign-on based on qr-code. In *International Conference on Information Society (i-Society 2011)*, pages 505–508. IEEE, 2011.

[MM17] Vladislav Mladenov and Christian Mainka. Openid connect. 2017.

[MMSW17] C. Mainka, V. Mladenov, J. Schwenk, and T. Wich. Sok: Single sign-on security — an evaluation of openid connect. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 251–266, 2017.

[O'G04] L. O'Gorman. Comparing passwords, tokens, and biometrics for user authentication. *Proceedings of the IEEE*, 91:2021 – 2040, 01 2004.

[Pri13] Arul Princy. A survey on single sign-on mechanism for multiple service authentications. *IJCSMC*, 2(12), 2013.

[RW12] Xuguang Ren and Xin-Wen Wu. A novel dynamic user authentication scheme. In *2012 International Symposium on Communications and Information Technologies (ISCIT)*, pages 713–717. IEEE, 2012.

[TDC15] Manoj V Thomas, Anand Dhole, and K Chandrasekaran. Single sign-on in cloud federation using cloudsim. *International Journal of Computer Network and Information Security*, 7(6):50, 2015.

[tec] Microsoft technet. ASP.NET Web Applications: How to Avoid Session Hijacking . `https://social.technet.microsoft.com/wiki/contents/articles/19364.asp-net-web-applications-how-to-avoid-session-hijacking.aspx`.

[Wika] Security Wiki. SINGLE-FACTOR AUTHENTICATION (SFA) . `https://doubleoctopus.com/security-wiki/authentication/single-factor-authentication/`.

[Wikb] Wikipedia. SAML 2.0 . `https://en.wikipedia.org/wiki/SAML_2.0`.

[WYX12] Guilin Wang, Jiangshan Yu, and Qi Xie. Security analysis of a single sign-on mechanism for distributed computer networks. *IEEE Transactions on Industrial Informatics*, 9(1):294–302, 2012.