

## Extending OpenID Connect Towards Mission Critical Applications

Deeptha R.<sup>1</sup>, Rajeswari Mukesh<sup>2</sup>

<sup>1</sup>Dept. of Information Technology, Hindustan University, Padur, Tamil Nadu, India

<sup>2</sup>Dept. of the School of Computing Sciences, Hindustan University, Padur, Tamil Nadu, India

E-mails: r.deeptha@gmail.com      rajeswarim@hindustanuniv.ac.in

**Abstract:** *Single Sign-On (SSO) decreases the complexity and eases the burden of managing many accounts with a single authentication mechanism. Mission critical application such as banking demands highly trusted identity provider to authenticate its users. The existing SSO protocol such as OpenID Connect protocol provides secure SSO but it is applicable only in the consumer-to-social-network scenarios. Owing to stringent security requirements, the SSO for banking service necessitates a highly trusted identity provider and a secured private channel for user access. The banking system depends on a dedicated central banking authority which controls the monetary policy and it must assume the role of the identity provider. This paper proposes an extension of OpenID Connect protocol that establishes a central identity provider for bank users, which facilitates the users to access different accounts using single login information. The proposed Enhanced OpenID Connect (EOIDC) modifies the authorization code flow of OpenID Connect to build a secure channel from a single trusted identity provider that supports multiple banking services. Moreover, the EOIDC tightens the security mechanism with the help of SAT to avoid impersonation attack using replay and redirect. The formal security analysis and validation demonstrate the strength of the EOIDC against possible attacks such as impersonation, eavesdropping, and a brute force login. The experimental results reveal that the proposed EOIDC system is efficient in providing secured SSO protocol for banking services.*

**Keywords:** *Online banking, SSO, authentication, identity provider, service provider, OpenID connect.*

### 1. Introduction

Single Sign-On (SSO) authentication [1] is an essential security technique that is widely adopted by modern Internet applications to simplify the login process and it avoids multiple authentication credentials. Owing to the flexibility and ease of use, SSO is preferred to access multiple systems using a single username and password. Security is the crucial aspect of mission-critical enterprise applications such as banking. Nowadays, online Internet banking applications are extensively used for

financial transactions by individuals as well as enterprise business systems. The mission-critical process of the bank requires a complex password to strengthen the security such that it is not easy to remember. Thus, instead of managing many complex passwords, SSO allows end users to log in to multiple accounts with the help of a single login credential. SSO offers several benefits that include the simple system maintenance, and easy to remember one password instead of many. Even though SSO is advantageous to the bank, the necessary disclosure of sensitive user information to the third party makes the enterprises skeptical about the implementation of SSO-based solutions. The Single Sign-On (SSO) facilitates the user to access multiple bank accounts with the help of single login credential. Since banking systems do not disclose mission critical data to any third party, most of the banks are reluctant to implement SSO. Moreover, the existing SSO systems like Security Assertion Markup Language (SAML) [2], Lightweight Directory Access Protocol (LDAP), Central Authentication Service (CAS), and OAuth2 [3] are not adaptable to the bank SSO due to their design vulnerabilities [4]. Therefore, this paper proposes an OpenID Connect system for banks that aim to deploy the secure SSO protocol without disclosing any user-sensitive data to the third party. The proposed system extends the existing OpenID Connect to enhance the authentication procedure to meet the SSO requirements of the banking world.

### 1.1. Scope and motivation

The OpenID Connect is an SSO provider designed for the consumer-to-social-network scenario and does not fulfil the strict security features enforced by mission critical enterprise business applications. The banking systems prefer secured private channel for bank transactions. The scope of the SSO implementation in banking significantly depends on the decision of regulating authority. The bank regulator emerging as an identity provider paves the way for SSO on the banking system. The SSO based banking system simplifies the user authentication and reduces the cost involved in user credential management. This work proposes Enhanced OpenID Connect (EOIDC), and the main contributions of the EOIDC are summarized as follows:

- To provide secure access to multiple bank accounts with the use of a single login credential by extending the existing OpenID Connect.
- To offer an optimal security protection of the mission critical information of the bank, the EOIDC enhances the existing OpenID Connect authorization code flow without disclosing the sensitive user information to the third party and also ensures the security of the user channel.
- To avoid the malicious transaction, the EOIDC tightens the security authorization code with a hard to guess mechanism and Security Alert Timer (SAT).

## 2. Related works

Several approaches in the past have addressed the problem of SSO in various scenarios with different techniques. SSO is widely utilized in various enterprise applications such as multi-database systems[5]. OpenID Connect is a popular and

widely deployed protocol for SSO over the Web. Owing to the significance of OpenID Connect, rigorous security analysis is paramount due to the potential applications. A formal security analysis of OpenID Connect is available in [6]. It developed a generic model to formalize the properties of authentication and authorization and provides sufficient security guidelines against new attack variants. The following section divides the conventional SSO-enabling technologies and protocols regarding the applications such as consumer-to-social-network scenario and mission critical applications.

**SSO Protocols for consumer-to-social-network.** Kerberos is one of the commonly used key distribution protocols which provides centralized SSO environment for the network, and it helps to overcome specific threats such as hacked passwords, information leakage, and alteration of information [7]. Moreover, the Kerberos infrastructure is more complex, and it is prone to several errors that create a negative impact on security. Thus, in Kerberos, users may impersonate and steal authentication tickets through simple network based attacks. Notably, the recent approaches have tried to overcome the security breaches in Kerberos [8]. LDAP is the API to access the directory service, which secures identities of the users such as employee names, telephone numbers, and credentials and network devices. Although the LDAP presented by Microsoft enables true SSO, it is applicable only in Windows environments [9]. The SAML provides a standard for exchanging the authentication and authorization data. The SAML uses the eXtended Markup Language (XML) based WS communication, and it allows a remote, unauthenticated attacker to craft specially formed messages. The arbitrary content that is crafted is known as an XML Signature Wrapping attack (XSW) [10]. To detect and overcome this attack, the detection techniques are grouped into three categories such as the policy-based approaches, the inline approaches, and the string-based approaches.

The WS policy-based approaches prevent the XML Rewriting attacks through forcing the position of the signed element in policy files. An advisor policy tool in [11] creates security policies for the WS protocols, but it is not directly applicable to SAML. WS-Security is widely accepted with its standards XML-Encryption and XML-Signature to provide security of SOAP (Simple Object Access Protocol) based web services [12]. The inline approach considers the structure of the SOAP message, and it includes the new header element called SOAP account which assists in identifying the XML rewriting attack [13]. However, the inline approach fails to address all types of XML Rewriting Attack [14]. The RewritingHealer approach extends the inline approach by considering new characteristics of the SOAP message. This extended approach improves the security in XML wrapping attack, but it is unable to completely stop the attack [15]. The string-based approach in [16] employs the effectiveness of the XPath mechanism to resist the XSW attacks in the SOAP context. However, it allows to perform the XSW attacks against the XPath referenced resources successfully [17]. OpenID Connect is a decentralized SSO web protocol that is built on top of the OAuth 2.0 protocol. It enables the client WS to rely on verification of access permission by an OIDP to authenticate an end user [18]. OpenID Connect is the child of OpenID, and it combines both OpenID and OAuth 2.0. The OpenID is a complex protocol and specified in a community standard

document [19]. OAuth (stands for Open Authorization) is a web SSO, which is a reference architecture, intended for authorization purpose, and does not facilitate user authentication. The OAuth 2.0 improves the security performance of OAuth 1.0 by including new authorization features [20]. The OAuth2 offers authorization only, whereas an OpenID Connect also offers authentication.

The characteristics of OpenID standard and its applications are discussed in [21]. The OpenID is found to be vulnerable to several attacks [22]. A novel framework improves the OpenID security using the One-Time Password (OTP) to overcome the Replay attack in SSO [23]. The benefits of OpenID Connect protocol are analyzed to apply the mechanism on the Web of Things (WoT) context for providing user authentication. The developed model is incorporated into the infrastructure that allows access to physical devices on the Web through an Enterprise Service Bus (ESB). In this context, the MultiAuth-WoT service and the provider of OpenID Connect establish communication for authentication purpose [24]. The new OpenID Connect protocol is described, and its design features such as Discovery and the Dynamic Registration extensions are analyzed in [25]. The extensions are still vulnerable to malicious attacks. A Fault-tolerant WS framework (FT-SOAP) is introduced in [26] that protect the SOAP messages using four components such as Replication Management (RM), membership management Fault Detector, Fault Notifier, and Logging/Recovery mechanism. A distributed hierarchical multi-agent architecture is developed to block malicious SOAP messages that assist in preventing the XML wrapping and Denial of Service attack [27]. The work in [28] has performed in-depth analysis of various SAML frameworks and discussed the vulnerabilities in XML Signature Wrapping (XSW). The analysis reveals that the security implications behind SAML and XML Signature are not clear yet. Moreover, the SAML framework is hard to scale on the Web. The real-world implementations of OpenID Connect is vulnerable to Cross-Site Request Forgery (CSRF) attacks and a mitigation technique is proposed in [29].

**Mission Critical Applications.** A Multi-Layered Framework is proposed in [30] that implements two-factor authentication and SSO together. The SSO protocols assure the federated identity credentials, which is utilized to access several services without re-authenticating the servers. These protocols have been widely deployed in banking Interoperability services online [31]. The paper [32] exploits the SAML for e-banking account login with single login credentials. Compared to the social network, the implementation of SSO on banking services is critical due to the stringent security requirements. The digital signature is essential in SSO protocol to ensure whether the sender has the possession of the login credentials for a bank account. In addition to the SAML, there are several SSO protocols to balance security and privacy [33]. The papers [34], [35] propose a solution to integrate and evaluate the core banking system with SSO. Moreover, the existing approaches such as Kerberos and LDAP are not suitable for the SSO applications. Though the SAML is a highly flexible and extensible framework, it is not scalable for the web. The OpenID Connect is a new RESTful model based on SAML with the desired changes according to REST Architecture. Hence, the proposed system adopts the existing OpenID Connect to deploy a secure SSO protocol for banking systems.

### 3. Proposed methodology

Password synchronization is a widely used solution to access multiple social networking accounts with one password. Despite the social network, the implementation of SSO on banking services is critical. The future Internet banking aims at providing a single identity provider that supports the transactions on multiple bank accounts. Strengthening the security of SSO is crucial to extend the usage of password synchronization from social network to mission critical applications. To solve these issues, this work proposes the EOIDC. Fig. 1 shows the block diagram of the proposed EOIDC protocol. There are three basic components used in the functions of EOIDC such as Bank Service Interface (BSI), Server System (SS), and Centralized Authentication Provider (CAP). The BSI connects the bank users with their accounts to access multiple banks online. To secure login, a bank user has to authenticate with the CAP system to access the banking service. The SS acts as a service provider that maintains the customer transaction information and enables the secure transactions with two-factor authentication such as One Time Password (OTP) and a transaction password.

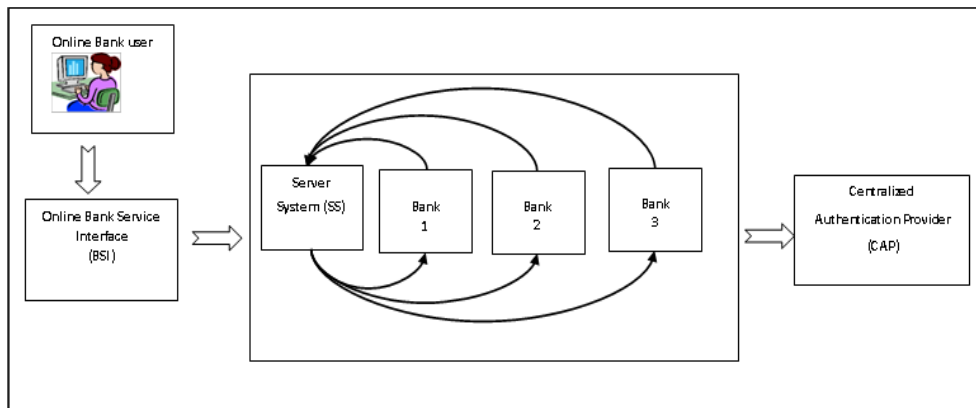


Fig. 1. Block diagram of Enhanced OpenID connect

#### 3.1. Enhanced OpenID connect operations

The proposed EOIDC is a distributed protocol to authenticate the bank users and allow them to access the online services, irrespective of the bank. To use the EOIDC services, a bank user has to be registered as an EOIDC user in SS holding any one of their bank accounts. During the registration process, the CAP assigns a common user id and secret information to the EOIDC user, and later the CAP utilizes that information to authenticate the user for accessing online banking services. The EOIDC user requests for a login and sends the request by providing the IP address via BSI. Also, the EOIDC includes an automated CAPTCHA during the login process, before receiving online services. This CAPTCHA is to ensure that the user is a human being as well as to strengthen the security of the EOIDC that exploits both

IP address and CAPTCHA as encryption keys in double encryption technique. The SS redirects the user request to the identity provider, and the user authenticates by providing the user id and secret information received during registration. Fig. 2 illustrates the communication between the user, SS, and CAP that typifies secure online services.

### 3.1.1. Connecting user to the Identity provider via SS

The SS has genuine single login information for the customers to access their multiple bank accounts online. As the EOIDC credentials are maintained by CAP, the SS has to redirect the request to CAP for authentication. Once authentication completes, the CAP redirects the request to the corresponding bank through BSI. However, the token encapsulates the bank authentication details but has no inherent protection against modification. Consequently, the EOIDC decides to send the bank authentication details in tokens which reside in the context of the authorization code and such residing token ensures secure transmission well in advance. Algorithm 1 illustrates an authorization code that covers the message using double encryption scheme. Through the encryption scheme, SS receives and validates the code integrity by generating the code with the same IP address and CAPTCHA information. After validation, the SS requests the CAP for the generated ID and Access Token through the double encryption scheme. When the user channel guarantees the security of authorization code, CAP sends the token to the user with the authorization code via BSI. The steps involved in connecting the user to the identity provider are as follows:

**Step 1.** User (U) clicks on a BSI login button adding the CAPTCHA string, and the User Agent (UA) in the SS sends the login HTTP request to a CAP.

**Step 2.** A SS connects to the CAP. The CAP confirms the receipt of EOIDC credentials from the user.

**Step 3.** The CAP requests the user for entering the credential information and accessing online services.

**Steps 4 & 5.** CAP receives the user credentials to authenticate the user.

**Steps 6 & 7.** For an authenticated user, CAP generates an authorization code AC which resides within the double encryption and redirects UA to SS via a BSI.

**Steps 8 & 9.** SS computes the authorization code, then, both the CAP and SS mutually validate the received authorization code to ensure that signed code is received from the communication channel without modification.

**Step 10.** If the CAP considers the received authorization code to be valid, then the CAP generates an ID token IDT and Access token ACT with a random and unique ID number, and sends that information to the SS via a secured direct communication.

**Step 11.** SS lists the banks in the ACT via BSI, and the user selects required bank for accessing online services.

**Steps 12 & 13.** User signs out from both the bank and EOIDC service.

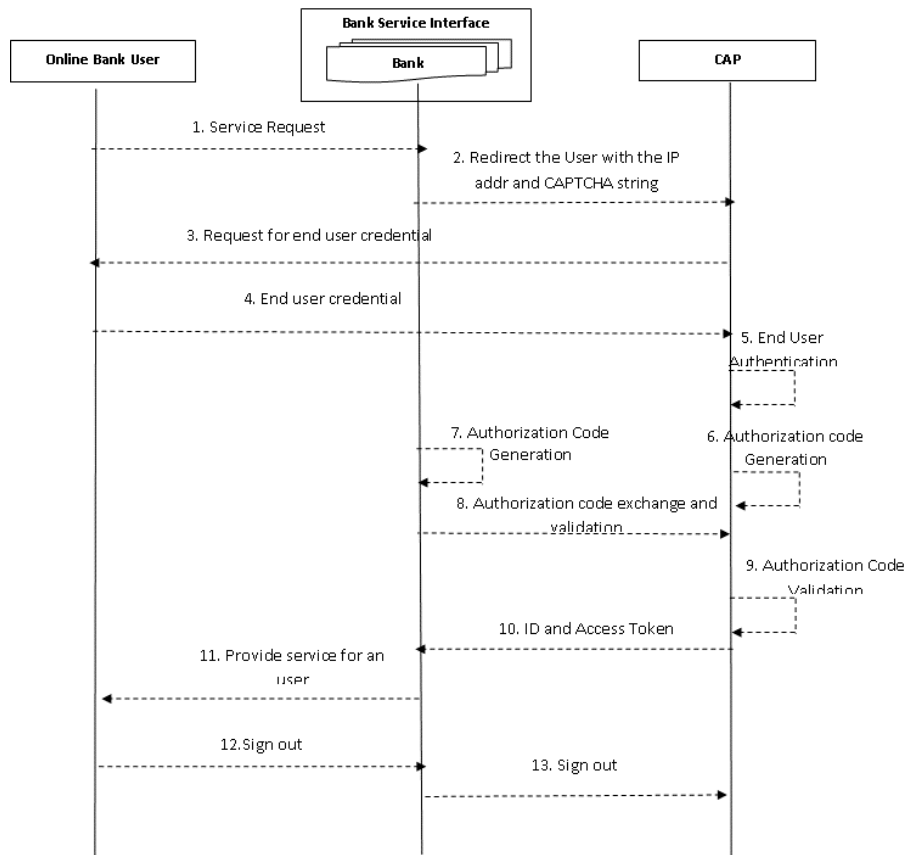


Fig. 2. Authorization code flow of EOIDC

To strengthen the process, the EOIDC adds security constraints in the authentication and authorization flows among the components. As similar to Open ID Connect, the proposed EOIDC is also implemented on the top of REpresentational State Transfer (REST) protocol. This protocol provides interoperability between CAP and SS on the Internet. Under the REST model, performing an appropriate validation and authorization is crucial to accept the user request on a communication channel. Hence, the proposed system establishes the SSL/TLS communication channel between the user and CAP which secures the user credential during authentication and authorization.

### 3.2. Secure interoperability between systems

Interoperability is a most crucial factor for both the authentication and authorization process. It avoids an adversarial user to login to the CAP and always ensures the login of the honest user. It prevents an adversary to track the information of the honest user or to interrupt the services of the legitimate user. During authentication, the SS redirects the user request to CAP, and it opens the login dialog page. After BSI loads the login page from CAP, it passes the Credential Information (CI) such as user id

and secret from UA as found under the session key. It creates a signature using the session key and avoids adversaries from learning the CI; the BSI encrypts the CI using  $CI_{key}$ , as shown in the next equation

$$(1) \quad CI = \text{encyr}\{\text{sig}(\text{User-ID \& User-Secret (Cs)})_{key}\}_{CI_{key}}.$$

Adversary users are unable to predict the correct key due to the characteristics of the SSL / TLS channel. The CAP authenticates the users and deals with the service provider, and at the same time, SS deals with the verified user for providing the online services.

Also, the proposed system includes the State Parameter SP with the authentication request [36]. The state variable must be unique such that it is hard to guess by an attacker. This variable must match with the authentication response returned by the CAP to ensure that the user is legitimate. The EOIDC employs a high-quality random-number generator to create a string of 30 characters as state parameter, which is hard for an attacker to calculate or guess. During authorization, the CAP issues the ID and Access token to the authenticated EOIDC-enabled bank users. Thus, the EOIDC designs the authorization code hard enough to guess for the attackers. Instead of generating a plain random string, the proposed system employs the IP address of the user and the captured CAPTCHA string at a BSI to create an authorization code. The CAP generates the authorization code as per the procedure explained in the algorithm 1 and sends it to the SS. The authorization code generating algorithm hashes the IP address with SHA-256 and then encodes the hash value with the  $CI_{key}$  as well as base64url to tighten the security of the code. This process ensures the secure interoperability between the servers online.

**Algorithm 1.** Authorization code generation

*/\* Authorization Code Algorithm \*/*

*Input:* CAPTCHA String, User IP address

*Output:* Encrypted IP address

*Notations:*  $a_p$ =alphabet position,  $k$ = No of letters in the CAPTCHA

*Procedure:*

**Step 1.** CAPTCHA String Conversion

A= Get the CAPTCHA string

Replace the CAPTCHA string letters with numerical value according to its alphabets position.

$CAP = ['a_p', 'a_p', \dots, 'a_p'];$

$CAP = [(\text{Sum of 'a_p'}) / k];$

**Step 2.** IP Address Conversion

IP = Obtain user IP address;

Retrieve each block of the IP address on the basis of separator '.' and store them in  $a[i]$ .

Convert  $a[i]$  integer value to individual digit and store in  $b[i]$ ,  $c[i]$ ,  $d[i]$ , and  $e[i]$ .

**Step 3.** Encrypt the IP Address

Replace the individual digits of  $b[i]$ ,  $c[i]$ ,  $d[i]$ , and  $e[i]$  with equal alphabets.

Shift the alphabets by CAP position.

$POS = b[i]+CAP; c[i]+CAP; d[i]+CAP; e[i]+CAP;$

Replace the alphabets of  $b[i]$ ,  $c[i]$ ,  $d[i]$ , and  $e[i]$  with equal numerical value.



EIP= {Encrypt – Hash( IP address)}.

**Step 4.** Added Security

Auth\_Code = BASE64URL-ENCODE(SHA256(ASCII(EIP)));

### 3.3. Secure user transaction

To initiate the secure user transaction online, the SS must receive the response for a token from the identity provider. The EOIDC redirects the user to the SS with the ID and Access Token. The SS responds to the user on verifying the ID and possibly the access token and subsequently provides the online services to the corresponding user. With the authorized access token, the CAP lists out the bank accounts to the user via BSI. It enables the users to call a particular bank web service and perform online banking transactions. The EOIDC protocol is built on the top of OAuth by adding an ID and Access Token. The ID token is a security token which includes the claim about the identity of a user by a CAP to prove the identity of a user to SS. According to OAuth, the ID token data structure is JSON Web Token (JWT). The CAP signs the token using JSON Web Signature (JWS) to provide authenticity as well as integrity to the tokens. A signed token includes a header that has the information about using a cryptographic algorithm; the body which includes the information required to authenticate the user and a signature that ensures the authenticity and integrity of the token.

**The identity of the user.** The identity of the user includes issuer and subject. The issuer field such as CAP or the EOIDC provider is mandatory, for instance, **<https://www.rbi.org.in/>**. The subject is also a mandatory field which represents the identity of the EOIDC user. The subject of an EOIDC user is unique and never resigned by the EOIDC, for instance, [alice@EOIDCProvider.com](mailto:alice@EOIDCProvider.com).

**Time-Stamps and freshness.** The fields of timestamp and expiry define the lifetime of the token. The lifetime is determined between the token creation and expiration of the token. The number used ONCE (NONCE) is a string value randomly chosen by the SS in the authentication request.

**Security Alert Timer.** The EOIDC enables the Security Alert Timer (SAT) at the SS side to prevent the redirection of the user to the malicious service. Moreover, the EOIDC enables the timer with the starting and ending time while the authorization request is transferred between the SS and CAP, and it gets enabled while the user requests for the service to the SS. Within the specified end time, if the user fails to redirect through the CAP, the EOIDC assumes that a malicious user has redirected an authenticated user. Hence, the BSI sends the alert message to the CAP, and it informs about the redirected attack to the user.

Another aspect considered in the EOIDC is the two-factor authentication during the transaction. One is the OTP, and the other one is the transaction password, given by the corresponding bank. As a result, the EOIDC facilitates the users from memorizing dozens of unique user ids and passwords and enables access to multiple bank accounts under a single login. However, the transaction password asked at the time of the transaction is separate for each bank. These entities along with the EOIDC protocol have the potential to provide secure multi-bank online transactions in a malicious environment.

### 3.4. Enhanced OpenID connect analysis and attacks evaluation

The security protocol implementations have to be rigorously verified before they are applied in the real-time for protecting the user credentials, sensitive data, and protected resources. To analyze the proposed EOIDC, this section models three attacks which exist in different parts of the EOIDC protocol. They are summarized below:

**Impersonation.** An impersonation attack is launched by sending login credentials of the legitimate user to the CAP through an attacker-controlled user-agent in BSI. The stealing of credential is possible if an attacker steals a copy of the credentials of legitimate EOIDC users or when credentials are not limited to one-time use. Lack of verification results in the response being sent to the same IP address from where the authorization request had been issued. The impersonation attack is explained as follows.

1) To impersonate a legitimate EOIDC user  $EOIDC_{U_i}$  with the identity  $ID_i$  for accessing service provider  $SS$ , an attacker  $EOIDC_{A_i}$  first sends  $U_i$  request message as  $EOIDC_{U_i}$  makes it.

2) Upon receiving message  $U_i$  from  $EOIDC_{A_i}$ , CAP verifies the signature and requests the credentials to validate the  $EOIDC_{U_i}$  (User-ID & Cs). Then, the attacker  $EOIDC_{A_i}$  ensures whether User-ID & Cs exists. If not,  $EOIDC_{A_i}$  chooses an another time to launch the attack. Otherwise, it follows the next step.

3) If the hash (IP address), as well as CAPTCHA entered by the legitimate  $EOIDC_{U_i}$ , are known by  $EOIDC_{A_i}$ ,  $EOIDC_{A_i}$  can procure the cryptography key using  $\text{enry}\{\text{sig}(\text{User-ID \& Cs})_{\text{key}}\}_{\text{CIkey}}$ .

**Eavesdropping.** By sniffing both the private key and encrypted communication between the CAP and  $SS$  via BSI, the eavesdroppers get the ID and Access Token. Using the log analyzer, the eavesdroppers are able to trace the ID and access token from its origin and overhear if the token is passed between the BSI and CAP during any subsequent communication. The SSL/TLS provides end-to-end protection. However, it is commonly used to mitigate the attacks that manipulate the traffic. The SSL imposes in such a way that the web contents are not cached, resulting in undesired side-effects such as browser warnings due to the combination of HTTPS and HTTP contents. Due to these complications, notably the websites used SSL only on login pages, but not in entire communication.

**Force-Login.** Cross-Site Request Forgery (CSRF) forces a legitimate EOIDC user to load a browser page with the malicious request and disrupts the legitimate user session. The attack also constructs the URL in a HTML construct such as  $\langle \text{img arc}=\text{bank.com/txn?To}=\text{evil} \rangle$ . It forces the browser to issue an adversary request automatically. Even though it is an adversary request, it originates from the legitimate browser, and so the adversary request cannot be differentiated from the legitimate user request. An attacker  $EOIDC_{A_i}$  forces a browser to contact the  $SS$  by faking the CAP.

$\text{Req} \rightarrow_{\text{CAP}} EOIDC_{A_i}: \text{User-ID} - \text{GET password ? TARGET}=\text{target} \dots (EOIDC_{U_i})$   
CAP: Checks the request.

### 3.4.1. Security constraints and validation in EOIDC

Three separate claims prove the authentication property of the EOIDC: (1) not learning the secrets of EOIDC User, (2) EOIDC Authentication, and (3) not observing the authorization code.

#### **Claim 1. User credential transmission against Eavesdropping and Impersonation**

Initially, the service provider SS registers with the identity provider CAP and acquires the user ID and Cs which are only known to CAP. The user credentials are transmitted in the HTTPS message via BSI while redirecting the user to the CAP. With the secured HTTPS, the request transmission is performed on the SSL/TLS communication channel. Hence, only the intended receiver CAP reads the outgoing HTTP POST requests. Other than the HTTP POST request, the messages do not include the user secrets as well as CAP data structures. Thus, the EOIDC ensures that the secret of the User (User-ID, Cs) is not disclosed or overheard by the attacker. Also, the BSI Cid and Cs avoid the type of phishing attack that is an impersonation attack of malicious service acting as a legitimate SS to gain the user credentials.

**Validation.** In the first protocol step, the User Agent UA establishes contact with BSI S

- 1)  $UA \rightarrow S \langle IST\_URL \rangle$  (HTTPS Request);
- 2)  $S \rightarrow \text{Accept the Request}; S \leftarrow U \{CAPTCHA\}$

UA connects to the inter-site transfer URL (IST\_URL) of SS and SS accepts the request and enforces the User  $U$  to enter the CAPTCHA string. The initial step has no vulnerabilities as it does not have any user credential.

#### **Claim 2. EOIDC authentication against Impersonation**

The CAP acquires the user credential from an end user via the UA and SS authenticates its end user with the CAP. However, sending a clear-text password creates the vulnerability as the same password in the clear text interaction is re-used in other interactions. The EOIDC overcomes this security issue by encrypting the user credential by applying the Algorithm 1.

**Validation.** After a successful login of  $U$ , CAP generates Authentication request Auth\_req with the required parameters.

- 3)  $S \rightarrow C \text{ Auth\_Req} (PS, Cid, Cs, Ip\_addr, CAPTCHA, RU, SP) \langle IST\_URL \rangle$

CAP generates Auth\_Req and sends it to the user via BSI. Malicious service is likely to access the Cid, Cs to act as a legitimate user. The HTTPS is a secured channel that does not allow an attacker to access these secrets. However, the C must examine the Cid and Cs before accepting the request which secures the EOIDC against the phishing attack. After a successful validation of Auth\_Req, the C presents the login form and acquires the user login credential.

- 4)  $C \rightarrow U \{ \text{Login Form} \};$
- 5)  $U \rightarrow C \{ \text{User Login Credential} \}$  using encrypted Ip\_addr and CAPTCHA

The malicious user may show interest towards acquiring a user credential for impersonating as a legitimate user. Therefore, the EOIDC forwards the user credential with the encrypted Ip\_addr and CAPTCHA.

### **Claim 3. Authorization code against force login**

The valid end user credential triggers the creation of the authorization code ( $\text{Auth\_Code} = \text{BASE64URL-ENCODE}(\text{SHA256}(\text{ASCII}(\text{EIP})))$ ) which is created for a particular user IP address and CAPTCHA. The authorization code generating algorithm encrypts the IP address using the CAPTCHA string. Moreover, to tighten the security of the code, the proposed system hashes the encrypted IP address with SHA-256, and acquires the leftmost 128 bits and encodes them with base64url. Thus, the attacker is prevented from simply reading the authorization code to impersonate the authenticated user. Moreover, even if an attacker obtains Auth\_Code, it is not possible to gain the access from CAP, as the Auth\_Code validation time is very short. Further, the Auth\_Code does not originate from any malicious party or an attacker-controlled origin in the honest browser as the attacker does not know the key terms needed for generating the code.

**Validation.** In EOIDC, CAP redirects the  $U$  to SS with the Authorization Code Auth\_Code.

6)  $\text{CAP} \rightarrow U$  to SS via UA {Auth\_Code, SP} <SS\_RU>;

On successful validation of a user credential, CAP generates an Auth\_Code and redirects the  $U$  with Auth\_Code to SS.

With the integrity maintained Auth\_Code, the  $U$  can acquire the IDT and ACT from CAP; the malicious user may attempt to access the Auth\_Code by modifying the value of SS\_RU. The system modifies the SS\_RU value to evaluate the Redirection vulnerability of EOIDC. However, the expired SAT indicates that the Auth\_Code is in the hands of malicious users before the legitimate user access the malicious service. Furthermore, to evaluate the EOIDC against the Replay attack, the Auth\_Code is accessed and replayed to gain the ID and Access Token from the CAP. However, due to the one-time validation and short time validity of the authorization code, the attacker fails to avail the service. Followed by the successful authorization code verification, the CAP generates and forwards ID and Access Tokens.

7)  $\text{CAP} \rightarrow \text{SS}\{\text{Auth\_Code}\}\langle\text{IST\_URL}\rangle$ ;

8)  $\text{SS} \rightarrow \text{CAP}\{\text{JWT}(\text{IDT}, \text{ACT})\}\langle\text{SS\_RU}\rangle$ ;

With the integrity maintained Auth\_Code, the SS obtains the JWT represented IDT and ACT from the CAP. As the secured IDT and ACT is exchanged between the CAP and SS uses the direct communication channel, an attacker is not able to access or read out the token values. With the valid IDT and ACT, the user can gain the service from the BSI. From the ACT, the BSI learns the list of banks wherein the user maintains the accounts and the bank user can directly login to the particular online services.

## **4. Experimental evaluation**

This study compares the performance of the EOIDC with the conventional OpenID Connect protocol [16] to illustrate the security enhancement.

#### 4.1. Experimental setup

EOIDC extends the OpenID Connect protocol using the MitreId Connect, which is an Open source Java implementation of OpenID Connect. The Mitre Corporation has developed it and is maintained by MIT-KIT. The Logging package in Tomcat called JULI captures all HTTP(s) traffic. The designed attack evaluation procedure is followed to examine the captured Log file to determine whether the attacks exist or not. The Log file generated by Logging package JULI is considered as the DataSet to evaluate the performance of EOIDC as the attack possibilities of EOIDC are identifiable using these Logs only.

#### 4.2. Performance metrics

The strength of the system proposed against the attacks is measured by various metrics such as Defense Strength, (Detection Accuracy, Sensitivity, and Specificity), and Delay in detection. The proposed system either detects and responds to the attacks or misses it. The authorization code has the vulnerability for both the Replay and Redirect attacks. Based on the system responses, both these attacks have four possible outcomes such as True Positive ( $A$ ), True Negative ( $D$ ), False Positive ( $B$ ) and False Negative ( $C$ ).

I) Defence strength:

$$\text{Detection Accuracy} = (A+D) / (A+B+C+D)$$

$$\text{Sensitivity} = D / (B+D)$$

$$\text{Specificity} = A / (A+C)$$

II) Delay in detection

Delay in Detection is the time taken to detect/react to the attack.

#### 4.3. Result analysis

This section evaluates the performance of EOIDC under various performance metrics, and the results are discussed in detail.

##### 4.3.1. Defence strength

A) Detection accuracy

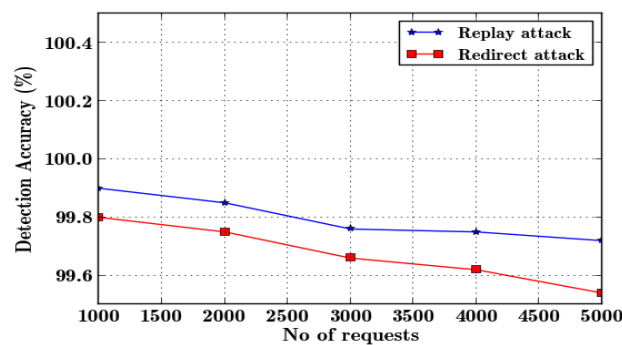
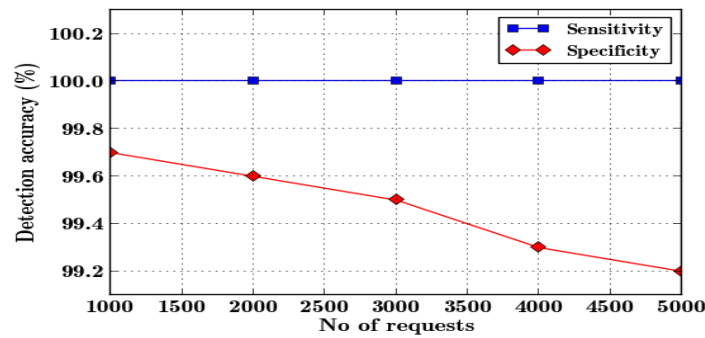


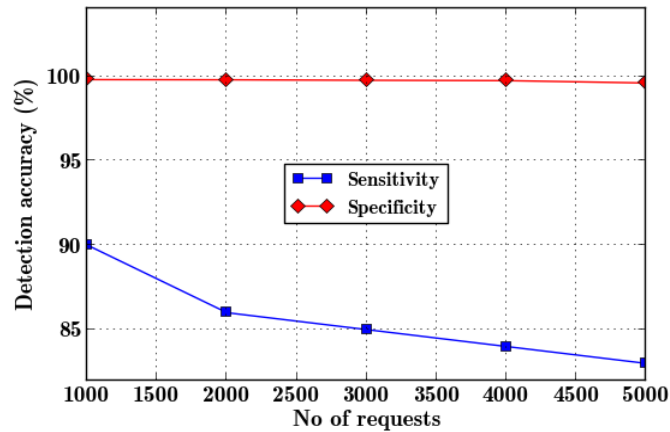
Fig. 3. No of requests vs. detection accuracy

Fig. 3 shows the performance of the EOIDC in terms of detection accuracy by increasing the number of requests from 1000 up to 5000. The accuracy detection value of both the Replay and Redirect attacks have a moderate variation while increasing the number of requests. It is observed that the accuracy detection value of Replay attack is 99.9% when the number of requests is 1000, and the detection accuracy drops to 99.72% when the number of requests is 5000. Similarly, the detection accuracy of Redirection attack falls from 99.8 up to 99.54% while increasing the request from 1000 to 5000. Though the EOIDC effectively identifies both the attacks with the security enhanced authorization code and the SAT, there is a possibility of losing the alert message due to the network congestion. Thus, there is a fall in detection accuracy.

### B) Sensitivity and specificity



(a)



(b)

Fig. 4. No of requests vs. detection accuracy

Fig. 4a and b shows the defence strength of the EOIDC in terms of sensitivity and specificity by increasing the number of requests. The sensitivity denotes the ratio of true positives, and the specificity, the ratio of true negatives. As shown in Fig. 4a,

the sensitivity value of Replay attack is 100% as in the EOIDC security mechanism the False Negative condition never occurs. However, the specificity value has moderate variation while increasing the number of requests. As shown in figure 4a, when the number of requests is 1000, the system has 99.7% of sensitivity, and it becomes 99.2% while the number of requests increases to 5000. Since the EOIDC effectively resists the replay attack due to the network congestion, the authorization code is likely to reach its expiration time, but the system assumes that the code is hacked by a malicious user, thus resulting in reduced specificity value. Similarly, Redirection attack sensitivity and specificity values also give a negative impact due to the network conjunction. As shown in figure 4b, when the no. of requests is at 1000, the system has 90% of sensitivity and 99.8% of specificity, and when the number of requests increases to 5000, the sensitivity and specificity of the system drops by 83%, and 99.6%, respectively.

#### 4.3.2. Impact of No of requests

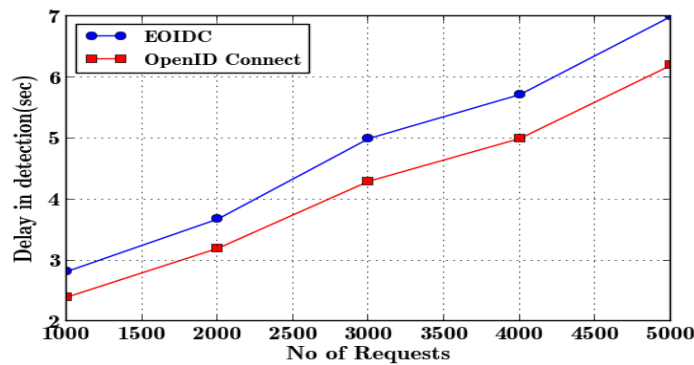


Fig. 5. No of requests vs. delay in detection

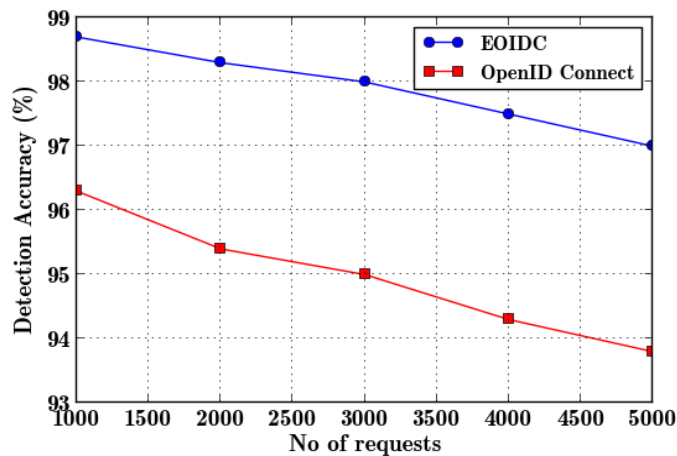


Fig. 6. No of requests vs. detection accuracy

Fig. 5 shows the performance impact of CAPTCHA in EOIDC in terms of detection accuracy and delay in detection. From Fig. 5, it is observed that the delay in detection increases linearly when escalating the number of requests. When the number of requests is at 3000, the delay in detection in the proposed EOIDC is 2.82 ms, and in the existing system, it is 2.40 ms only. It is because the proposed system has included more security on the authorization code. The detection accuracy of the EOIDC is compared with the existing OpenID Connect protocol [16]. As shown in Fig. 6, the detection accuracy of both the existing OpenID Connect and the proposed EOIDC decreases when increasing the number of requests. While the number of requests is at 3000, the EOIDC achieves 98% of accuracy, and at 5000 requests, the detection accuracy decreases to 97%. However, the proposed EOIDC achieves better detection accuracy than the OpenID Connect protocol as it restricts the possible attacks in the OpenID Connect protocol with the enhanced authorization flow.

## 5. Conclusion

This paper proposes OpenID Connect system for the mission critical enterprise applications. The proposed system enhances the existing OpenID Connect authorization code flow, which avoids the disclosing of sensitive user data to the third party and also ensures the user channel security. Moreover, the proposed EOIDC tightens the security mechanism with the support of SAT to prevent impersonation, eavesdropping, and force login attack. Finally, on formal analysis, the security validation demonstrates the protection features of EOIDC. The experimental results show that the EOIDC system successfully provides the secure SSO protocol and significantly enhances the security aspects of the existing OpenID Connect.

## References

1. Pashalidis, A., C. J. Mitchell. A Taxonomy of Single Sign-on Systems. – In: Proc. of 8th Australasian Conference on Information Security and Privacy, Vol. 27, 2003, No 27, Springer, pp. 249-264.
2. Lewis, K. D., J. E. Lewis. Web Single Sign-on Authentication Using SAML. – International Journal of Computer Science Issues, Vol. 2, 2009, pp. 41-48.
3. Li, W., C. J. Mitchell. Security Issues in OAuth 2.0 SSO Implementations. – In: Proc. of 17th International Conference on Information Security, Vol. 87, 2014, No 83, Springer, pp. 529-541.
4. Bai, G., J. Lei, G. Meng, S. S. Venkataraman et al. AUTHSCAN: Automatic Extraction of Web Authentication Protocols from Implementations. – In: Proc. of Network and Distributed System Security Symposium, 2013.
5. Zhang, L., H.-y. Ning, Y.-y. Du, Y.-x. Cui, Y. Yang. A New Identity Authentication Scheme of Single Sign on for Multi-Database. – In: Proc. of 7th IEEE International Conference on Software Engineering and Service Science, 2016.
6. Fett, D., R. Küsters, G. Schmitz. The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis and Security Guidelines. – In: Proc. of 30th IEEE Computer Security Foundations Symposium, 2017.
7. Mukhamedov, Aybek. Full Agreement in BAN Kerberos. – In: Proc. of IEEE Workshop of the 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks, 2005, pp. 218-223.



8. Abdelmajid, N. T., et al. Location-Based Kerberos Authentication Protocol. – In: Proc. of 2nd IEEE International Conference on Social Computing, 2010, pp. 1099-1104.
9. Shaw, J. Enterprise Single Sign-on: The Holy Grail of Computing. 2009.
10. McIntosh, M., P. Austel. XML Signature Element Wrapping Attacks and Countermeasures. – In: Proc. of ACM Workshop on Secure Web Services, 2005, pp. 20-27.
11. Bhargavan, K., C. Fournet, A. D. Gordon. An Advisor for Web Services Security Policies. – In: Proc. of ACM Workshop on Secure Web Services, 2005, pp. 1-9.
12. Gruschka, N., N. Luttenberger, R. Herkenhöner. Event-Based SOAP Message Validation for WS-Security Policy-Enriched Web Services. – In: Proc. of International Conference on Semantic Web and Web Services, 2006, pp. 80-86.
13. Rahaman, M. A., A. Schaad, M. Rits. Towards Secure SOAP Message Exchange in a SOA. – In: Proc. of 3rd ACM Workshop on Secure Web Services, 2006, pp. 77-84.
14. Gajek, S., L. Liao, J. Schwenk. Breaking and Fixing the Inline Approach. – In: Proc. of ACM Workshop on Secure Web Services, 2007, pp. 37-43.
15. Benameur, A., F. A. Kadir, S. Fenet. XML Rewriting Attacks: Existing Solutions and Their Limitations. – In: Proc. of International Conference on Applied Computing, 2008.
16. Gajek, S., M. Jensen, L. Liao, J. Schwenk. Analysis of Signature Wrapping Attacks and Countermeasures. – In: Proc. of IEEE International Conference on Web Services, 2009, pp. 575-582.
17. Jensen, M., L. Liao, J. Schwenk. The Curse of Namespaces in the Domain of xml Signature. – In: Proc. of ACM Workshop on Secure Web Services, 2009, pp. 29-36.
18. Sakimura, N., J. Bradley et al. OpenID Connect Core 1.0. The OpenID Foundation, S3, 2014.
19. Bellamy-McIntyre, J., C. Luteroth, G. Weber. OpenID and the Enterprise: A Model-Based Analysis of Single Sign-on Authentication. – In: Proc. of IEEE Conference on Enterprise Distributed Object Computing, 2011, pp. 129-138.
20. Hardt, Dick. The OAuth 2.0 Authorization Framework. 2012.
21. Alecu, F., P. Pocatilu, G. Stoica et al. OpenID, a Single Sign-on Solution for e-Learning Applications. – Journal of Mobile, Embedded and Distributed Systems, Vol. 3, 2011, No 3, pp. 136-141.
22. Sun, S.-T., K. Hawkey, K. Beznosov. Systematically Breaking and Fixing Openid Security: Formal Analysis, Semi-Automated Empirical Evaluation, and Practical Countermeasures. – Journal Computers and Security, Vol. 31, 2012, No 4, pp. 465-483.
23. Wang, H., C. Fan et al. A New Secure OpenID Authentication Mechanism Using One-Time Password (OTP). – In: Proc. of 7th International IEEE Conference on Wireless Communications, Networking and Mobile Computing, 2011, pp. 1-4.
24. Vinicius, C., T. G. Do. MultiAuth-WoT: A Multimodal Service for Web of Things Authentication and Identification. – In: Proc. of 21st Brazilian Symposium on Multimedia and the Web, 2015, pp. 17-24.
25. Mladenov, V., C. Mainka et al. On the Security of Modern Single Sign-on Protocols – OpenID Connect 1.0. arXiv:1508.04324v1, 2015.
26. Liang, D., et al. Fault Tolerant Web Service. – In: Proc. of 10th IEEE Asia-Pacific Conference on Software Engineering, 2003, pp. 310-319.
27. Pinzón, C. I., J. Bajo, J. F. De Paz, J. M. Corchado. S-MAS: An Adaptive Hierarchical Distributed Multi-Agent Architecture for Blocking Malicious SOAP Messages within Web Services Environments. – In: Expert Systems with Applications, Vol. 38, 2011, No 5, pp. 5486-5499.
28. Somorovsky, J., A. Mayer et al. On Breaking SAML: Be Whoever You Want to Be. – In: Proc. of 21st USENIX Security Symposium, 2012, pp. 397-412.
29. Li, W., C. J. Mitchell, T. Chen. Mitigating CSRF Attacks on OAuth 2.0 and OpenID Connect. – In: Proc. of IEEE PST, 2018.
30. Bekmezci, A. B., Ç. Eriş, P. S. Bölük. A Multi-Layered Approach to Securing Enterprise Applications by Using TLS, Two-Factor Authentication and Single Sign-on. – In: Proc. of 26th Signal Processing and Communications Applications Conference, 2018.
31. Benson, G., S. K. Chin, S. Croston, K. Jayaraman, S. Older. Banking on Interoperability: Secure, Interoperable Credential Management. – Computer Networks, Vol. 67, 2014, pp. 235-251.

32. Groß, T. Security Analysis of the SAML Single Sign-on Browser/Artifact Profile. – In: Proc. of 19th IEEE Conference on Computer Security Applications, 2003, pp. 298-307.
33. Bhargav-Spantzel, A., A. C. Squicciarini, E. Bertino. Establishing and Protecting Digital Identity in Federation Systems. – J. Comput. Security, Vol. **14**, 2006, No 3, pp. 269-300.
34. Ali, A., M. Afzali. Towards Securing e-Banking by an Integrated Service Model Utilizing Mobile Confirmation. – Research Inventy: International Journal of Engineering and Science, Vol. **4**, 2014, No 9, pp. 26-30.
35. Ardagna, C. A., E. Damiani, F. Frati, S. Reale. Adopting Open Source for Mission-Critical Applications: A Case Study on Single Sign-on. – In: Proc. of IFIP International Conference on Open Source Systems, Springer, 2006, pp. 209-220.
36. Zeller, W., E. W. Felten. Cross-Site Request Forgeries: Exploitation and Prevention. – The New York Times, 2008, pp. 1-13.

*Received 30.09.2016; Second Version 16.03.2017; Accepted 07.06.2018*