Vol.2018-DPS-174 No.13
Vol.2018-CSEC-80 No.13
2018/3/5

OpenID Connect を利用したプライバシー保護手法

持木勇輝[†1]　宇田隆哉[†1]

概要：現在，利用者がインターネット上で会員サービスを利用する際には，自分の個人情報をサーバに登録し，ID・パスワードを用いることでサービスを受けることができる．しかし，利用するサービスが多くなるにつれ，ID・パスワードの管理が煩雑になり ID の紛失やパスワードの使い回しが起こるようになる．それにより成りすまし等のセキュリティ上のリスクが発生する．それを防ぐ手段としてシングル・サインオン（Single Sign-On，以下 SSO とする）という確認方式が挙げられる．現在 OAuth が SSO の主流になっていることによって，ユーザの本人性が確かめられないことによる成りすまし問題が起きている．アプリケーション同士の連携を行うためにユーザ情報である ID・パスワードを入力し連携を行うが，OAuth では本人性を確認しないため，悪意のあるアプリケーションを使用してユーザ情報が取られた場合，その情報をもとに他のアプリケーションを使用できてしまう可能性がある．既存対策として PKCE がある．しかし，認可コードの比較検討を OP 側，サーバ内で行っており，OP がもし何らかの影響で使えない場合，認可コードの比較検討ができなくなり，ユーザがサービスを受けることができない可能性がある．そこで，チケット方式を応用し，事前に End-User，OP 間にてチケットを発行し，発行したチケットを用いて End-User，RP 間で本人確認を行う．サーバ側が止まった場合でもユーザがサービスを受けられるようにすることによって，SSO の利便性の向上をプライバシー保護の観点から図ることが本研究の目的である．

キーワード：OpenID Connect，チケット方式，セキュリティ

Privacy protection method using OpenID Connect

Yuki Mochiki[†1]　Ryuya Uda[†1]

Abstract: Users on web services are usually identified by the combination of ID and password, and their personal information is stored on servers with their IDs. Some of the users reuse the same combination of their ID and password since the number of the services has increased. In this situation, a user whose ID and password of one service are stolen may be impersonated on other services. Single Sign-On (SSO) is one of the countermeasures of the risk. However, OAuth, one of the major SSO systems, has a problem that identification of users is never verified when access permission is checked. Applications on the web refer the same access permission of a user without verifying his identity when the applications collaborate in personal identification. If one of the combinations of ID and password is stolen, all collaborating applications can be logged in by attackers. PKCE has solved the problem. However, all services are stopped when OP is halted since permission code is checked in a server of OP. Therefore, in this paper, we propose to apply a ticket method to SSO. In our proposal, users can be identified by tickets which are issued by OP before the identification. RP can verify the ticket without connecting to OP. Our method has tolerance to OP halt and personal information is protected by the separation of access control.

Keywords: OpenID Connect，Ticket Method，Security

1. はじめに

1.1 背景

現在，利用者がインターネット上で会員サービス（Web アプリケーション等）を利用する際には，自分の個人情報をサーバに登録し，ID・パスワードを用いることでサービスを利用することができる．しかし，利用するサービスが多くなるにつれ，ID・パスワードの管理が煩雑になり ID の紛失やパスワードの使い回しが起こるようになる．それにより成りすまし等のセキュリティ上のリスクが発生する[1]．それを防ぐ手段としてシングル・サインオン（Single Sign-On，以下 SSO とする）という確認方式が挙げられる．SSO とは一つの ID・パスワードを用いて複数のサービスを利用できる仕組みである．SSO では登録を行った個人情報をサーバ内に保存し，ユーザが利用したいサービスに個人情報をサーバから送りサービスをユーザに提供する．

SSO では OAuth が主に使われている．OAuth とは SSO の一種であり複数の Web アプリケーションを連携させ，情報を相互に利用できるようにする仕組みである．OAuth サーバがユーザの認証を行い，認可情報の発行を行う．それぞれのサービスが OAuth を実装し，認証されたユーザ情報を共有することによって，本人確認を行っているまた，このほかにも OpenID というシングル・サインオンも存在する[2]. OpenID Provider(ユーザ情報が保存されているサーバ，

†1 東京工科大学大学院，東京都八王子市片倉町 1404-1，Tokyo University of Technology，1404-1，Katakuramachi，Hachioji，Tokyo，Japan

Vol.2018-DPS-174 No.13
Vol.2018-CSEC-80 No.13
2018/3/5

以下 OP とする)がユーザを認証しており，ユーザを認証しているという情報の提供を行う．その情報をもとにユーザに対しサービスを提供する．OAuth が SSO として多く使われるようになった理由として，OAuth では他のアプリケーションと連携が取りやすいということが上げられる．どちらもユーザ情報である ID・パスワードを入力することにより，複数のアプリケーションにて相互の連携ができる仕組みである．この二つの違いは OpenID では使用したいアプリケーションがある場合ユーザ情報を登録したサイトからアプリケーションに対し，このユーザが正しい者であるという紹介状が発行される．それにより使用したいアプリケーションはこのユーザは正しいユーザだと判断し，ユーザに対しアプリケーションの提供を行う．OAuth では使用したいアプリケーションがある場合，登録したサイトからユーザ情報が送られてくるのは同じだが，その情報はそのサイトでユーザができること全てに対しての許可書であり，連携を行ったアプリケーションは登録サイトにてユーザが行える権限を譲渡されたことと同じである．それにより連携したアプリケーションは登録サイトの機能の使用や登録情報を参照し，ユーザに対しサービスを提供することができる．また，OpenID と違いユーザの本人性を確認しなくても連携ができてしまうことが OpenID と OAuth の大きな違いになる．

しかし，現在 OAuth が SSO の主流になっていることによって，ユーザの本人性が確かめられないことによる成りすまし問題が起きている．アプリケーション同士の連携を行うためにユーザ情報である ID・パスワードを入力し連携を行うが，OAuth では本人性を確認しないため，悪意のあるアプリケーションを使用してユーザ情報が取られた場合，その情報をもとに他のアプリケーションを使用できてしまう可能性がある．そのため，OAuth と OpenID を組み合わせた OpenID Connect（以下 OIDC とする）という新しい SSO が作られた[3]．OIDC では OpenID のようにユーザ情報を登録したサイトから証明書が発行され，ユーザの本人性が確認されると同時にアプリケーションが使用する情報または権限が譲渡される．これによって OAuth とは違い，全ての権限をアプリケーションに譲渡をしてはいないため，悪意のあるアプリケーションがユーザ情報を取得した場合，渡された情報が限定的なので，OAuth とは違いすべての情報が流出する可能性は低い．

現在，SSO は OAuth から OIDC へ移行している．OIDC は上記で説明ように，全ての権限をアプリケーションに譲渡をしてはいないため，悪意のあるアプリケーションがユーザ情報を取得した場合，渡された情報が限定的なので，OAuth とは違いすべての情報が流出する可能性は低いが，ゼロではない．その理由としては， OIDC のフローにおける認可コードがあげられる．認可コードとは ID トークンとアクセストークンを発行時に使用するための文字列のことであり，ユーザが OP へのアクセスを認可したことを示すコードである．認可コードを送信し，その結果，OP から発行されるトークンが ID トークンである．このトークンを用いてサービスはユーザを識別する．ID トークンは「ヘッダー」,「ペイロード」,「署名」からなる．ペイロードには「発行元の OP 識別子」「発行先の Relying Party(利用したいサービス，以下 RP とする)識別子」「ユーザ識別子」「発行日時」を含むデータがエンコードされている．ヘッダーに記載されたアルゴリズム(RAS256 等)を使い,「ヘッダー」,「ペイロード」に対して OP は署名を行う．OP からユーザの属性情報の取得時に使用するアクセストークンには「アクセストークンである識別子(ユーザ情報にアクセスする際に使用する識別子)」「アクセストークンの有効期限」「リフレッシュトークン(アクセストークンを再取得するためのトークン)」が含まれている．アクセストークンは ID トークン発行時に OP から発行される．

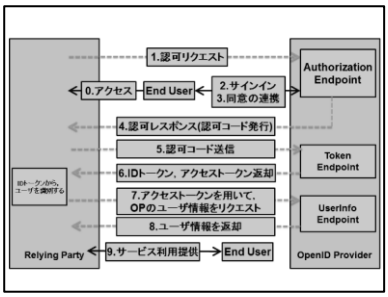

図 T　OpenID Connect のフロー概要図

Figure 1　OpenID Connect flow diagram

OIDC のフローについて，図 1 に示す．OIDC のフローでは，「0.のアクセス」の部分でユーザが Relying Party（利用したいサービス，以下 RP とする）にアクセスを行う．その際，本人確認を行う OP を選択する．その後，「1.の認可リクエスト」の部分で RP から本人確認要求がユーザに対し来る．ここで，ユーザは選択した OP にてログインを ID・パスワードで行う．図 1 では「2.サインイン」「3.同意の連携」の部分である．「3.同意の連携」では，ユーザが OP に保存されている，ユーザ情報へのアクセスを認可したことを同意することである．「4.の認可レスポンス」の部分でログインを行うと OP から認可コードが発行される．認可レスポンスは，ユーザが OP のユーザ情報へアクセスを認可したレスポンスである．認可コードは基本， URL のクエリ文にてユーザに対し発行される．認可コードが発行された後，ユーザは RP に対し認可コードを提出する．「5.の認可コード送信」で RP は提出された認可コードを用いて OP から ID トークン，アクセストークンを引き換える．図 1 の「6.ID トークン，アクセストークン返却」の部分である．引き換えた ID トークンを用いて RP はユーザの確認を行う．ID トークンには「発行元の OP 識別子」「発行先の RP 識別子」「ユーザ識別子」「発行日時」を含むデータがエンコードされて，署名付きで書きこまれている．RP にて本

Vol.2018-DPS-174 No.13
Vol.2018-CSEC-80 No.13
2018/3/5

人確認が完了したら,今度はアクセストークンを用いて OP に保存されている属性情報にアクセスを行う．アクセストークンには「アクセストークンである識別子」「アクセストークンの生存期間」「リフレッシュトークン」が含まれている．アクセストークンは， OP に保存されているユーザの属性情報へのアクセスに必要な文字列である．これらを用いて，図 1 の「7.アクセストークンの提出」を行う．その後，RP は OP から送られてきた属性情報を用いて，ユーザに対しサービスを行う．図 1 の「8.ユーザ情報を返却」「9.サービス利用提供」の部分である．

本研究では，OIDC における成りすまし問題についてプライバシー保護を踏まえ研究を行っていく．その中でも，OIDC フローの認可コード横取り攻撃に対する手法を提案する．認可コード横取り攻撃とは，OP，RP 間において発行されるコードを横取り,第三者が奪ったコードを用いて，OP にアクセスを行い, End-user 情報を盗む攻撃のことである．図 1 のフロー内では「4.の認可レスポンス」,「5.の認可コード送信」のフロー中に行われる．

OIDC の本人確認を行うサーバが止まる理由として，サーバに不特定多数のユーザからアクセスが集中すること想定している．想定図を図 2 に示す．OIDC では，本人確認を行う認証サーバにはサービスだけがアクセスするわけではなく，インターネットから不特定多数のユーザが来ることになる．また，SSL では，オフラインでユーザの本人確認を行うが，OIDC ではユーザがサービスを利用するたびにインターネットを介して，ユーザの本人確認を行う，そのため，認証サーバにアクセスが集中することになるそのため，認証サーバが高負荷となり止まる場合がある．

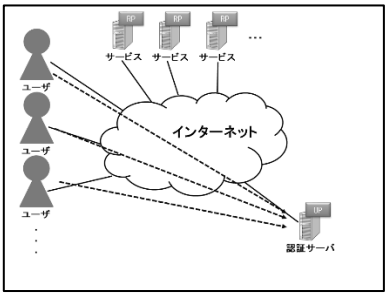

図 1　止まる場合の想定図

Figure 2　Assumption diagram when it stops

2. 既存対策

2.1 OAuth

OAuth サーバがユーザの認証を行い，認可情報の発行を行う．それぞれのサービスが OAuth を実装し，認証されたユーザ情報を共有することによって，本人確認を行っている．

2.2 PKCE

現在，認可コード横取り攻撃に対応として，OIDC には PKCE といわれる対策がなされている．PKCE で実装されていることは二つである．図 2 の「1.の認可リクエスト」後，認可リクエストが行われてから認可レスポンス（認可コード発行）までの間に，認可リクエストの値及び認可リクエストメソッド, 認可コードを OP 上のデータベースに保存する．図 2 の「3.5.DB に保存」部分である．PKCE の保存フロー概要図を図 2 に示す.

PKCE では図 3 の認可コード送信後に，認可コードからリクエストパラメーター(認可コード内の 43~128 文字列)の値を取り出し，その値とデータベースに保存しておいた認可リクエストメソッドを用いて認可リクエストを計算，その計算した値がデータベースに保存しておいた認可リクエストの値と等しいかどうかチェックする[4]．チェックを行うフローは，図 3 の「5.1.リクエストパラメーターの値取り出し」から「5.5.等しいなら返却」の部分である．PKCE のチェックフロー概要図を図 3 に示す．Token Endpoint では送信された，認可コードに対して，ID トークン，アクセストークンの発行を行う．

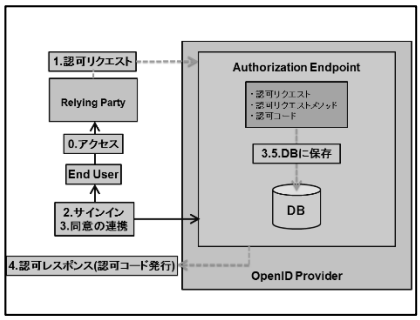

図 2　PKCE 保存フロー概要図

Figure 3　PKCE save flow outline drawing

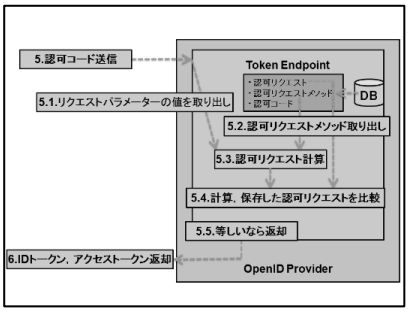

図 3　PKCE チェックフロー概要図

Figure 4　PKCE check flow outline drawing

しかし, PKCE では認可コードの比較検討を OP 側，サーバ内で行っており，OP の本人確認にアクセスが集中し，本人確認が行えない場合，認可コードの比較検討ができなくなり，ユーザがサービスを受けることができない可能性がある．

Vol.2018-DPS-174 No.13
Vol.2018-CSEC-80 No.13
2018/3/5

3. 既存研究

3.1 Self-Issued OpenID Provider

Self-Issued OpenID Provider（以下, Self-Issued OP とする）とは，OpenID の一種であり，OS（PC，モバイル端末）やブラウザ内で動作し，自分自身でクライアントの管理を行い，クライアント（Web サービスやアプリ）からリクエストを受けたら署名付けの ID トークンを返すものである．これを利用することによって，OIDC を行う端末がネットワークにつながらない場合でも，本人確認を行うのが自分自身のため ID トークン，アクセストークンを発行することができる．以下の図 4，図 5 に Self-Issued OP をモバイル端末の特定に利用するユースケースを示す.

ID トークンには「発行元の OP 識別子」「発行先の RP 識別子」「ユーザ識別子」「発行日時」，アクセストークンには「アクセストークンである識別子」「アクセストークンの生存期間」「リフレッシュトークン」が含まれている．state には Self-Issued OP を利用するユーザのセッション状態，scope にはサービスが必要としているユーザ情報が記載されている. 図 4 では 1.の部分にて, アプリケーションが Self-Issued OP に対し，本人確認要求を行っている．その後，2.の部分にて Self-Issued OP が利用者のユーザ情報を返却している

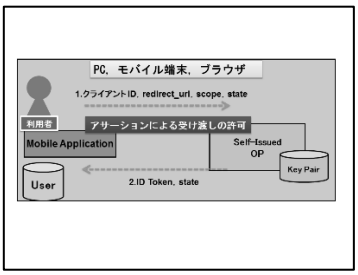

図 4　Self-Issued OP を用いた User Identifier 取得フロー
Figure 5　User Identifier acquisition flow using Self-Issued OP

図 5 では web アプリケーションが 1.で本人確認要求を行っている．その後，図 4 と同じく，Self-Issued OP に対し，本人確認要求が 2.にて行われている. 要求後は 3.4 を経て，Web アプリケーションに利用者のユーザ情報が送信されている.

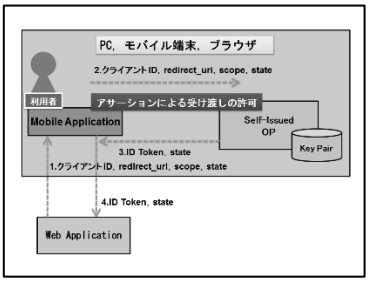

図 5　Self-Issued OP を用いた User Identifier 取得フロー
Figure 6 User Identifier acquisition flow using Self-Issued OP

4. 提案手法

4.1 既存対策での問題点

既存の技術である, PKCE ではサーバ側が止まった場合，ユーザがサービスを受けられない可能性がある[4]．認可コードの比較検討を行うフローが機能しないため，比較検討が行えず，ユーザの本人性が確かめられないからである．そのため，本研究では，認可コード以外のものを使い，ユーザの本人性を確かめることによって，PKCE が使えない場合でもユーザがサービスを受けることができるようにするためのものである.

また，本人確認を行うサーバが止まる理由として，サーバにアクセスが集中する(何らかの予約の受付時間や悪意のある他者からの攻撃など)場合, サーバに繋がるのを待っている人が多数存在する場合(順番待ちの渋滞状態)がある．本提案では，サーバにアクセスが集中する場合を想定している.

そこで，チケット方式を応用し，事前に End-User，OP 間にてチケットを発行する．発行したチケットを用いて End-User，RP 間で本人確認を行い，サーバ側が止まった場合でもユーザがサービスを受けられるようにすることによって，SSO の利便性の向上をプライバシー保護の観点から図ることが本研究の目的である.

4.2 チケット方式の提案

本提案手法は OIDC のフローに, チケット方式を応用し，End-User，OP 間にてチケットを発行する．そのチケットを用いて, End-User, RP 間で本人確認を行う. チケット発行，本人確認の手順は以下に示す.

OP：S，End-User の属性情報を保存した OP：k_a, k_b
End-User の権限：A，RP を利用するユーザの権限 ：k_a
RP：B，ユーザが利用するサービス　　：k_b

1.End-User → OP　：ログイン処理，乱数生成：n1, B
2.OP →End-User　：チケット発行：$E_{k_a}(\mathrm{n1, B, k}, AT_b)$
$AT_b = E_{k_b}(A, k, D)$ は権限のみ，本人情報はなし

1.End-User → RP　：$AT_b, E_k(\mathrm{n2})$
2.RP → End-User　：$E_k(\mathrm{n2}-1, \mathrm{n3})$
3.End-User → RP　：$E_k(n3-1)$
n1，n2，n3 は乱数

まず, ユーザは k_a , RP は k_b の共通鍵を持っている．そして，OP にはユーザと RP の共通鍵 k_a, k_b を共有している.

4.2.1 チケット発行

End-User は OP にアクセスを行う．OIDC ではログイン処理を行った場合，アクセストークンがユーザに発行される．このとき，OP に対して，乱数及び本人確認を行う RP の送信を行う．その後，OP 側から End-User に対し，n1 及

び選択した RP を用いたチケットが発行される．チケットの中身は $E_{k_a}(\mathrm{n1},\mathrm{B},\mathrm{k},AT_b)$ で表せる．n1 は生成した乱数，B は選択した RP，k は生成したセッション鍵，ATb は $E_{k_b}(A,k,D)$ である．D は有効期限，ATb は権限のみであり，本人情報は含まれていない．このとき，ユーザは発行したチケット ATb¬¬¬の中身を見ることはできない．チケット ATb は RP，B の共通鍵 kb にて暗号化されているためである．そのため，End-User，OP 間の通信が第三者に盗聴されていた場合でも，第三者はチケット ATb を見ることができない．チケット ATb を使用できるのは RP，B のみである．

4.2.2 本人確認

End-User は発行したチケットを用いて RP に対し本人確認を行う．まず，チケット ATb とセッション鍵 k で暗号化した乱数 n2 を RP に送信する．RP はチケットを受け取ったのち，乱数 n3 の生成を行う．乱数 n3 と n2-1 をセッション鍵 k で暗号化し，End-User に送り返す．End-User が乱数 n3 を受け取ったのち，もう一度，RP に対し，n3-1 をセンッション鍵 k で暗号化し，通信を行う．End-User，RP 間での通信相手を確かめる場合，乱数をそのまま利用するのではなく，マイナス 1 を行った乱数を用いる．これは，一度使用した乱数を使用した場合，通信を第三者に盗聴された場合に通信内容が分かってしますためである．これによって，通信を盗聴され，再総攻撃を行われた場合でも，乱数の値が違うため，正しい通信相手だとは認識されない．

4.2.3 チケットの有効期限

本研究で構築するチケット方式でのチケットには有効期限が設定されている．有効期限は 3 日となっている．この期限設定はサーバがダウンしてから復旧にかかる期間が平均しても 1 日掛からない．長くても 2 日ほどで復旧が行われている．そのため，多めにとって 3 日とした．

4.3 システム概要

提案システムでは，OIDC の環境を構築し，構築した OIDC のフローに 4.2 節のチケット方式を追加することで，End-User，OP 間にてチケットの発行，そのチケットを用いて，End-User，RP 間で本人確認を行う．ユーザはこのシステムを使用するユーザ，使用するには PC を用いてアクセスを行う．本人確認サーバには Google を使用する．Google を OP として使用するには Google Developers Console にて登録の必要がある．サービスには，XAMPP を用いてローカル環境のサーバ上に PHP にて作成したものを RP として使用する．

5. 実装内容のシステム構造

5.1 実装概要

本研究では，OIDC の環境を構築し，構築した OIDC のフローに 4.2 節のチケット方式を追加する．追加したフローにて，End-User，OP 間にてチケットの発行．そのチケットを用いて，End-User，RP 間で本人確認を行うシステムを構築する．構築の際には，OP は Google Identity Platform，RP には 4.2 節のチケット方式を追加した RP を用いる

5.2 Google Developers Console

今回，OP には Google Identity Platform を使用する．Google を OP と使用する際には API キーを取得しなければならない．API キーがなければ RP が Google の ID・パスワードで本人確認ができず，RP として機能しない．API キーの取得には Google Developers Console にて新しい API プロジェクトを立ち上げ，API キーを取得する．

まず，サイト（https://console.developers.google.com/）にアクセスし，名前を入力し，プロジェクト作成を行う．その後，新しいプロジェクトが作成される．ここでは「API project」とした．以下の図 6 に示す．

図 6　作成したプロジェクト
Figure 7　Project created

作ったプロジェクトの中に入り，作成を選択する．選択すると以下の図 7 に示す選択肢が現れるので，「OAuth クライアント ID」を選択する．Google で OIDC を使用する場合，「OAuth2.0」を拡張して使用する．

図 7　選択肢
Figure 8　Choice

その後，ウェブアプリケーション名，API キーを使用するドメイン名の設定を行う．ここでは名前は「ウェブアプリケーション 2」とした．設定を終えると API キーのクライアント ID・クライアントシークレットが生成され，画面に表示される．この文字列を RP に相当する PHP に書き込むことで，Google の ID・パスワードで本人確認が行えるようになる．生成されたクライアント ID・クライアントシークレットを図 8 に示す．

図 8　生成されたクライアント ID，
クライアントシークレット

Figure 9　Generated client ID, client secret

その後，OAuth 同意タブに移動し，Google の ID・パスワードでユーザ確認時に表示されるサービス名，管理者メールアドレスの設定を行う．ここではサービス名を「apitest 持木」とした．以下の図 9 に示す．

図 9　OAuth 同意画面設定

Figure 10　OAuth consent screen setting

5.3 RP 構築

End-User，RP 間で本人確認を行うシステムの RP 構築を行う．XAMPP を用いてローカル環境をサーバとして機能させる．所定の場所に RP に当たる PHP にて作成した Web ページを置き，アクセスを行う．アクセスを行うと図 10 が表示される．

図 10　作成した web ページ

Figure 11　The created web page

TOP 画面に表示された「google openid connect login」をクリックすると，Google アカウントを選択する画面に遷移する．Google アカウント選択画面を図 11 示す．

図 11　Google アカウント選択画面

Figure 12　Google account selection screen

この画面にて，使用するアカウントを選択すると以下の図 12 が表示される．

図 12　サービス認可画面

Figure 13　Service authorization screen

図 12 の画面ではサービスが要求している属性情報を表示しており，ユーザがサービスに開示して良いなら許可ボタンをクリックし，属性情報を送信する．図 20 の場合，自分で作った API 名の「apitest 持木」というサービスがメールアドレス，プロフィールの開示を求めている．既存の OIDC を使ったサービスなら，この後，属性情報がサービスに送信され，ユーザに対しサービスが提供される．

5.3.1 RP に提案手法を追加

提案方式の 4.2.1 項に従い，ログイン処理を行った後，アクセストークンが発行され，乱数が生成される．それを用いて End-User，OP 間にてチケットが発行される．その後，チケットを用いて，End-User，RP 間で本人確認を行う．提案手法の 4.2.2 項の本人確認部分である．

6. 評価

6.1 評価項目

5.3 節で構築した RP の動作確認を行い，以下の点で評価を行う．

- 本人性を確かめるフローを追加したことで， OIDC が正常に機能しており，かつユーザがサービスを受けることができているか．追加した提案手法を用いた場合と通常の OIDC のフローとの差はないか

- OIDC における成りすまし問題についてプライバシー保護を踏まえ研究ができたか
- チケット方式を適応したことでOIDCのフローが複雑になっていないか，チケット方式だけでよいのではないか

6.2 評価方法

5.3 節で作成した RP を動かし，提案手法 4.2 節のチケット方式を応用した OIDC フローの動きの確認を行う．行った結果, アクセストークンが発行, 同時に乱数が生成され，画面に表示されればチケット発行手順は成功とする. また，表示されたチケット，乱数 n2，n3 を用いて，End-User，RP 間での変数が提案手法道理の値になっていれば本人確認ができたとする．

通常の OIDC フローと提案手法の 4.2 節を追加したフローとの差は，処理時間の差をもとに評価を行う．評価を行うのは提案手法 4.2.1 項と 4.2.2 項で追加したフローである，チケット発行と本人確認である．まず，通常の OIDC フローの動作確認を規定の回数行う．このときの処理に掛かった時間を記録する．次に，4.2.1 項と 4.2.2 項で提案した手法を追加したフローを同じく既定の回数行い，こちらも処理に掛かった時間を記録する．この時間の記録を比較し，処理時間に差ができているか確認を行う．

6.3 評価環境

評価に使用した, PC のスペックは OS:Windows 10 Pro 64bit, CPU:Intel Core i7-2637M 1.70GHz，メモリ:6GB である．また，ネットワーク構成を以下の図 13 に示す．

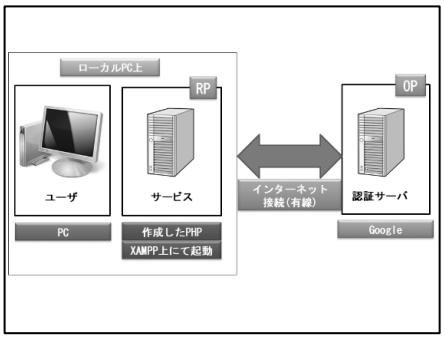

図 13　ネットワークの構成

Figure 14　Network configuration

6.4 評価結果

6.4.1 動作結果

5.3 節で作成した RP を動かし，提案手法 4.2 節のチケット方式を応用した OIDC フローの動きを確認した．結果，web ページにはログイン時に OP から発行されたアクセストークンと乱数が表示された．その後，それをもとにチケットが発行された．チケットを用いて，End-User，RP 間で本人確認を行う．

以上の結果から，アクセストークン，乱数，チケットが表示されたことによって，OP 側の本人確認サーバにて本人確認が行えない場合でも，End-User，RP 間でチケットを用いて本人確認ができ，かつ End-User が RP のサービスを享受できることができたと評価する．

6.4.2 処理時間の差

通常の OIDC フローと提案手法 4.2 節を追加したフローとの差は，処理時間をもとに評価を行う．まず，通常の OIDC フローの動作確認を規定の回数行う．このときの処理に掛かった時間を記録する．このときの記録を表 1 に示す．既定の回数は 50 とした．

表 1　通常 OIDC フローの処理時間（50 回）

Table 1　Normally OIDC flow processing time （50 times）

試行回数	処理時間(秒)		試行回数	処理時間(秒)
1	1.190		26	1.206
2	1.219		27	1.227
3	1.216		28	1.208
4	1.224		29	1.259
5	1.242		30	1.203
6	1.222		31	1.213
7	1.258		32	1.231
8	1.244		33	1.234
9	1.211		34	1.254
10	1.209		35	1.230
11	1.238		36	1.290
12	1.300		37	1.229
13	1.205		38	1.185
14	1.246		39	1.220
15	1.247		40	1.235
16	1.274		41	1.220
17	1.298		42	1.202
18	1.242		43	1.216
19	1.214		44	1.228
20	1.295		45	1.204
21	1.269		46	1.194
22	1.220		47	1.205
23	1.236		48	1.211
24	1.216		49	1.213
25	1.228		50	1.206

節 4.2 で提案した手法を追加したフローを同じく既定の回数行い，こちらも処理に掛かった時間を記録する．このときの記録を表 2 に示す．既定の回数は 50 とした．

表 2　提案手法の処理時間(50 回)

Table 2　Processing time of the proposed method (50 times)

試行回数	処理時間(秒)		試行回数	処理時間(秒)
1	1.270		26	1.299
2	1.284		27	1.205
3	1.297		28	1.295
4	1.281		29	1.293
5	1.309		30	1.201
6	1.298		31	1.216
7	1.262		32	1.229
8	1.261		33	1.261
9	1.308		34	1.218
10	1.296		35	1.268
11	1.275		36	1.200
12	1.294		37	1.524
13	1.226		38	2.031
14	1.213		39	1.237
15	1.201		40	1.214
16	1.218		41	1.261
17	1.322		42	1.263
18	1.299		43	1.297
19	1.299		44	1.226
20	1.247		45	1.240
21	1.208		46	1.269
22	1.301		47	1.233
23	1.926		48	1.245
24	1.233		49	1.260
25	1.216		50	1.299

また，表 1，表 2 からそれぞれの平均，最大値，最小値を抜き出し，標準偏差を求めた表を以下の表 3 に示す．

Vol.2018-DPS-174 No.13
Vol.2018-CSEC-80 No.13
2018/3/5

表 3　比較表

Table 3　Comparison table

	平均値(秒)	最大値(秒)	最小値(秒)	標準偏差
通常OIDCフロー	1.230	1.300	1.185	0.027320
提案手法	1.293	2.031	1.200	0.149375

7. 考察

本研究では，既存の技術である，PKCE ではサーバ側が止まった場合，ユーザがサービスを受けられない可能性がある.認可コードの比較検討を行うフローが機能しないため，比較検討が行えず，ユーザの本人性が確かめられない可能性がある．そのため，認可コード以外のものを使い，ユーザの本人性を確かめることによって，PKCE が使えない場合でもユーザがサービスを受けることができるようにするためのものである．

そこで，チケット方式を応用し，事前に End-User，OP 間にてチケットを発行する．発行したチケットを用いて End-User，RP 間で本人確認を行い，サーバ側が止まった場合でもユーザがサービスを受けられるようにすることによって，SSO の利便性の向上をプライバシー保護の観点から，システムの開発を目指した．

動作結果を見て，Google の ID・パスワードを用いてログインが行うことができたこと，本人確認，作成した API のサービス認可ができていること，OP からの属性情報も取得され，表示されていること，チケット方式を応用した本人確認フローができていることから，動作は正常に行われると判断する．

表 3 から，通常の OIDC フローでは 50 回の平均として，1.230 秒が処理に掛かっていることが分かった．また，提案手法の 4.2 節を追加した場合の処理時間の平均は，1.293 秒であったことが表 3 から分かった．通常の OIDC フローより提案手法を追加したフローでは 0.063 秒, 割合では 0.05%増加したことも実行結果の表 3 から分かった．また，表 3 から最大値は，通常 OIDC フローでは 1.300 秒，提案手法 4.2 節を追加したフローでは 2.031 秒，差は，0.731 秒であった．最小値は，通常 OIDC フローでは 1.185 秒，提案手法を追加したフローでは 1.200 秒，差は，0.015 秒であったことが表 3 から分かった．また，通常 OIDC フローと提案手法 4.2 節を追加したフローの標準偏差を計算した結果．表 3 から，それぞれ，0.027320，0.149375 であった．

処理時間の平均値が 0.05%増加したがほぼ変わらない．しかし, 最大値に 0.731 秒の差が出た. この理由としては，テスト環境がスタンドアローン環境で行ったわけではないため，処理中にバックグランドの処理が割り込んだと推察する．また，提案手法 4.2 節を追加したときのフローの標準偏差からも，0.149375 とバラつきが多い値が出ている．

8. まとめ

本研究では，既存の技術である，PKCE ではサーバ側が止まった場合，ユーザがサービスを受けられない可能性がある．認可コードの比較検討を行うフローが機能しないため，比較検討が行えず，ユーザの本人性が確かめられない可能性がある．そのため，チケット方式を応用し，事前に End-User，OP 間にてチケットを発行し，発行したチケットを用いて End-User，RP 間で本人確認を行う．これにより，OP 側の本人確認サーバがダウンした場合でもユーザがサービスを受けられるようにする．それによって，SSO の利便性の向上をプライバシー保護の観点から，システムの開発を目指した．

評価項目の一つ目については，8.章より動作は正常に行われると判断する．しかし，提案手法 4.2 節を追加したことで処理時間の増加が見られた．増加した処理時間の割合は 0.05%であり，実際に実装を行った場合でも，通常 OIDC フローとほぼ同じ処理時間で処理できると考える．

評価項目の二つ目でついては，チケット方式を応用したことで，End-User，RP 間では乱数を用いて通信を行っているので, 第三者が通信を盗聴し再送攻撃を行ったとしても，値が違うので，成りすましにはならないと考える．

評価項目の三つ目については，既存の本人確認が使えない場合にのみ使うこと，End-User が事前に使用する RP を選択し, チケットの発行を行っている必要があることから，問題ないと考える．

だが，利便性の向上という点においては，実際に使用する際には事前に OP 側にアクセスし，チケットを自分が使用する RP の数だけ発行する必要がある．また，標準偏差によるバラつきが多く，生成した乱数によっては処理時間が通常より長く掛かってしまう場合がある．だが，発行したチケットを用いて End-User，RP 間での本人確認は行えた．よって，一定の向上は図られたと思われる．また，プライバシー保護の観点からは，チケット方式を応用したため，第三者からの盗聴による，再送攻撃については防げると考える．今後，明らかにあった不便な点において解決策を模索していきたい．

参考文献

[1]今の ID 管理では不安あり、認証システム見直しは必至, http://itpro.nikkeibp.co.jp/article/COLUMN/20101122/354410/, (2018/1/10 閲覧)
[2]OpenID Authentication 2.0 - 最終版，http://openid-foundation-japan.github.io/openid-authentication.html(2018/1/10 閲覧)
[3]OpenID Connect 仕様書, http://www.openid.or.jp/document/(2018/1/10 閲覧)
[4]RFC 7636，https://tools.ietf.org/html/rfc7636(2018/1/10 閲覧)