# 情報流通連携のためのオープンな ID 連携プラットフォームにおける プライバシー保護機能の高度化の研究開発

Privacy Enhancement for Open Federated Identity/Access Management Platforms

**研究代表者**

中村 素典　国立情報学研究所

Motonori Nakamura　National Institute of Informatics

**研究分担者**

山地 一禎[†]　西村 健[†]　佐藤 周行[††]　岡部 寿男[‡]　崎村 夏彦[‡‡]　南 剛志[‡‡]　山崎 崇生[‡‡]

Kazutsuna Yamaji[†]　Takeshi Nishimura[†]　Hiroyuki Sato[††]　Yasuo Okabe[‡]

Natsuhiko Sakimura[‡‡]　Tsuyoshi Minami[‡‡]　Takao Yamasaki[‡‡]

[†]国立情報学研究所　[††]東京大学　[‡]京都大学　[‡‡]野村総合研究所

[†]National Institute of Informatics　[††]The University of Tokyo

[‡]Kyoto University　[‡‡]Nomura Research Institute

**研究期間**　平成 24 年度

## 概要

これまで一つの大学や企業など組織内に閉じていた認証基盤を、組織の壁を越えた社会的な ID 連携プラットフォームとしてオープン化し実用化するためには、個人情報の流通を制御しプライバシーを保護する技術の開発と仕様の標準化が重要である。本研究開発は、ID 連携におけるプライバシーの保護を考慮しつつ利便性を低下させない、さらには利便性の向上につながる安全なプライバシー情報の管理・加工・利用技術の開発と、その効果の実環境での評価を行う。

## １．まえがき

これまで一つの大学や企業などの内部に閉じて利用されてきた認証基盤を、組織の壁を越えた社会的な ID 連携プラットフォームとして実用化する動きが始まっている。このような認証基盤の社会基盤への展開の試みは、近年学術分野において活発に始められており、国を単位として欧米を中心に 30 を超える国々で構築が進められている。このような ID 連携基盤は認証フェデレーションあるいは単にフェデレーションと呼ばれるが、日本においては国立情報学研究所が中心となって学術認証フェデレーション「学認」を構築中である。

このように学術分野では国を単位としたフェデレーションとなっているが、当然のことながら、このような認証連携はフェデレーション内のみにとどまるべきものではなく、将来的には、フェデレーション間の連携、すなわち、国を超えた連携あるいは産学間の連携が浸透し、国や分野を問わず様々な用途に活用されていくことが望まれる。学術分野においては、オランダに拠点を持つ TERENA (Trans-European Research and Education Networking Association) が運営する REFEDS や、欧州 FP7 プロジェクト GEANT における活動の一つである eduGAIN 等において、フェデレーション間の認証連携に対する様々な検討や取り組みが進められており、日本からもその活動に一部参加し、共同して検討を進めているところである。

一方、民間では、OpenID を用いた ID 連携の枠組みが広がりを見せており、Google や Yahoo といったサービスプロバイダが ID プロバイダとなって他社の様々なサービスにアクセスできるようになり始めている。また、このような ID 連携における信頼性を確保するための枠組みの構築も始まっており、一定の保証レベルを担保するための信頼性評価を受けることで、米国 NIH（国立衛生研究所）を初めとする連邦政府系機関が提供するサービスへのアクセスが認められるようになってきた。OpenID では、新たに OpenID Connect と呼ばれる新たなプロトコルの策定が開始され、技術的にもさらなる信頼性と柔軟性の向上が実現されてきている。学術分野における SAML (Security Assertion Markup Language)を用いた ID 連携や民間の OpenID を用いた ID 連携は、ユーザに対する単なる認証の利便性を提供するだけでなく、オープンデータをはじめとする様々なデータを活用し流通を活性化する上でも重要な役割を担っていくことになると考えられる。

フェデレーションに代表される ID 連携プラットフォームを社会において本格的に実用化するためには、ID 連携プラットフォーム上でやりとりされる個人情報の流通を的確に制御しプライバシーを保護することが求められる。現在、広く用いられている SAML や OpenID といった ID 連携プラットフォームでは、個々のユーザの認証を行った後、そのユーザに関する属性情報を ID 管理側からサービス提供側に提示し、それに基づいて認可判断を行う形態をとる。ここで、属性情報には個人情報が含まれ、誰がどのようなサービスにアクセスしたかという情報はプライバシー情報として保護されるべきものであり、本来必要としない者に対して提供されるべきではないし、関係者に対しても必要以上に提供されるべきではない。しかしながら、現時点では、まずは認証連携を機能させ相互接続性を確保することが優先されているため、個人が特定可能な属性情報の安直な利用を前提とした仕組みが前提となっていることが多く、個人情報の送信の際のユーザ同意をどのように取得するか、という個人情報の開示に関する法制度等のみを考慮したものにとどまっており、プライバシー保護に関する考慮が十分ではない。

本研究開発では、これらの属性情報を適切に暗号化、仮名化（スードニマイズ, pseudonymize）した上で流通させる仕組みを標準化することにより、たとえば、ユーザがどのサービスにアクセスしたかが ID 管理側に分からない、あるいは、どのユーザがアクセスしてきたかがサービス提供側に分からない（しかし正しいユーザであることは保証され、インシデント発生時には関係者の協力によりユーザの特定可能）、といった機能を持つオープンなプラットフォームを開発し、ユーザのプライバシー保護を担保した上

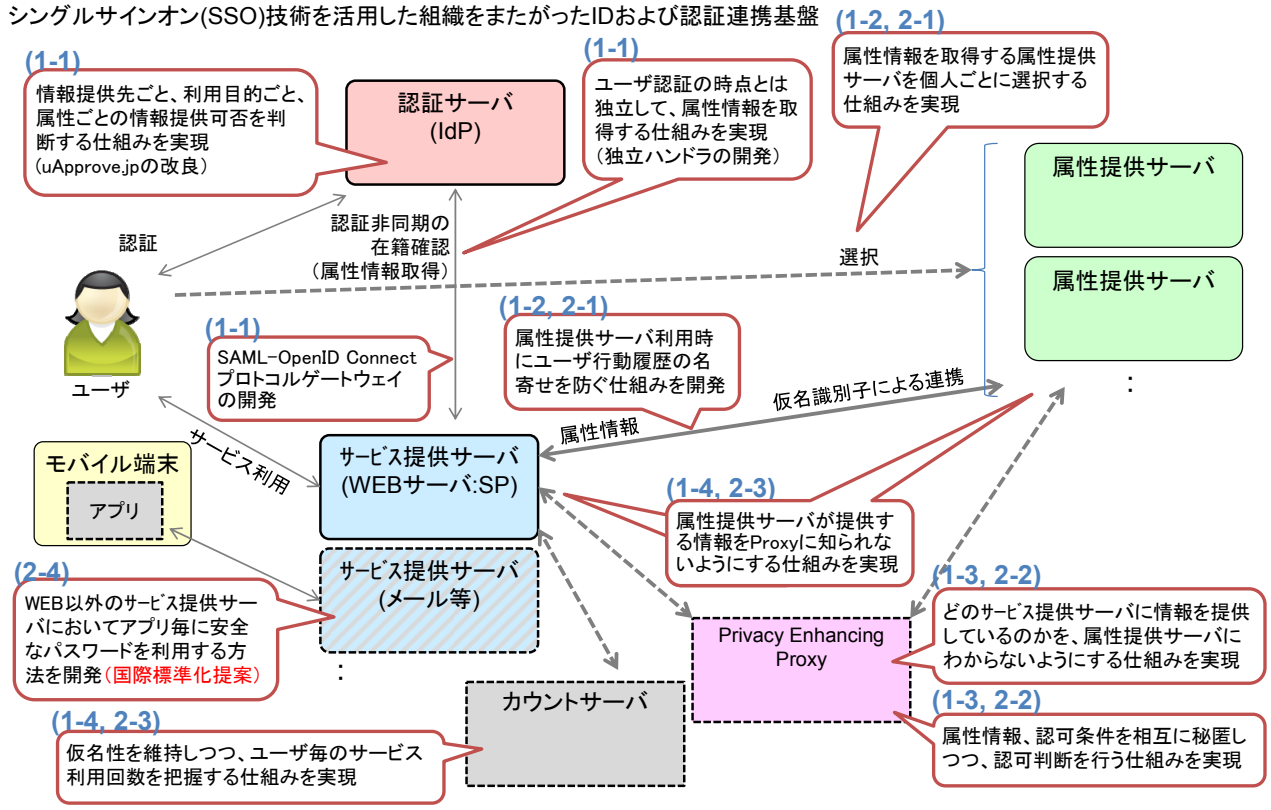



図１　プロジェクト研究開発課題の関係

での自由な情報の流通・連携を実現する。このようなプライバシー保護機能の実現は、SAML や OpenID といった個々の認証連携プロトコルの中に閉じた話題ではなく、信頼構築のための統一的な基準の下で異なるプロトコルを連携させた、さらに幅広い連携を視野に入れた場合にも重要なものとなることから、本研究開発においても、SAML と OpenID との連携を例として、複数のプロトコルにまたがったプライバシー保護機能の実現にも取り組む。

## ２．研究開発内容及び成果

本研究開発におけるプロジェクト全体の実施内容（研究開発課題）は次の通りである（計８つ）。各研究開発課題の間の関係を図１に示す。

１）Shibboleth 上でのプライバシーに配慮した属性連携に関する研究開発
（研究開発内容）
１－１）情報提供可否に関する研究開発
１－２）属性提供サーバ個人設定に関する研究開発
１－３）属性提供サーバに対するプライバシー保護にする研究開発
１－４）プロキシに対するプライバシー保護に関する研究開発

２）OpenID Connect 上でのプライバシーに配慮した属性連携に関する研究開発
（研究開発内容）
２－１）属性提供サーバ個人設定に関する研究開発
２－２）属性提供サーバに対するプライバシー保護にする研究開発
２－３）プロキシに対するプライバシー保護に関する研究開発
２－４）WEB 以外のサービス提供サーバに同様の仕組みを提供する仕組みの研究開発

課題１－２と２－１、１－３と２－２、１－３と２－３については、それぞれ同一の問題に対して SAML/Shibboleth と OpenID Connect という異なるプロトコルによるアプローチをとる。SAML/Shibboleth 側（１－２～１－４）では分散クレームモデルによる属性情報取得、OpenID Connect 側（２－１～２－３）では集約クレームモデルによる属性情報取得を前提とした（図２）。なお、以下では、IdP (Identity Provider)とは認証を行うサーバ、AP (Attribute Provider)とは、属性情報を提供するサーバ、SP (Service Provider)とは、サービスを提供するサーバのことを指す。

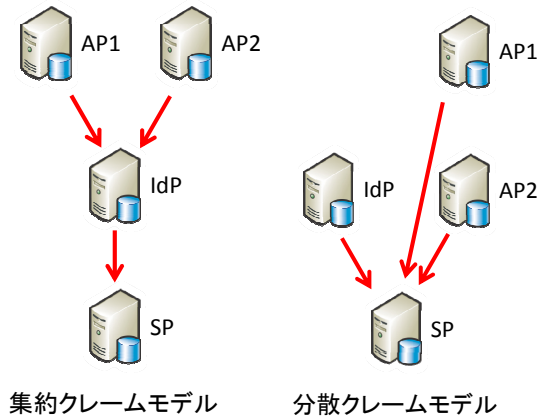



図２　２つのクレームモデル

以下、研究開発課題ごとに順を追って説明する。

## 2.1 情報提供可否に関する研究開発（課題１－１）

本研究開発では、ユーザ認証に基づいて提供される属性情報について、情報提供先に関する細かな制御を実現するための仕組みを開発する。個人情報保護の観点からは、どのような情報をどこに対して提供することを認めるのか、ということをユーザが指定でき、その指定に基づいて適切に制御することが求められる。

属性情報の取得・提供方式には、様々なバリエーションがあり、例えば、ユーザ認証と同じタイミングで属性情報を一度だけ提供する場合もあれば、継続的な費用の支払いが生じる契約を学生割引で提供を受ける場合に必要となる定期的な在学確認のようなユーザ認証とは非同期に属性情報の取得を許すような場合も考えられる。前者の場合は、ユーザによる ID およびパスワードの入力操作に連続して、提供される属性情報に関する内容の確認と、提供方法についての指示を行うことが可能であるのに対して、後者の場合は、ユーザによる ID およびパスワードの入力操作のタイミングとは無関係（非同期）に属性情報のやりとりが行われることになるため、事前にどのサービス(SP)に対してどのような属性情報を提供しても良いかについて確認し、それに基づいた属性情報の提供が必要となる。

現状では、世界的に見ても、前者のユーザ認証に同期した一度限りの属性情報の提供が主流であるが、今後の高度な ID 連携基盤の活用を考慮すると、後者のような利用形態についても、検討を進めておくことが非常に重要である。特に、上記で例示した学生割引のような形態は、学術分野における ID と民間が提供するサービスとの連携として、今後大きな需要が見込まれているものの一つであることから、そのようなサービスの実現性の実証実験も踏まえ、SAML と OpenID という異なるプロコトル間の連携も考慮した検討も行う。

SAML では従前よりこのような属性情報の利用形態をサポートしているが、OpenID においても最近策定された OpenID Connect においてこのような属性情報の利用形態に柔軟に対応できるようになったことから、これらのプロトコル連携について取り組むことは新しい試みであり、今後 OpenID Connect が民間の ID 連携で広く用いられるプロトコルとなることを考えると、本研究開発が先導的な役割を果たすものになると期待される。

### 2.1.1 SAML と OpenID Connect の比較

歴史的には OpenID より SAML の方が古く、本研究開発で取りあげる OpenID Connect は、いくつかある OpenID のバージョンの中でも現在標準化が進みつつある最新の仕様である。OpenID Connect 自体は認証技術であり認可部分には OAuth 2.0 を利用する。このために正確には OpenID Connect with OAuth 2.0 と呼ばれる。一方、Shibboleth は学術分野において SAML を用いた認証・認可のオープンソースのプロジェクトであり、日本における学術フェデレーションである学認においても使われている。現在は、SAML 2.0 の利用が主流となっており、今回の開発でも SAML 2.0（Shibboleth 2.0）の仕様を利用している。表１に両者の比較を示す。

このように SAML も OpenID Connect もともに Web サービスを前提とした認証・認可の SSO（シングルサインオン）技術であることから、プロトコル変換によってゲートウェイの実装が可能となる。組み合せ方により、OpenID Connect の RP (Relying Party, SP と同意)から SAML の IdP を使うためのプロトコル変換と、SAML の SP から OpenID Connect の OP (OpenID Provider, IdP と同意)を使うためのプロトコル変換、の 2 種類が存在するが、今回、本開発においては前者の OpenID Connect の RP から SAML の IdP を使うためのプロトコル変換を行うゲートウェイにターゲットを絞って開発した。これにより OpenID Connect に対応した民間サービスを、学認の ID を持つユーザが利用できるようになる。民間サービス側から見ると、本ゲートウェイを利用することで学認の属性を取得して、例えば学割を提供するようなサービスが実装可能となる。

### 2.1.2 システム構成

本研究課題の解決方法を提示するプロトタイプとして、一つのシステムを構築した。本システムは学術分野の ID 及び属性を、民間が提供するサービスにて利用するためのものである。先にも述べたように、学認では SAML/Shibboleth を利用している。一方、民間のサービスでは OpenID 系が一般的に広く利用されており、最新の OpenID Connect は今後民間サービスではスタンダードになると考えられる。

実装したシステムは、学認の SAML の ID を保有する利用者が OpenID Connect 民間サービスを利用する際に異なるプロトコルを意識せずに利用するための、SAML/OpenID Connect ゲートウェイサーバと、認証非同期に属性提供を行う機能、デモ用サービスから構成される（図３）。

表１ SAML と OpenID Connect との比較

| 項目 | SAML/Shibboleth | OpenID Connect with OAuth 2.0 |
|---|---|---|
| ID プロバイダ | IdP（Id Provider） | OP（OpenID Provider） |
| サービス提供 | SP（Service Provider） | RP（Relying Party） |
| 認可 | Shibboleth で行う | OAuth 2.0 を利用 |
| フェデレーション（SP/RP 登録） | 公開鍵とメタデータの交換が必要<br>Shibboleth のインストールと設定が必要になる | 事前に RP の情報を OP に登録しクライアント ID 等を取得<br>標準の HTTP/HTTPS 環境のみ |
| 採用状況 | 学術分野で世界的に利用<br>エンタープライズ分野でも利用 | Yahoo!や Google 等で採用<br>コンシューマ分野で利用が広がる<br>エンタープライズ分野でも注目 |
| モバイル対応 | 不可 | 可能（※ 色々と考慮は必要） |
| 言語 | XML<br>形式を重視したマークアップ言語 | JSON（JavaScript Object Notation）<br>軽量なデータ記述言語 |
| 情報 | http://www.internet2.edu/shibboleth | http://openid.net/connect/ |

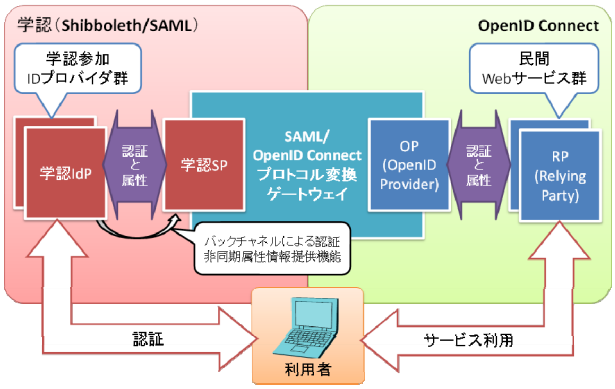



図３　システム概要

### 2.1.3 プロトコル変換ゲートウェイの設計

SAML/OpenID Connect プロトコル変換ゲートウェイは図４のような利用手順において適切にプロトコル変換を含めた処理を行う必要がある。プロトコルの手順はもっと複雑であるが、ここでは簡略化して記載している。②③⑧⑨はリダイレクトによる移動である。以下、これを「研究開発 1-1-1」とする。

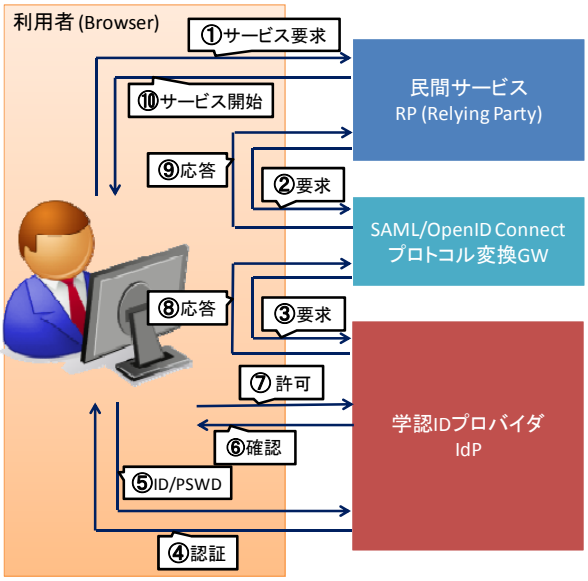



図４　SAML/OpenID Connect GW を利用する手順

### 2.1.4 バックチャネル属性提供機能の設計

認証時にフロントチャネルを利用し認証結果と一緒に属性を取得することは従来より可能であったが、今回はこれに加えて、一ヶ月等の比較的長期にわたり、その期間内であればゲートウェイシステムが認証無しで属性を返す仕組みを実装した。もちろん、認証なしで属性を取得するには、最初に正しい認証が完了しており正しいアクセストークン等を保持している場合に限られる。

表２に属性提供フローを示すが、その中で、①⑤についてはプロトコル変換ゲートウェイが担うため、詳細についてはそちらで説明を行う。

②③④および利用者からの属性提供状況の確認については、ベースとして Shibboleth を使うが、Shibboleth が提供していない機能については Shibboleth を拡張することにより実現する。以下に述べるように、Shibboleth SP の拡張、Shibboleth IdP の拡張、新たな UI の提供、の３つの部分から構成される。

表２　SAML/OpenID Connect GW を利用する手順リスト

| 手順 | | 説明 |
|---|---|---|
| ① | サービス利用要求 | 利用者は RP に対してサービス利用を要求 |
| ② | GW へ認証要求 | RP は GW の認証へリダイレクトさせる |
| ③ | IdP へ認証要求 | GW は IdP の認証へリダイレクトさせる |
| ④ | 利用者へ認証要求 | IdP は利用者へ認証を要求する |
| ⑤ | ID とパスワード入力 | 利用者は ID とパスワードを入力する（属性利用の確認も） |
| ⑥ | 属性利用の確認 | IdP は利用者に対して確認画面を表示 |
| ⑦ | 属性利用の許可 | 利用者は確認画面に同意する |
| ⑧ | GW で認証結果取得 | GW は IdP の認証結果を取得する |
| ⑨ | RP へ認証結果を返す | GW は RP に対して認証結果を返す |
| ⑩ | サービス利用開始 | RP は取得した認証結果を取得してサービスを開始 |

②において、Shibboleth SP には、IdP の entityID に基づいて属性情報の要求を行う機能はあるものの、認証と連動して働くもの(Simple Attribute Aggregation)しかなく、任意のタイミングで呼び出すことができない。そこで、本件では Shibboleth SP に属性情報要求を任意のタイミングで呼び出せる属性情報要求ハンドラを開発した。GW は Shibboleth SP の属性情報要求ハンドラの URL にアクセスすることで IdP に対し属性情報要求を行える。これを「研究開発 1-1-2」とする。

③④に関して、本来 Shibboleth IdP には属性送信に対するユーザからの同意取得機能は存在しないが、uApprove.jp (http://www.gakunin.jp/docs/fed/uapprove-jp) という IdP プラグインによってこれを実現することができる。ただし従来の uApprove.jp は認証と同時、つまりフロントチャネルでの属性送信に対する同意しか想定していないため、本研究課題のユースケースのような GW から IdP へのバックチャネルによる属性情報の問合せ(AttributeQuery)に応答する上ではユーザが同意していない情報を提供してしまうという問題があった。そこで本件では、AttributeQuery への応答を構築する際に属性情報の取捨選択が適切に行われるように、バックチャネルによる属性取得処理を改修した。以後、当該改修を「uApprove.jp の持続的 AttributeQuery 対応」と呼ぶ。これを「研究開発 1-1-3」とする。

最後に、利用者からの属性提供状況の確認について説明する。従来フロントチャネルでの属性送信および同意では送信のタイミングが利用者にとって明確であったため、同意をとった上で属性情報を送信することは容易であり問題にならなかったが、認証処理のタイミングとは独立したバックチャネルによる属性取得では利用者が関知しないところで行われるため、これを利用者に対して可視化する必要があった。この機能は前項と同様 uApprove.jp では提供されていないものであったため、uApprove.jp を拡張する形で実現した。これを「研究開発 1-1-4」とする。

デモ用 RP システムでは、購入画面が開かれる時にバックチャネル経由で学生かどうかを示す属性（eduPersonAffiliation）を取得し、学生であれば学割を有効にしている。OpenID Connect の Endpoint としては Userinfo Endpoint を利用している。

### 2.1.5 動作環境

今回のシステムではゲートウェイシステムとデモ用 RP システムの両方を 1 サーバ内で実現した。

表 3　開発システムの環境

| プロジェクト | バージョン | 補足 |
|---|---|---|
| Ruby | 1.9.2p290 | ソースよりインストール |
| Rails | 3.2.12 | gem によりインストール |
| SQLite | 3.6.20 | システム標準をそのまま利用 |

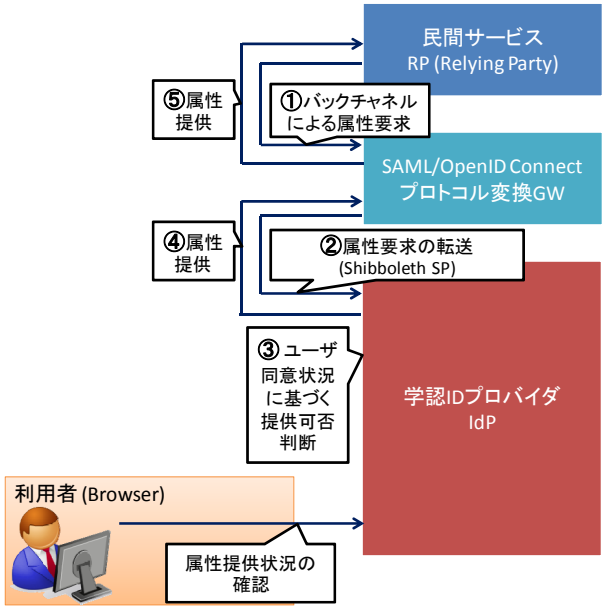



図 5　バックチャネル属性提供機能のフロー

### 2.1.6 成果及び結果

一度限りの契約による学割サービス、および継続的な課金（月毎の課金等）のための定期的な在学確認を必要とする学割サービスの両者を念頭において、SAML と OpenID Connect が連携して実用サービスを構築するために必要となるシステム（図 5）として、以下のソフトウェアを開発し、目標とする機能が実現されていることを確認した。

1-1-1)　uApprove.jp の持続的 AttributeQuery 対応の実装
Shibboleth IdP においてユーザ同意に基づく属性情報の提供を行うためのプラグインである uApprove.jp を、本課題において要求される仕様に適合するよう改良を行った。ユーザが同意すべき内容も異なるため、ユーザに提示するメッセージも変更している。これは、大学等で運用される認証サーバ(IdP)に組み込んで利用する。

1-1-2)　Shibboleth SP 属性情報要求ハンドラ
ユーザに対する認証処理を伴わずに（認証非同期に）バックチャネル経由で属性情報を取得するためのインターフェース(API)を実現した。これは、次項のプロトコル変換ゲートウェイに組み込んで利用するものである。

1-1-3)　プロトコル変換ゲートウェイ
SAML（Shibboleth)と OpenID Connect を連携させるためのプロトコル変換ゲートウェイを構築した。昨年度策定された OpenID の新しいプロトコルである OpenID Connect に対応している。

1-1-4)　属性送信済み SP 一覧ページの提供
ユーザが認証非同期な属性情報の送信に同意した場合、実際にどのような頻度で属性情報が送信されているかを随時把握できることが望ましいことから、属性情報の送信状況を確認するための機能を追加した。ユーザは、このページに認証を経てアクセスできるとともに、既に承諾した同意を取り消すことも可能である。

なお、本課題の成果については、平成 25 年 5 月に開催された情報処理学会 IOT 研究会にて「学割サービス実現のための SAML-OpenID ゲートウェイの試作」という題目で発表した（予稿は、情報処理学会研究報告として発刊）。

## 2.2 属性提供サーバ個人設定に関する研究開発（課題１－２）

現在の Shibboleth の実装において属性情報を付加的に提供するための属性提供サーバ(AP)を活用する枠組みでは、サービス提供側であるサービス提供サーバ(SP)が属性プロバイダを指定する方法しか提供されておらず、それも固定的であるため、ユーザが利用する属性プロバイダに関して自由度がない。しかし、Shibboleth の実装のベースとなっている SAML 標準の仕様において、属性プロバイダはユーザに関連する様々な属性情報を、ユーザが属する機関とは独立して提供することも可能であることから、本来、属性プロバイダはユーザ毎に選択できるようになっているべきであり、どの属性プロバイダを利用するかはがサービスプロバイダによってのみ決まりユーザが選択できないことは将来的な実サービスへの展開の上で大きな制約となる。

また、属性プロバイダを利用する上でのもう一つの制約として、ユーザの識別問題がある。現在の実装では、まず ID プロバイダがユーザ認証を経てサービスプロバイダに対してユーザ識別子を提供し、サービスプロバイダがそのユーザ識別子を用いて属性プロバイダに属性情報を問い合わせる、という仕組みが用いられている。このため、ID プロバイダとサービスプロバイダ、および属性プロバイダは必ずユーザ固有のグローバルな識別子を共有しなければならず、SAML や OpenID 等による ID 連携プラットフォームが本来サポートしているはずの、仮名による（プライバシー保護に考慮した）アクセスが利用できない。このため、ID プロバイダの属性提供機能を属性プロバイダとして分離した ID 連携プラットフォームの本来の柔軟性が損なわれ、実社会への展開における属性プロバイダの活用が大きく制限される懸念がある。そこで、本研究開発では、ID 連携プラットフォームにおいて属性プロバイダを利用する際にも仮名識別子を用いた仮名によるアクセスが可能となる技術を確立し、Shibboleth の拡張を行う。

### 2.2.1 フロントチャネルを用いた AP からの属性取得(研究開発 1-2-1)

#### 2.2.1.1 Shibboleth の認証方式

Shibboleth は SAML 1.0 及び 2.0 をサポートし、ユーザが特定の Web サイト（サービス提供サーバ：SP）へアクセスした時にブラウザへリダイレクト指示により認証サーバ(IdP)に制御を移して認証を要求し、認証後、再びブラウザへのリダイレクト指示によりサービス提供サーバ(SP)に制御を移しその際に認証結果を返す。 この時の認証シーケンスを図 6 に示す。

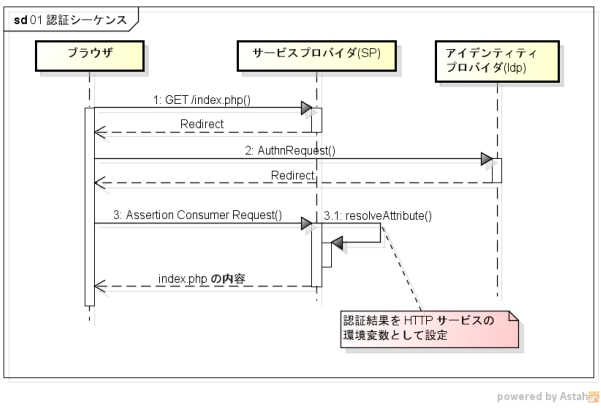



図６　認証シーケンス

## 2.2.1.2 Shibboleth の認証シーケンス

最初にブラウザから SP へ HTTP GET アクセスが行われ、SP がブラウザの持つクッキー（_shibsession_***）からログインしていないことを判断すると、認証シーケンスを開始する（1:GET /index.php() の箇所）。SP はユーザに Idp での認証を促すため、AuthnRequest(※1) 形式のリクエストフォーマットを作成しブラウザへリダイレクトを要求する。ブラウザはこれに従い IdP へ AuthnRequest 要求を HTTP POST メソッドを用いて行う（2:AuthnRequest() の箇所）。IdP では AuthnRequst を受け取りブラウザが持つクッキー（_idp_session）からログインしていないことを判断するとユーザに ID とパスワードを入力するよう促す。ユーザが認証に成功すると IdP は Assertion Consumer Service(ACS) フォーマットの XML に認証結果を記載しブラウザへリダイレクトを要求する。IdP は、IdP 自身が持つ属性情報を認証結果に加えて送信することができる。最後にブラウザはリダイレクトにより再度 SP へ接続し SP が属性情報を解析した後に HTTP サービスの環境変数として認証結果および属性情報を設定し本来アクセスのあった URL のコンテンツを返す（3:Assertion Consumer Request の箇所）。このような方法で Shibboleth では認証情報および認証に基づいた属性情報の受け渡しが行われる。以上のように、Shibboleth IdP は AP として機能を含んでいるが、属性情報の提供を専門に行う IdP を特に区別して AP と呼ぶ。これにより、自分の所属している機関（大学）以外の組織から属性を取得できるようになる。

※1　AuthnRequest：AuthnRequest は SAML で規定され認証や属性、権限付与関連の情報を符号化した XML 形式のデータ

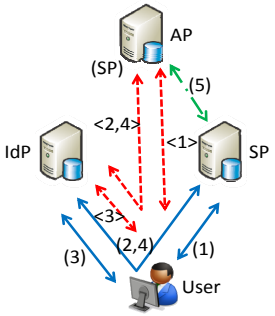
## 2.2.1.3 AP からの属性取得における 2 つの方式

AP から属性を取得する方式には下記の 2 通りの方法が考えられる（図 7）。

① バックチャネル方式
② フロントチャネル方式

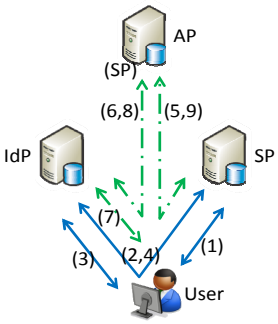
バックチャネル方式では SP と AP が直接 HTTP（SOAP）で通信し、属性を取得する方式である。フロントチャネル方式では SP と AP は直接通信せず、ブラウザを介しての通信で属性を取得する方式である。先に説明した Shibboleth の認証方式と類似のプロトコルとなる。属性取得にブラウザが介在するため、SP からユーザの ID が渡されない場合、かつ AP が直接ユーザを認証できない場合は、追加のフローとして（AP が SP となり）IdP に認証要求をすることができる。

バックチャネル方式は、SP と AP が直接通信を行う形となり、ブラウザからのアクセスで用いられる HTTPS のポート番号 443 と異なるポート番号が用いられることも多いことから、ファイアウォールにてブラウザからのアクセスとは別に通信許可を与える設定が必要となる場合がある。そのため、AP が大学内からのアクセスのみに限定されている場合は SP が大学外にあると SP から AP にアクセスできないという問題が発生する。また、SP がグローバルなユーザ ID（例えば eduPersonPrincipalName）を AP に渡さないとそのユーザの属性を取れないといった問題がある。逆にフロントチャネル方式には、ブラウザから SP および AP にアクセスでき、SP を利用しているユーザ ID とは別の ID で AP から属性を取得できるといった利点がある。




<1>: access to AP as a SP
<2>: redirect to IdP
<3>: authentication
<4>: assertion with attributes (ePPN)
(1): access to SP
(2): redirect to IdP
(3): authentication
(4): assertion with attributes (ePPN)
(5): back-channel query with ePPN

バックチャネルを用いた従来方式




(1): access to SP
(2): redirect to IdP
(3): authentication
(4): assertion with attributes (ePTID1)
(5): front-channel query (without ID)
(6): redirect to IdP
(7): authentication (skipped by SSO)
(8): assertion with attributes (ePTID2)
(9): assertion with attributes (without ID)

フロントチャネルを用いた提案方式

図７　２通りの属性プロバイダとの連携方式

### 2.2.1.4 Shibboleth が提供するバックチャネル方式の仕組み

Shibboleth ではバックチャネル方式で属性情報を取得する仕組みが備わっており、Simple Attribute Aggregation と呼ばれている。この仕組みは認証と連動するものである。バックチャネルを使った属性取得は Shibboleth の認証方式 に記載した認証シーケンスの 3.1:resolveAtrribute() の中で行われ、以下のようなシーケンス図になる（図８）。

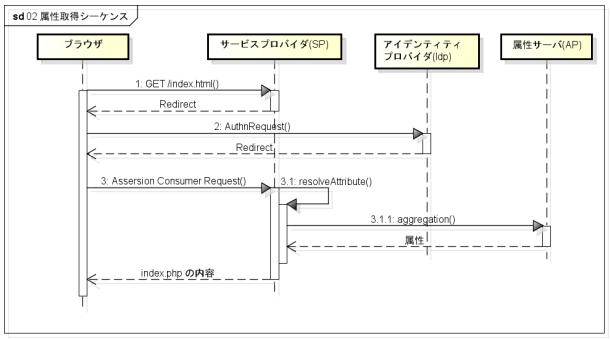



図８　属性取得(Simple Attribute Aggregation)を含んだ認証シーケンス

Shibboleth 設定ファイル（shibboleth2.xml）に Simple Attribute Aggregation が定義されていた場合、SP はそこで entityId として指定された AP へ SOAP リクエストを使って属性情報取得を行い、Shibboleth の認証方式同様 HTTP サービスの環境変数として属性情報を設定する。

### 2.2.1.5 Shibboleth 方式を用いた場合の問題点

Shibboleth のバックチャネルを用いた属性取得は、認証(IdP)と属性取得(AP)をそれぞれ異なる管理主体が運用することが可能となるため、共同研究プロジェクトなどの機関をまたがるグループ情報の管理等に利用できるようになっている。

ただし、このバックチャネルを使った属性情報取得を行う場合、SP がユーザ（ブラウザ）に代わって直接 AP とやりとりするため、グローバルなユーザ ID が SP 側に与えられる必要があるため、プライバシー上の懸念が残る。また、AP からの属性取得が認証直後にしか行うことができず認証と独立して任意のタイミングで行うことが不可能である、さらに、属性取得する AP は SP の設定ファイルに設定しなければならずユーザが取捨選択することができない、という問題がある。

### 2.2.1.6 フロントチャネルを用いた属性取得の設計

前節で述べたグローバルなユーザ ID の問題を解消するためには認証シーケンスで利用されているようなリダイレクトを用いたフロントチャネルを用いた属性情報取得が有効と考えられる。

フロントチャネルを用いた属性情報取得（フロントチャネルアグリゲーション）は Shibboleth の機能として備わっていないが、次のような方式で実装できる。

- SP 管理者がフロントチャネル属性取得を行うハンドラを Shibboleth 設定ファイルに定義する
- ユーザが認証後の遷移先 URL(target パラメータ）にフロントチャネル属性取得ハンドラを指定する
- SP が認証後、認証した IdP の entityId をフロントチャネル属性取得時の AuthnRequest に含める（新たにハンドラを実装し、その中で行う）
- AP が認証した IdP への再認証をユーザに促す（同じセッションであるため、ログイン画面は表示されない）
- SP が AP から取得した属性情報を認証した IdP の属性とマージした上で HTTP サービスの環境変数に設定する（ACS を改修し、その中で行う）

これらの方式を前提としたフロントチャネルアグリゲーションの全体像となるシーケンスを図９に示す。AP での処理についてなど詳細は後述する。

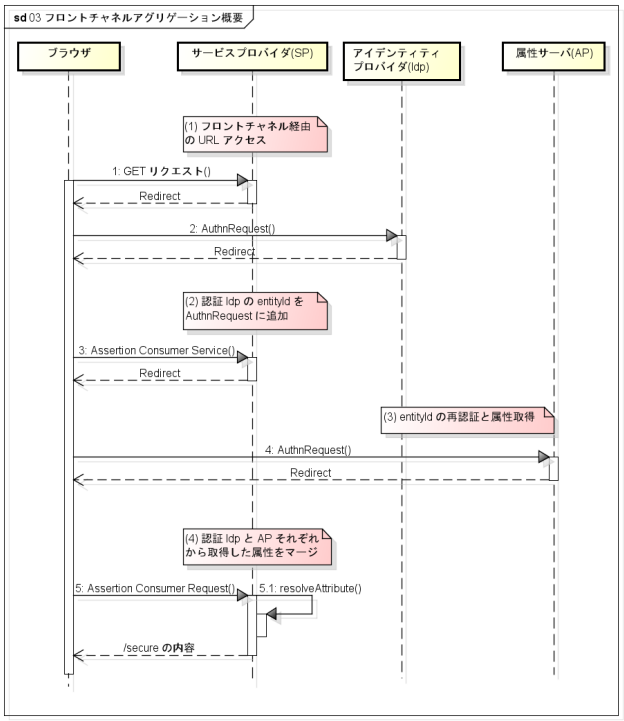



図９　フロントチャネルアグリゲーション全体シーケンス

## 2.2.2 AP セレクタ（研究開発 1-2-2）

### 2.2.2.1 AP セレクタの設計

複数ある AP の中からユーザが選択して利用することを可能にする機能が AP セレクタである。AP セレクタは SP の画面上（ブラウザ）に表示され、ユーザに AP を選択させた後、その AP に対して SAML のリクエストを投げることで、AP から SP に属性を提供させることを可能にしている。実際のリクエスト生成に関しては、先に説明したフロントチャネル属性取得機能が利用できる。

### 2.2.2.2 AP セレクタのユースケース

AP セレクタではユーザ（SP を利用する人）に対しての振舞いと管理者（SP を管理している人）に対しての振舞いの２つがある。ユーザには AP の一覧から利用したい AP のみを表示させる「AP を絞り込ませる」ユースケースと、利用したい AP を選ぶ「AP を選択する」の２つユースケースがあり、管理者には SP で利用する属性を持つ AP を予め設定しておき、AP を限定してユーザに表示させる「利用する AP を決める」ユースケースがある。これらの AP セレクタの振舞いを図１０ に示す。

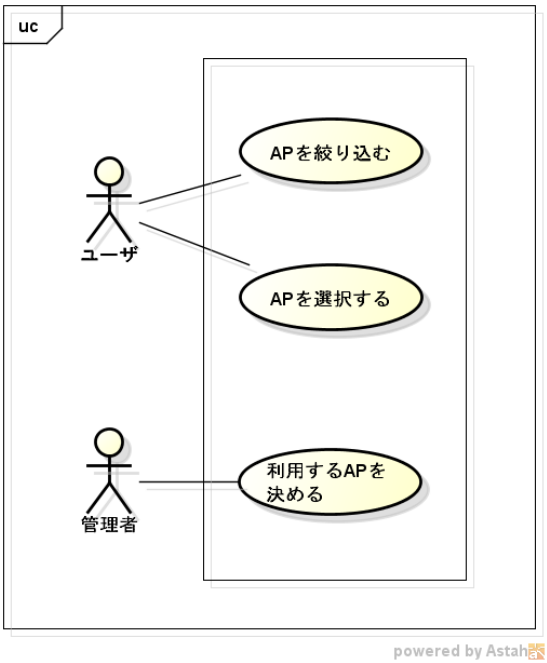



図１０　APセレクタのユースケース図

### 2.2.2.3 APセレクタの構成

通常のシボレスでの利用はSPとIdPの２つが必要だが、APセレクタを利用するためにはそれに加え、APセレクタ、APの２つが必要になる。その関係を図１１に示す。SPにAPセレクタの表示制御を行う設定ファイル（APセレクタ設定js）があり、APセレクタにはAPセレクタの実体であるjavascriptファイルを置く。

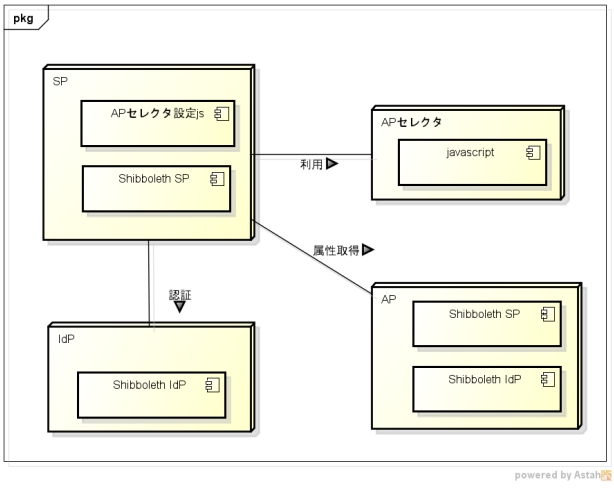



図１１　APセレクタの構成図

### 2.2.3 成果及び結果

提案方式を実現するために、以下に示すソフトウェアを開発し、目標とする機能が実現されていることを確認した。

### 2.2.3.1 フロントチャネルを用いたAPからの属性情報取得

本研究開発では、フロントチャネルアグリゲーションの基点となるShibboleth SPハンドラを実装した。また、APにおけるShibboleth IdPを拡張し、以下に述べるIdP再認証を実現した。他の部分、特にフェデレーションに参加するIdPの改修は不要である。

#### 2.2.3.1.1 Shibbolethの Simple Attribute Aggregation類似の属性取得フロー

実装したハンドラおよびAPによる効果を示すために、従来のSimple Attribute Aggregation類似の認証+属性取得フローがこのシステムで実現できることを示す。まず、フロントチャネルアグリゲーションを用いて属性情報を取得するためにSP管理者はこれを処理するハンドラを定義し、URLに関連づける（ここでは/MAPをURLとして仮定）。

ユーザはこのハンドラを使うために/MAPへアクセスするが、SSO用のURL（通常/Shibboleth.sso/Login）を使ってIdPでの認証を経る必要があるため、/MAPへのアクセスをtargetパラメータ（認証後の遷移先URL）に設定する。

さらにこの指定に加えて、属性取得後にアクセスしたいURLも/MAPのtargetにURL Encodeした上で設定する。これらを踏まえ、以下のようなURLにアクセスする。

https://[SP ホスト]/Shibboleth.sso/Login?target=/Shibboleth.sso/MAP%3FentityID%3Dhttps%3A%2F%2F[AP ホスト]%2Fidp%2Fshibboleth%26target%3D/secure

このURLへのアクセスにより「SPアクセス -> IdP認証 -> APでの属性取得 -> 本来のアクセス先へ」といったアクセスが可能になる。

#### 2.2.3.1.2 認証IdP情報のAPへの伝達

ユーザが上述のURLへアクセスし、IdP認証に成功(シーケンスの2:AuthnRequest() の箇所)すると次にフロントチャネルを使ってAPから属性を取得するシーケンスに移るが、APでは属性を返すためにはユーザの認証情報が必要となる。

SP毎に異なる仮名識別子を使うという前提なのでSPから何らかのユーザIDを渡すことは不可能である。SPからAPへは、SPが認証したIdPのentityIdを渡す。具体的には、3:Assersion Consumer Service() の箇所で次のURLリクエスト（/MAPへのアクセス）時に発行するAuthnRequestにentityIdを含めておく。

AuthnRequestの仕様にはもともとIDPListというリクエスタが信頼しているIdPに対し、プレゼンタの本人認証を行うよう指定するXML要素が存在するため、entityIdはこの要素に設定する。

#### 2.2.3.1.3 IdPでの再認証

グローバルなユーザIDの不在により、属性要求を受けたAP側で再度認証が必要になる。さらに詳細なシーケンスを以下に示す。

IdPでの初回の認証が完了しAPへ接続が行われるとAPではRemoteUserログインハンドラ機能によりこのアクセスを仲介しAuthnRequestからentityIdを取り出す。

Shibbolethではsso時のURLにentityIDパラメータの指定があればそこに認証をリクエストするため、entityIDパラメータに取り出したIdPのentityIdを指定し、自ホスト(AP)のSPへリダイレクトする。

このリダイレクトにより IdPで認証し、APのURL(自ホスト)へ戻ってくるようにする。URLは以下のようになる。

https://[AP ホスト]/Shibboleth.sso/Login?entityID=[Idp ホスト]&target= /idp/Authn/RemoteUser

なお、この認証は最初に認証した IdP と同じ entityId であるため、クッキーにセッション情報(_idp_session)が残っており、ユーザへのログイン要求は行われず成功する。/idp/Authn/RemoteUser へのリダイレクトが行われると IdP は属性情報を取得し SP へリダイレクトを要求する（図１２）。

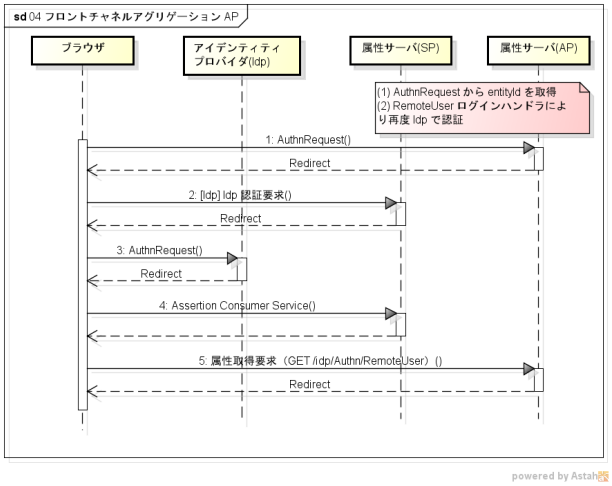



図１２　再認証時のシーケンス図

### 2.2.3.1.4 認証 IdP と AP それぞれから取得した属性の統合

SP の ACS がリダイレクトにより呼び出されると SP ではクッキーに格納されたセッション情報から認証した IdP が返した属性情報を取り出し、AP から取得した属性情報とマージする。

最後にマージ済みの属性情報を HTTP サービスの環境変数に格納し target パラメータに定義された（本来アクセスしたい）URL にリダイレクトして一連の処理を終える。

このようなフロントチャネル経由の属性取得方式によりバックチャネルとは異なり SP 側でユーザの一意となる情報を持つ必要がなくなる。

### 2.2.3.1.5 フロントチャネルアグリゲーションを利用した認証非連動な属性取得

フロントチャネルアグリゲーションを用いた属性取得は既述したようにバックチャネルの抱える問題を解消することができる他、認証と連動しない属性取得および複数 AP からの属性取得も容易に行うことができる。

研究課題 1-1-2 の Shibboleth SP 属性情報要求ハンドラと同じ理屈であるが、フロントチャネルアグリゲーションの機能をハンドラとして構築したため、このハンドラへ追加的にアクセスさせることで認証と連動しない属性取得が可能となる。さらにハンドラに与えるパラメータの中に AP の entityId があるため、属性取得する AP を動的に変更することが可能となっている。

## 2.2.3.2 AP セレクタ（属性情報提供サーバの選択）

### 2.2.3.2.1 設計方針

AP セレクタについては、以下に示す方針に基づいて設計し、実装を行った。

（１） 利用
AP セレクタは SP に組込み利用する。

（２） 利便性
SP 側で簡単に AP 選択画面の利用を可能にするため、AP 情報と AP 選択を表示させるライブラリをウェブブラウザで標準に動作させる。SP 側ではこのライブラリを呼びだすだけで容易に AP 選択画面の表示が行え、表示／非表示の制御も SP 側で行えるようにする。

（３） 変更
AP は増えること、AP が提供できる属性に変更が可能なことを考慮し、提供するライブラリは SP に配置するのではなく、AP サーバ内に配置する。これによりサーバ内にある javascript を変更すれば AP セレクタを利用している SP 全てに適用されるようにする。

（４） AP 選択方法
AP の一覧を画面に表示し、ユーザが利用したい AP を選択させる。多くの AP の中から目的にあう AP を見つけやすくするため、持っている属性に限定した AP に絞り込む機能を提供する。
また、AP を選択した状態を Cookie に記録しておき、再度選択の必要がないようにする。

（５） 属性の取得方法
１つの AP から属性を取得するには AP へリクエストするだけでよいが、複数の AP が選択された場合は、複数の AP に対してリクエストを行わないといけない。この AP へリクエストする制御を AP セレクタ側で受け持ち、選択した AP とリクエストした AP の情報を Cookie に持つことで可能にする。

### 2.2.3.2.2 設計における考慮点

**（１）Javascript の利用**

AP をブラウザ内で選択させるため、クロスブラウザで動作する Javascript を利用する。その理由を下記に示す。

① フロントチャネルを有効に使え、ブラウザからリクエストを AP に送信できる
② AP の情報をリアルタイムに更新できる
（AP の増加を直ぐに AP セレクタに反映することが可能）
③ SP ごと AP セレクタの設定を行える
（SP には不要な AP を表示させないように制御できる）

**（２）厳密な CSS の利用**

AP セレクタは SP の画面に組み込まれることを考慮する必要がある。AP セレクタのカスケード・スタイル・シート(CSS)が SP のデザインに影響を及ぼさないようにし、また SP の CSS から影響されないようにするため、AP セレクタの CSS のセレクタ（このセレクタは AP セレクタのセレクタではなく CSS のスタイルを適用する対象を選択するものを意味している）に厳密な親子関係を設ける。その具体的な例を下記に示す。

① SP のデザイン（CSS）への影響を防ぐ
例えば、a { color: red} より　div a { color: black }、div a { color: black }より div#selecter a.title{color: blue}の方が最優先に適用される(文字の色は青になる)。このように挿入する AP セレクタの最上位タグから適用したいタグまでの親子関係を全て定義したスタイルは他のデザインへ影響を与えない。
※同じ id 等を利用しない場合に限る

② SP からのデザイン（CSS）の影響を防ぐ
{margin, padding, position, color, background, border, cursor, width, height, font}を定義することで他で定義された CSS からによるデザイン崩れを防ぐ。

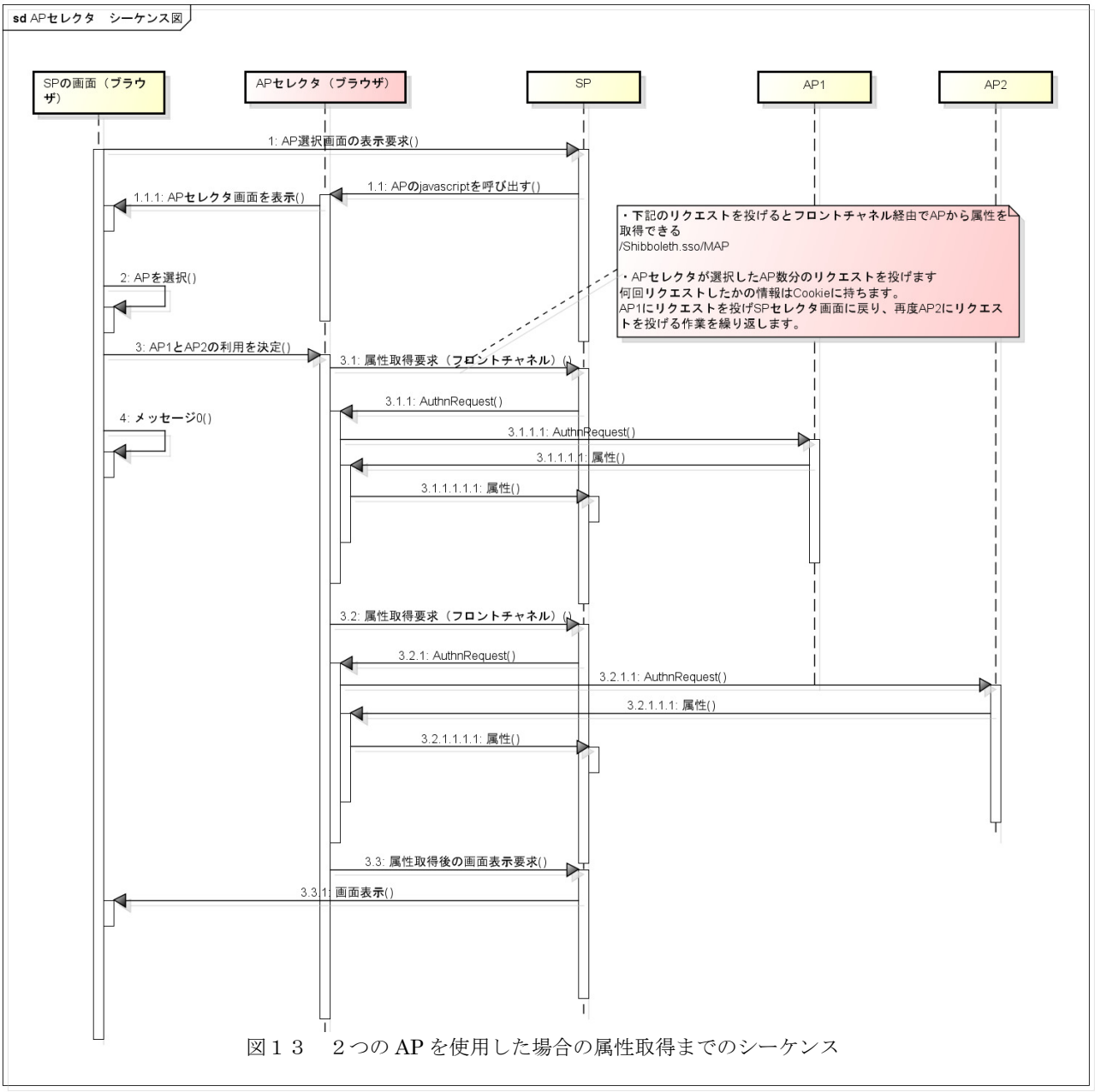



図１３　２つの AP を使用した場合の属性取得までのシーケンス

powered by Astah

**（3）SP 側の記述**

SP 管理者が AP セレクタの Javascript の取り込みと最小限のコードの記述で AP セレクタを動作させる。

## 2.2.3.2.3 処理フロー

２つの AP を使い属性を取得するまでの処理の流れを図１３に示す。ブラウザ (ユーザエージェント)、SP、IdP、AP にて、どのような順番で処理が行われているが把握できる。

このシーケンスで重要な部分は以下の表４のとおりである。

表４

| 名前 | 説明 |
|---|---|
| 3 AP1 と AP2 の利用を決定 | ユーザが AP セレクタの画面に表示されている AP の一覧から AP１と AP2 を選択し、これらの AP から属性の取得を実行する。 |
| 3.1 属性取得要求（フロントチャネル） | AP セレクタ（ブラウザ）から SP に対して**AP１**から属性を取得する要求をする |
| 3.1.1 AuthnRequest<br>3.1.1.1 AuthnRequest | SP から IdP へ認証要求である SAML の AuthnRequest をブラウザ経由で送信する。<br>これは SAML の通常の認証である |
| 3.1.1.1.1 属性<br>3.1.1.1.1.1 属性 | **AP１**からブラウザ経由で SP へ**AP１**の属性が渡る。 |
| 3.2 属性取得要求（フロントチャネル） | AP セレクタ（ブラウザ）から SP に対して**AP２**から属性を取得する要求をする |
| 3.2.1.1.1 属性<br>3.2.1.1.1.1 属性 | **AP２**からブラウザ経由で SP へ**AP２**の属性が渡る。 |

| | |
|---|---|
| 3.3 属性取得後の画面表示要求 | AP1 と AP2 から属性を取得できたことがわかると **AP セレクタ**が SP の画面へ遷移させる |

なお、本課題の成果のうち前者 1-2-1 については、IEEE の COMPSAC 国際会議の MidArch ワークショップに “Privacy Preserved Simple Attribute Aggregation to Avoid Correlation of User Activities Across Shibboleth SPs”という題目で投稿し、採択された（今回は 7 月に京都にて開催）。

## 2.3 属性提供サーバに対するプライバシー保護にする研究開発（課題１－３）

全体構想の中で新たに導入する認証連携プロキシにおいて、属性提供サーバが発行する属性アサーション（ユーザの属性情報が含まれる）をサービス提供サーバに伝達する際に、具体的なサービス提供サーバの存在を属性提供サーバには見せないようにすることで、プライバシー保護を実現するための仕組みを、以下の 3 課題に分けて開発した。

### 2.3.1 属性提供サーバに対しサービス提供サーバを隠蔽する機構（１－３－１）

ユーザが認証を経てサービス提供サーバ(SP)にアクセスする際に、どのサービス提供サーバにアクセスしたかというプライバシー情報を ID 管理側(IdP)あるいは属性提供サーバ(AP、属性)にわからないようにしようとするための仕組みを実装する。

例えば、従来からの対面でのサービス提供において大学が発行する学生証を提示して学生割引を受けるモデルでは、大学はどのようなサービスに対して当該ユーザが学割サービスを受けたかについて関知しない。このような仕組みをオンライン上で実現することに相当する。

IdP によるユーザ認証に基づいて属性提供サーバ AP（あるいは IdP）が発行する属性アサーションをサービス提供サーバ SP に伝達する際に、具体的なサービス提供サーバの SP 存在を属性提供サーバ AP（あるいは IdP）に見せないようにすることで、プライバシー保護を実現する。ここでのサービス提供サーバは任意のものを許すわけではなく、事前に信頼関係を構築しているサービス提供サーバのうちの何れかであることが前提であることから、認証連携プロキシにおいて新たなアーキテクチャを設計し実装する。

### 2.3.2 属性情報と認可条件を相互に秘匿する機構（１－３－２）

属性情報を用いてサービス提供サーバが認可判断を行う場合において、認証連携プロキシを経由することで、属性提供サーバが提供する個々の属性情報の詳細をサービス提供サーバに対して秘匿し、サービス提供サーバの認可の条件を属性提供サーバに対して秘匿したいという要求に対応するための仕組みを実装する。例えば、ある企業（サービス提供者）が、学生の大学（属性提供者）での成績のうち、英語の成績と数学の成績の和がある閾値以上であれば、採用プロセスの一部を免除するような場合を考える。大学が学生の全科目の成績の素点をサービス提供サーバに伝えることを避けるためには、英語と数学の和が閾値を超えているかどうかの結果だけを企業に伝えれば良いが、ここでさらに、企業としては、そのルールを大学に開示することは避けたい（多数の教科のうち英語と数学の成績しか判定に用いていいないという認可の条件は大学側には開示したくない）という相反する要求がある場合を考える（図１４）。認証連携プロキシに対応するための機能を組み込むことによって、両者の要求をかなえることを可能にする。

従来手法として、Blind Signature, Zero Knowledge Proof などを利用した方式が検討されているが、パフォーマンス面や実装の容易さ、特許など知財上の問題などを抱えている。本研究開発で実現する方式は、これらに比べて実装が容易であるのみならず、パフォーマンスも高く、オープンで特許の問題も抱えないという利点がある。

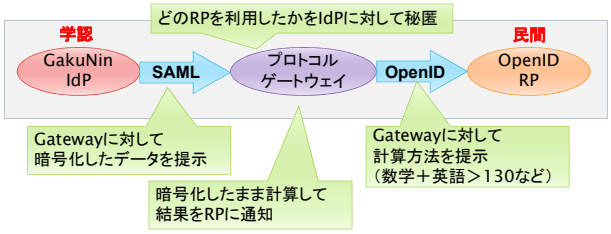



図１４

### 2.3.3 仮名性を担保しつつユーザの同一性を確認する機構（１－３－３）

属性提供プロバイダを介した認証連携において、ユーザの仮名性を担保しつつユーザの同一性を確認し、同一ユーザにおける利用回数等を制限する機構を実装する。例えば、学割サービスを提供する際に、1 人のユーザに対する割引の適用回数にある上限を設けておきたい状況を考える。IdP 提供側からは、プライバシー保護の観点からユーザを特定する情報となるグローバルユニークな識別子情報を出すことは望ましくないため仮名識別子が用いられる。そのため、ユーザの同一性が確認できず、同一のユーザが制限を超えた回数の割引を受けられてしまう抜け道が生じうる。そこで、属性提供サーバからは仮名識別子を提供しつつ、同一のサービスに対してその利用回数等を制限することができる仕組みを設計し、実装する。

### 2.3.4 成果及び結果

本課題 1-3-1 と次項の課題 1-4 は、同時に実現されるべきことを考慮すると、機能が密接に絡み合うことから、それぞれを担当する京都大学と東京大学で密に連携することとし、検討の結果、それぞれで方式の異なるプロキシシステムを実現し、比較することとした。また、1-3-3 については、その内容から東京大学側の課題とするのが適当と考え、担当を整理し直した。

以下に示すソフトウェアを開発し、目標とする機能が実現されていることを確認した。

#### 2.3.4.1 属性情報が秘匿されたまま中継される匿名化プロキシシステム（１－３－１）

要件を満たすプロキシシステムとして、次の 2 種類のプロトコルに基づくソフトウェアをそれぞれ開発した。

1. DH 鍵交換型プロキシ（図１５）
2. カスケード型プロキシ（図１６）

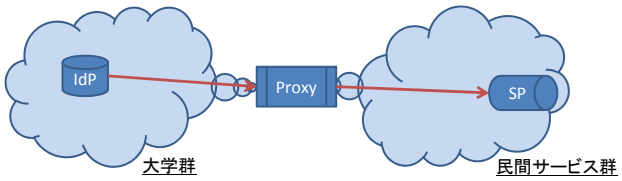



図１５

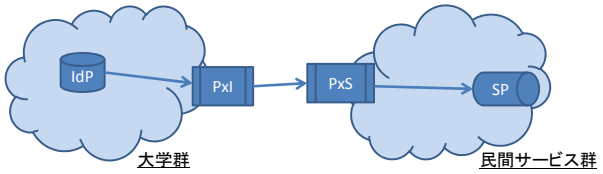



図１６

DH 鍵交換型プロキシとは、IdP と SP が Diffie-Hellman 鍵交換プロトコルにより動的に秘密鍵を共有し、それを用いてプロキシに秘匿した形で属性情報を送る方式である。DH 鍵交換型においては、以下の性質が保証される。

・SAML 同様、SP と IdP はフェデレーション内で属性情報を受け渡すことができる
・IdP は認証と属性情報を要求した SP を知ることはできない
・SP は属性情報を返答した IdP を知ることができない
・Proxy は属性情報の中身を知ることができない(Diffie-Hellman 鍵交換の性質による)

これに対しカスケード型プロキシは IdP 側プロキシと SP 側プロキシの二つの独立に運用されるプロキシからなり、IdP 側プロキシと SP、SP 側プロキシと IdP がそれぞれ公開鍵暗号を用いて動的に秘密鍵を共有し、それにより IdP 側プロキシ、SP 側プロキシのいずれに対しても秘匿された形で属性情報が送られる。

Cascade 型プロキシにおいては、以下の性質が保証される。

・SAML 同様、SP と IdP はフェデレーション内で属性情報を受け渡すことができる
・IdP は認証と属性情報を要求した SP を知ることはできない
・SP は属性情報を返答した IdP を知ることができない
・2 つのプロキシ PxS、PxI は属性情報の中身を知ることができない
・2 つのプロキシ PxS、PxI のどちらか片方において中間者攻撃を受けた際に、IdP と SP、および残った側の Proxy はその事実を知ることができる

## 2.3.4.1.1 DH 鍵交換型プロキシ

DH 鍵交換型プロキシのプロトコルにおいては、IdP と SP の間に 1 つ Proxy と呼ばれるサーバを置き、SP と IdP 間では Diffie-Hellman 鍵交換アルゴリズムを用いることで、情報の秘匿を行う。

DH 鍵交換型プロキシのプロトコルを図１７に示す。図の記号の意味は表５のとおりである。

表５ DH 鍵交換型プロキシのプロトコルにおける記号

| | |
|---|---|
| p | SP の生成する乱数。SP のみが知り得る |
| q | IdP の生成する乱数。IdP のみが知り得る |
| A | 開示される属性情報 |
| g | 十分大きい素数を法とする有限体の原始元。フェデレーション内で共有 |
| $K=g^{pg}$ | Diffie-Hellman 鍵交換で交換される共有鍵 |
| $e_k(A)$ | A を k で暗号化したもの |
| $e_s(p)$ | SP の秘密 p を、SP だけが解ける暗号 $e_s$ で暗号化したもの |

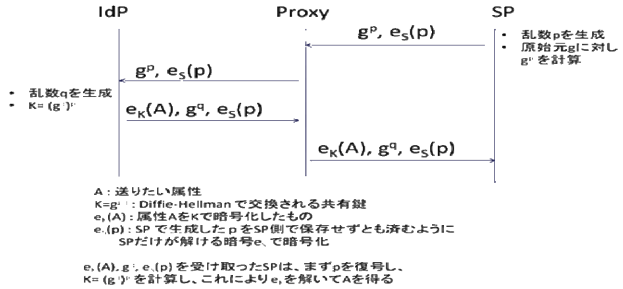



図１７　DH 鍵交換型プロキシのプロトコル

Proxy は、SP に対しては IdP として動作し、IdP に対しては SP として動作することで、IdP に対して属性情報の要求元を秘匿すると同時に、SP に対して認証情報の提供元を秘匿する。Proxy は SAML プロトコルで利用される HTTP Redirect によって、SP から IdP への認証要求を送信し、本来の SAML プロトコルにより規定された HTTP Redirect、POST、Artifact 等によって IdP から SP への認証結果と属性情報を返送する。

認証要求と認証結果の往復の際、SP は同時に、2 つの情報を送る。

- $g^p$ ... Diffie-Hellman 鍵交換における共有鍵の片側(SP 側)
- $e_s(p)$ ... Diffie-Hellman 鍵交換における SP 側の秘密を SP 向けに暗号化したもの

ここで、「X 向けに暗号化したもの」は通常、「X のみが知る対称鍵で暗号化したもの」もしくは「非対称鍵暗号における X の公開鍵で暗号化したもの」のいずれかとなる。

これらを受け取った IdP は、その場で共有鍵のもう片側である q を生成し、共有鍵である $K=g^{pg}$ を生成する。この鍵を用いて、IdP は新たに以下の情報を生成し、SP へ返送する。

- $g^q$ ... Diffie-Hellman 鍵交換における共有鍵の片側(IdP 側)
- $e_k(A)$ ... 属性情報 A を共有鍵 $K=g^{pg}$ によって暗号化したもの
- $e_s(p)$ ... Diffie-Hellman 鍵交換における SP 側の秘密。SP から送信されたものをそのまま返信する

SP は自分が送信し、そのままの状態で返ってきた $e_s(p)$ から p を復号し、$g^q$ を復元します。この値と新たに IdP から送られた $g^p$ から共有鍵 $K=g^{pg}$ を取得、それによって暗号化された $e_k(A)$を復号し、最終的に A を得る。

なお、この過程においては、SP は自分自身の中にこの認証要求に関する追加の状態を保存しないことを仮定している。仮に SP がこの認証プロトコルの過程で自身の秘密である p を覚えておける場合には $e_s(p)$は省略でき、プ

ロトコルも簡略化される。具体的には、Key-Value Store 型のデータベースを SP が拡張モジュールに提供している場合、SAML プロトコルの ID を元にした key と p を value にしたデータをデータベースへ保存することで、属性情報が到達した際に p を取得することができるため、$e_s(p)$を Proxy および IdP へ送信してそれが認証後に返すよう要求する必要はない。なお、同様の省略は後述する Cascade 型プロトコルにおいても可能である。

DH 鍵交換型においては、以下の性質が保証される。

- SAML 同様、SP と IdP はフェデレーション内で属性情報を受け渡すことができる
- IdP は認証と属性情報を要求した SP を知ることはできない
- SP は属性情報を返答した IdP を知ることができない
- Proxy は属性情報の中身を知ることができない (Diffie-Hellman 鍵交換の性質による)

ただし、Proxy が悪意を持った第三者によって操作された際などにおいて、中間者攻撃により交換される属性情報の盗聴や改竄が生じる可能性は排除できない。SP と IdP の双方で、共有鍵である $K=g^{pg}$（ないしそのハッシュ値）をログとして記録しておくことにより、万一中間者攻撃が行われた場合でも疑義が生じた時点でそのことを検出できる点で、一定の抑止力にはなるが、中間者攻撃のリスクを完全に排除するためには、次項で述べる Cascade 型プロトコルを用いるのが適当である。

### 2.3.4.1.2 Cascade 型プロキシ

Cascade 型プロキシのプロトコルは、プロキシを 2 つ (PxS, PxI) 配置して共有される秘密を分割することで、DH 鍵交換型において回避できない（1 箇所への）中間者攻撃の検知を実現する。

Cascade 型プロキシのプロトコルを図１８示す。図中の記号の意味は表 6 のとおりである。

表 6　Cascade 型プロキシのプロトコルにおける記号

| | |
|---|---|
| $K_1$ | SP, PxI において有効な共有鍵 |
| $K_2$ | PxS, IdP において有効な共有鍵 |
| $e_{SP}(A)$ | A を SP のみが復号できる鍵で暗号化したもの。$e_{IdP}$, $e_{PxI}$, $e_{PxS}$ も同様 |
| $K_1 \oplus K_2$ | $K_1$ と $K_2$ のビットワイズ排他論理和。<br>$K_1 \oplus K_2 = K_2 \oplus K_1$ と $A \oplus K \oplus K = A$ が成り立つ |

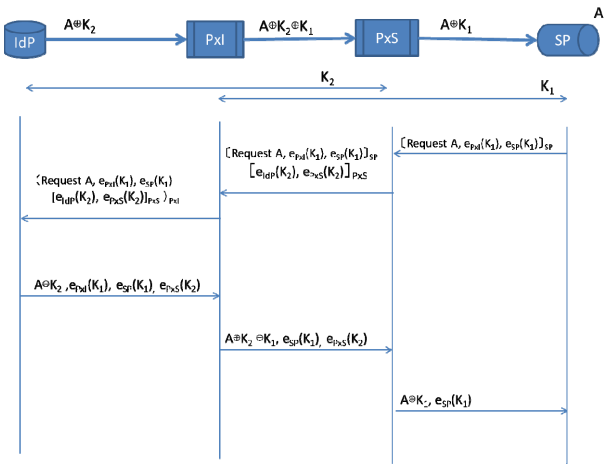



図１８　Cascade 型プロキシのプロトコル

本プロトコルにおける 2 つの Proxy PxS および PxI の基本動作は DH 鍵交換方式と同様であり、SP 側からは IdP として動作し、IdP 側からは SP として動作する。

SP が認証と属性情報を要求を送信する際、SP は $K_1$ を生成し、それを SP 自身と PxI のみが読めるように暗号化する（$e_{PxI}(K_1)$, $e_{PxS}(K_1)$が該当）。さらに SP 自身の署名を付加して PxS に送出する。

PxS は $K_2$ を生成し、それを PxS 自身と IdP のみが読めるように暗号化し PxS 自身の署名を付加した上で、SP から送られてきた暗号化された情報に加えて PxI へ送信する。

往路においては、PxI は SP から送られてきた情報の署名を検証した上で除き、PxS から送られてきた追加の情報と共に、認証要求を IdP へそのまま送る。

認証後、IdP は認証要求に付された PxS の署名を確認の上、PxS から与えられる $K_2$ により属性値 A を暗号化し $(A \oplus K_2)$、PxI へ送る。PxI は $K_1$ を復号することで PxI から送られてきた$(A \oplus K_2)$を更に暗号化する $(((A \oplus K_2) \oplus K_1) = A \oplus K_2 \oplus K_1)$。

PxS において、PxS は $K_2$ を知ることができるため $A \oplus K2 \oplus K1$ に対して $K_2$ を適用し、ビットワイズ排他論理和の性質から$((A \oplus K_2 \oplus K_1) \oplus K_2) = A \oplus K_1$ を得る。

SP においては $K_1$ を得られることから、$A \oplus K_1 \oplus K_1 = A$ が得られる。

DH 鍵交換型プロキシのプロトコル同様、SP、 PxS、PxI において当該セッション中に状態を保存する場合、上記のフローで自分自身の秘密鍵向けに向けに暗号化された各種情報は省略できることがあります。後述する Cascade 型プロキシのプロトコルの実装においては、実際に、PxS と PxI においてこの省略により実装を簡略化している。

Cascade 型プロキシにおいては、以下の性質が保証される。

- SAML 同様、SP と IdP はフェデレーション内で属性情報を受け渡すことができる
- IdP は認証と属性情報を要求した SP を知ることはできない
- SP は属性情報を返答した IdP を知ることができない
- PxS、PxI は属性情報の中身を知ることができない
- PxS、PxI のどちらか片方において中間者攻撃を受けた際に、IdP と SP、および残った側の Proxy はその事実を知ることができる

以上の設計仕様に基づく「属性情報が秘匿されたまま中継される匿名化プロキシシステム」を、既存の SAML 実装と整合する形で実装した。実装を行う上で利用したソフトウェアは下記の通りである。

- Ubuntu 12.04.1 LTS
  - Linux ディストリビューション
- Apache 2.2.22-1ubuntu1
  - Web サーバ
- Tomcat 6.0.35-1ubuntu3.1
  - Java サーブレットコンテナ。Shibboleth IdP の動作に必要
- Shibboleth IdP 2.3.8
  - Tomcat と協調動作する SAML IdP の Java 実装
- Shibboleth SP 2.4.3
  - Apache と協調動作する SAML SP の C++ 実装

- OpenSSL 1.0.1-4ubuntu5.5
  - セキュリティ機能を提供するライブラリ。秘密鍵を利用する際に用いる
- SimpleSAMLphp 1.10.0
  - Proxy 機能を提供する SAML 認証の php 実装
- Firefox 19.0+build1-0ubuntu0.12.04.1
  - ブラウザ

開発した 2 種類のプロキシシステムは、商用のクラウド環境にインストールして、シナリオ通りに動作することを検証した。

### 2.3.4.2 属性条件と認可条件を相互に秘匿する機構（１－３－２）の設計と実装

前述の課題 2 に対する解として、SAML を通信プロトコルに採用し、Shibboleth IdP (Identity Provider)および SP (Service Provider)でフェデレーションが構築されているという条件のもとで、属性情報に基づく認可を行う際に、属性情報をスクランブルし、スクランブル後の属性情報に基づいて、独立したサーバで判定を行いサービスプロバイダに判定結果を知らせることで、ID 管理サーバ、判定を行うサーバ、サービスプロバイダの間で属性情報と認可条件を相互に秘匿する機構を提供するためのソフトウェアシステム（以下、「属性情報と認可条件を相互に秘匿するソフトウェア」と）として設計し、開発した。

開発したソフトウェアは、以下の各機能を備える。

○　サーバ隠蔽機能

ID 管理サーバ(IdP)からサービス提供サーバを隠蔽する機能を提供する。サービス提供サーバに対しては ID 管理サーバとして認証要求を受け取り、ID 管理サーバに対してはサービス提供サーバとして、同じ内容の認証要求を ID 管理サーバに送り直し、受け取った結果を元のサービス提供サーバに返す。このとき、元のサービス提供サーバの情報を ID 管理サーバに渡すことはない。本機能は、独立したサーバ(以下、プロキシサーバ)として機能する。

○　属性情報スクランブル機能

サービスプロバイダは、属性情報をスクランブルするための情報と、スクランブルされた b 属性情報を使って判定を行うための認可情報を生成する。プロキシサーバは、上記情報を ID 管理サーバへ転送する。ID 管理サーバは、サービスプロバイダが指定し、プロキシサーバが転送した、属性情報のスクランブル情報に基づき、属性情報をスクランブルし、プロキシサーバに送信する。

○　暗号化機能

属性情報スクランブル機能で生成したスクランブル情報、認可情報を適切に暗号化することにより、情報を必要としないサーバに対して内容を隠蔽する。属性情報をスクランブルするための情報は、サービス提供サーバと ID 管理サーバのみが知ればよく、認可情報は、サービス提供サーバと判定を行うプロキシサーバのみが知る。

○　判定機能

スクランブル化された属性情報と認可情報にもとづいて、サービスの可否を表す判定結果を算出し、サービス提供サーバに送信する。

開発したシステムは、ID 管理サーバ(IdP)、プロキシサーバ(Proxy)、サービス提供サーバ(SP)から構成されている。

1.　ID 管理サーバ (IdP)

ユーザの認証を行い、ユーザの属性情報を払い出すという通常の IdP の機能に加えて、サービス提供サーバから渡された、暗号化されたスクランブル情報に基づき、整数である属性のスクランブルを行い、暗号化してプロキシサーバに渡す。

2.　プロキシサーバ(Proxy)

ID 管理サーバとサービス提供サーバの通信を仲介し、ID 管理サーバにサービス提供サーバの情報を渡さずに、ユーザ認証と属性情報のサービス提供サーバへの受け渡しを行うとともに、属性の問い合わせの際に、ID 管理サーバからスクランブルされた属性が渡された場合は、サービス提供サーバから渡された、暗号化された認可条件によって判断を行い、結果をサービス提供サーバに渡す。

3.　サービス提供サーバ(SP)

サービス提供サーバ上のアプリケーションからの問い合わせ内容に応じて、ID 管理サーバに渡す暗号化されたスクランブル情報と、プロキシサーバに渡す暗号化された認可情報を生成し、まとめてプロキシサーバに認可判断を依頼し、その結果に基づいて、サービス提供の可否を判断する。

サービス提供サーバは、プロキシサーバサーバと通信し、プロキシサーバは ID 管理サーバと通信することで、ID 管理サーバ、判定を行うプロキシサーバ、サービス提供サーバの間で属性情報と認可条件を相互に秘匿しつつ、システム全体では属性による認可処理を行うことが可能になる。

本システムの動作の概要を図１９に示す

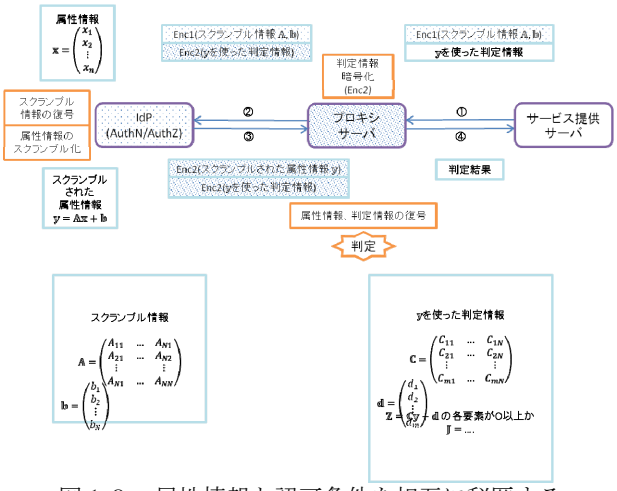



図１９　属性情報と認可条件を相互に秘匿するソフトウェア

本システムを使用するためには、大きく分けて３つの手順がある。

1.　フェデレーション内で、認可処理を行う際に使用する属性についての合意を行う。

SP と IdP が参加するフェデレーション内で、認可処理にて使用する SAML 属性名と、IdP 側で持つ複数の属性値との対応を合意し、IdP・Proxy・SP に設定を行う。

2.　認証連携 Proxy を経由し、ユーザ認証を行う。

ユーザがサービス提供サーバ(SP)にアクセスすると、SP は認証を要求する。ユーザは DS (Discovery Service)を経て ID 管理サーバ(IdP)を選択し、その認証を受ける。このとき、ユーザはプロキシサーバ(Proxy)、IdP の二度、DS による選択を経由する必要がある。ユーザが IdP による認証にパスすると、Proxy を経由し、SP に認証情報が伝達される。

この際、SP-Proxy 間、Proxy-IdP 間でそれぞれ、セッションが確立される。また、SP は Proxy から、どのサーバによって認証されているのかを認識する。

3. 属性による認可処理を行う。

アプリケーションから SP への要求に基づき、SP - Proxy, Proxy - IdP の間で、属性スクランブル情報・認可条件・スクランブル化された属性・認可結果を、暗号化の上で相互に受け渡し、認可処理が行われ、最終的に認可結果がアプリケーションに返却される。

SP → Proxy → IdP への問合せは、SAML2 の AttributeQuery リクエストとして送られ、回答は、AttributeQuery に対応する AttributeStatement の情報として返される。

なお、その際に使用する Binding は、SAML の AttributeQuery リクエストの制限に基づき、SOAP Binding となる。

本システムの認可処理フローは下記の通りである。

1. アプリケーションから SP へ、認可条件を提示

SP 上で動作しているアプリケーションは、sspmod_decideAttribute_DecideAttributeBuilder クラスに対して認可条件を提示し、認可結果を要求する。認可条件は、下記の３つの情報により構成される。

a) Ax + b ≧ 0 形式の行列 A, ベクトル b を用いた不等式

b) 各不等式の真偽により、最終的な１つの真偽を判定するための真理値表

c) 上の問い合わせに対応する、予め合意された認可処理にて使用する SAML 属性名

2. SP から Proxy 経由で IdP へ、認可要求を送付

SP は属性をスクランブルするための情報を生成するとともに、スクランブル化された属性のままで認可判断を行う認可条件を生成する。

また、送付する情報を暗号化するために、ユーザ認証時に得られ、SP が保持している IdP と Proxy の EntityID を元に、metadata より公開鍵を取得し、暗号化を行う。これにより、IdP, Proxy ごとに、必要最小限の情報を開示することが可能になる。

そして、IdP 宛てのスクランブル情報と Proxy 宛ての認可情報を１つの Attribute 要素にまとめ、アプリケーションから渡された SAML 属性名の問い合わせとして、AttributeQuery のリクエストを Proxy に送付する。

リクエストは、SAML 上では図 ２０のようになる。

```
<saml2p:AttributeQuery ～～>
  <saml2:Subject>認証されたユーザの情報
  </saml2:Subject>
  <saml2:Attribute  Name="(要求属性名)"
SP2IDP=" (スクランブル情報を暗号化したもの)"
 SP2PROXY="(認可情報を暗号化したもの)" />
</saml2p:AttributeQuery>
```

図 ２０ 認可要求の SAML 上での表現

3. Proxy でリクエストを中継

Proxy にて AttributeQuery リクエストを IdP に中継する。

この時に中継する先の IdP は、Subject 要素に含まれる NameID で表されるユーザが認証時に使用した IdP になる。一般的なプロキシサーバの動作として、NameID・送信者などは、IdP に合わせて調整される。

この時点では、認可に関する処理は行わない。

4. IdP にて、属性のスクランブルを実施

IdP は一般的な動作として、AttributeQuery にてリクエストされた Subject 要素が示すユーザの属性のうち、Attribute 要素で指定された SAML 属性に対応する属性を Proxy に返却する。

本システムでは、要求された属性のうち事前合意された SAML 属性名について下記の処理を行う。

はじめに、SP から渡された暗号化されたスクランブル情報を、IdP 自身の秘密鍵を用いて復号する。

次に、要求されたユーザの SAML 属性名に対応した、事前合意された複数の属性値を、合意された順に、LDAP 等から取得し、スクランブル情報に基づいて、スクランブル化された属性値を計算する。

そして、Proxy の EntityID を元に metadata より公開鍵を取得し暗号化を行うことで、スクランブル化された属性値を Proxy でのみ復号することが可能になる。

最後に、要求された１つの Attribute 要素の内容として、AttributeValue を 2 つ作成し、SP から送られた Proxy 宛ての認可情報と、IdP で生成したスクランブル化された属性情報を格納して、AttributeQuery の応答として、Proxy に返却する。

応答は、SAML 上では図 ２１のようになる。

```
<saml2p:Response>
  ～～
  <saml2:Assertion>
  ～～
    <saml2:AttributeStatement>
      <saml2:Attribute  NameFormat="  ～  "
FriendlyName="～"  Name="(属性名)">
        <saml2:AttributeValue> (認可情報を暗号化
したもの) </saml2:AttributeValue>
        <saml2:AttributeValue>(スクランブル化され
た属性情報を暗号化したもの)
</saml2:AttributeValue>
      </saml2:Attribute>
    </saml2:AttributeStatement>
  </saml2:Assertion>
</saml2p:Response>
```

図 ２１ 認可情報とスクランブル化属性を含んだ応答の SAML 上での表現

5. Proxy にて認可判断を行い、SP へ認可結果を送付

Proxy は、IdP から返却されたスクランブル化された属性情報と、SP から送られた認可条件の暗号化を、Proxy 自身の秘密鍵を用いて解読する。

スクランブル化属性値を認可条件に代入・計算することにより、1bit で表される真偽を認可結果として生成する。

そして、SP の EntityID を元に metadata より公開鍵を取得し暗号化を行うことで、認可結果の真偽値を SP でのみ復号することが可能になる。

AttributeValue 要素の値を、認可結果の真偽値として、SP に返却する。

応答は、SAML 上では図 ２２のようになる。

```
<saml2p:Response>
  ～～
  <saml2:Assertion>
  ～～
    <saml2:AttributeStatement>
<saml2:Attribute  NameFormat="  ～  "
FriendlyName="～"  Name="(属性名)">
        <saml2:AttributeValue>(認可結果を暗号化
したもの) </saml2:AttributeValue>
</saml2:Attribute>
    </saml2:AttributeStatement>
  </saml2:Assertion>
</saml2p:Response>
```

図２２ 認可結果応答の SAML 上での表現

6. SP からアプリケーションへ、認可結果の回答
SP にて、Proxy から返却された認証結果の真偽値を、SP 自身の秘密鍵を用いて解読する。この結果を、SP からアプリケーションに返却することで、アプリケーションは認可結果に基づき、制御を行う。

ここまでの認可フローの各段階で、情報の生成・加工を表にしたものが 表 7 である。 暗号化・復号化については、生成・加工時に暗号化され、加工・使用される際に復号される。

表 7 処理の流れに伴う、各種情報の生成と使用

| | スクランブル情報 | スクランブル化属性 | 認可条件 | 認可判定結果 |
|---|---|---|---|---|
| 1.アプリ | | | 生成 | |
| 2.SP | ランダム生成 | | 加工 | |
| 3.Proxy | ↓通過↓ | | ↓通過↓ | |
| 4.IdP | 生の属性値をスクランブル化 | | ↓通過↓ | |
| 5.Proxy | | 認可条件・スクランブル化パラメータにより判定 | | |
| 6.SP | | | | 受信 |
| 7.アプリ | | | | 使用 |

本システムにおける制限事項としては以下がある。
まず、SP と IdP は同じフェデレーション内であることが必要である。SP が IdP の公開鍵を得る際、IdP の Entity ID を元に、SAML の metadata に依存しており、結果としてフェデレーション内でのみ動作することが可能である。フェデレーションをまたいで利用するためには、認証時などに、IdP の公開鍵(証明書)を Proxy から SP に対して、送付するような仕組みを拡張して実装する必要がある。
また、 属性値は整数値であることが必要である。行列計算を行い、範囲条件で判定を行うために、対応できる属性値は数値である必要がある。また、浮動小数点演算には誤差が伴うことにより、スクランブルの結果、属性値に誤差が発生する可能性があるために、誤差が原理的に発生しないように、属性値を整数に限定している。
次に、スクランブル化した属性で判定を行う原理について述べる。
まず外部から与えられるパラメータは以下の通りである。
1. 使用する属性の合意
IdP が提供する属性を、列ベクトルとしてxとし、属性の数を m とする。
xの各要素について、実際の値と添え字のマッピングを事前に合意しておく。

$$\mathbb{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

例) m=3 のときの例
$x_1$ ： 生年月日(1899 年 12 月 31 日からの経過日数)
$x_2$ ： 数学の点数
$x_3$ ：TOEIC の点数

2. サービス提供サーバ(SP)にて、実際に知りたい条件を作成
2.1. 個々の属性についての条件を不等式として表現

Step1. $x_n$ に対する条件を不等式として表現し、その個数を n とする。

例)
式 1: $x_1 \geq$1989 年 1 月 1 日-1899 年 12 月 31 日の日数＝32509[日]
式 2: $x_1 \leq$1989 年 12 月 31 日-1899 年 12 月 31 日の日数=32873[日]
式 3: $x_2 \geq$60[点]
式 4: $x_3 \geq$800[点]

Step2. この条件を、$\mathbb{C}\mathbb{x} + \mathbb{d} \geq \mathbb{0}$ 形式の連立不等式として表す。
$\mathbb{C}$は n 行 m 列の行列、$\mathbb{d}$は n 行の列ベクトルとなる。
$\mathbb{C}$を下の判定行列、$\mathbb{d}$を下の判定ベクトルと呼ぶことにする。

例)

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbb{x} + \begin{pmatrix} -32509 \\ 32873 \\ -60 \\ -800 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

この場合、

$$\mathbb{C} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \mathbb{d} = \begin{pmatrix} -32509 \\ 32873 \\ -60 \\ -800 \end{pmatrix}$$

となる。

2.2. 個々の項目の判定結果を判断する真理値表を用意する。
n 個の条件それぞれが成立したとき 1、そうでない場合を 0 として、最終の 1bit の判定結果を得るための真理値表 T を作成する。

例 1) 式 1..4 とも成立したとき真(1)とする場合。
意味:生年月日が 1989 年 1 月 1 日～1989 年 12 月 31 日、数学 60 点以上 、TOEIC800 点以上である者

| 出力 | 入力 | | | |
|---|---|---|---|---|
| | 式 1 | 式 2 | 式 3 | 式 4 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

例 2) 式 1,2 が共に成立するか、式 3,4 が共に成立したときに真(1)とする場合
意味:生年月日が 1989 年 1 月 1 日～1989 年 12 月 31 日であるか、数学 60 点以上 かつ TOEIC800 点以上である者

| 出力 | 入力 | | | |
|---|---|---|---|---|
| | 式 1 | 式 2 | 式 3 | 式 4 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 |

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

次に、スクランブル情報を生成し、対応する判定式を得る処理について述べる。この処理は、サービス提供サーバで行われる。

1. IdP にてスクランブルを行うためのスクランブル情報を生成する。

$\mathbb{y} = \mathbb{A}\mathbb{x} + \mathbb{b}$ の式でスクランブルするために、m 行 m 列の行列 $\mathbb{A}$と、m 行の列のベクトル $\mathbb{b}$ をランダムに生成する。
行列$\mathbb{A}$は、逆行列$\mathbb{A}^{-1}$が存在する必要があるため、逆行列$\mathbb{A}^{-1}$を求めるとともに、行列式 det を生成する。

$$\mathbb{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{n2} & \cdots & a_{mm} \end{pmatrix}、\ \mathbb{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

ここで、行列式 det = 0 の場合には逆行列が存在しないため、行列$\mathbb{A}$を再度生成する。
また、行列式 det < 0 の場合に、スクランブル化された属性値を用いた判定の符号が逆転するので、この場合も行列$\mathbb{A}$を再度生成する。
ここで得られた行列$\mathbb{A}$をスクランブル行列、ベクトル$\mathbb{b}$をスクランブルベクトル、合わせてスクランブル情報と呼ぶことにする。

2. Proxy にて判定するための判定行列・判定ベクトルを生成する。

元の判定式、$\mathbb{C}\mathbb{x} + \mathbb{d} \geq \mathbb{0}$ と、スクランブル式 $\mathbb{y} = \mathbb{A}\mathbb{x} + \mathbb{b}$ から、属性をスクランブルしたままで判定できるような判定式$\mathbb{G}\mathbb{y} + \mathbb{h} \geq \mathbb{0}$を作成する。

Step1. $\mathbb{y} = \mathbb{A}\mathbb{x} + \mathbb{b}$ の逆変換を行う式、$\mathbb{x} = \mathbb{A}^{-1}(\mathbb{y} - \mathbb{b})$ を求める。

Step2. 条件式、$\mathbb{C}\mathbb{x} + \mathbb{d} \geq \mathbb{0}$ に、Step1 で求めた式を代入する。

$$\mathbb{C}(\mathbb{A}^{-1}(\mathbb{y} - \mathbb{b})) + \mathbb{d} \geq \mathbb{0}$$

これを展開すると下記のようになる。

$$\mathbb{C}\mathbb{A}^{-1}\mathbb{y} - \mathbb{C}\mathbb{A}^{-1}\mathbb{b} + \mathbb{d} \geq \mathbb{0}$$

逆行列$\mathbb{A}^{-1}$ の各要素は、一般に分母を行列式 det とした分数で表され、浮動小数点になるため、各要素をの係数を整数にするため、両辺に行列式 det を掛ける。
(行列式 det <0 の場合、符号判定して不等号の向きが反転するため、det>0 が前提である。)

$$\det \mathbb{C}\mathbb{A}^{-1}\mathbb{y} - \quad \det \mathbb{C}\mathbb{A}^{-1}\mathbb{b} + \det\mathbb{d} \geq \mathbb{0}$$

Step3. $\mathbb{y}$以外の値を求め、$\mathbb{G}\mathbb{y} + \mathbb{h} \geq \mathbb{0}$形式の式に変形する。

a) 行列$\mathbb{G} = \det \mathbb{C}\mathbb{A}^{-1}$の要素を求める
b) ベクトル
$\mathbb{h} = -\det \mathbb{C}\mathbb{A}^{-1}\mathbb{b} + \det \mathbb{d} = -\mathbb{G}\mathbb{b} + \det\mathbb{d}$
の要素を求める。

ここで得られた行列$\mathbb{G}$を判定行列、ベクトル$\mathbb{h}$を判定ベクトルと呼ぶ。これと、外部から与えられた真理値表を、合わせて認可情報と呼ぶことにする。
SP では、スクランブル情報を IdP に、認可情報を Proxy に送付する。
IdP で行われる属性値をスクランブルする処理は以下の通りである。

IdP にて、属性ベクトル$\mathbb{x}$を取得し、スクランブル情報と合わせて、スクランブル化属性ベクトル$\mathbb{y}$を計算する。

$$\mathbb{y} = \mathbb{A}\mathbb{x} + \mathbb{b}$$

ここで得たスクランブル化属性ベクトル$\mathbb{y}$を Proxy に返却する。
次いで、Proxy で行われる、スクランブル化属性と認可情報により認可判定を行う処理について述べる。Proxy にて、スクランブル化属性ベクトル$\mathbb{y}$と、認可情報に含まれる判定行列$\mathbb{G}$・判定ベクトル$\mathbb{h}$を元に下記の計算を行う。

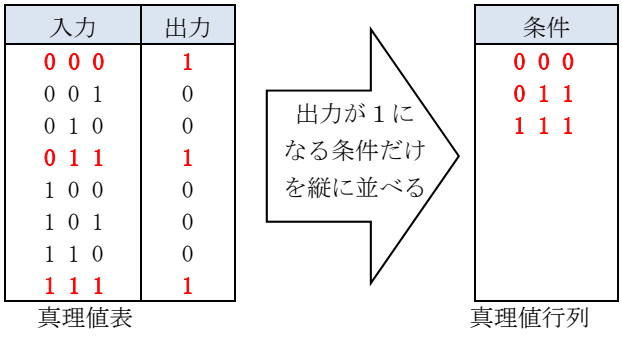
$$\mathbb{G}\mathbb{y} + \mathbb{h} \geq \mathbb{0}$$

ここでは、不等号の評価により、評価結果の真偽値が n 個生成される。これを、認可情報に含まれる真理値表の入力に当てはめることで、認可結果の真偽値を得る。得られた認可結果を、SP に通知することにより、SP は最終的な認可結果を知る。
n 個の式の判定結果の全ての組合せは $2^n$ 個存在するため、真理値表は
表 8 のような形になる。

表 8 一般的な真理値表

| 入力 | 出力 |
|---|---|
| 0 … 0 0 | z(1) |
| 0 … 0 1 | z(2) |
| 0 … 1 0 | z(3) |
| : | : |
| 1 … 1 1 | z($2^n$) |

出力は 真・偽のいずれかであり、通信するデータ量を削減するために、この真理値表のうち、認可結果が真となる入力を行ベクトルの形で表し、これを列方向に積み重ねたものを、便宜上、真理値行列と呼ぶことにする（図２３）



真理値表　　真理値行列

なお、本課題の成果のうち 1-3-1 については、情報処理学会 DICOMO2013 シンポジウムに「属性提供サーバに対してサービス提供サーバを秘匿する匿名化プロキシ」という題目で投稿し、採択された（2013 年 7 月に北海道にて開催）。また、1-4-1～1-4-3 の成果と併せて IEEE の COMPSAC 国際会議の MidArch ワークショップに “Privacy Enhancing Proxies in Attribute Release:Two Approaches”という題目で投稿し、採択された（7 月に京都にて開催）。

## 2.4 プロキシに対するプライバシー保護に関する研究開発（課題１－４）

全体構想の中で新たに導入する認証連携プロキシに対して、ユーザのプライバシーを開示しないための仕組みを開発する。

以下の 1-4-1、1-4-2 に示す 2 テーマにおいて実現した機構を 1-4-3 において 1 つの Privacy Enhancing Proxy として統合し、公開する。また、1-4-4 は京都大学と東京大学との間の調整に基づき、1-3-3 の課題を引き継いだものである。

### 2.4.1 認証連携 Proxy サーバに対するユーザのプライバシー非開示機構（１－４－１）

ユーザ認証を経てそのユーザに関する属性情報を属性提供サーバがサービス提供サーバに提供する際、ユーザがどのサービス提供サーバに対してアクセスしたかというプライバシー情報を保護するためには、プロキシの導入が必要となる。しかし、単純にプロキシを導入すると、それぞれのユーザに関する属性情報の内容をプロキシが知ることになってしまい、ユーザのプライバシーがプロキシに知られてしまうことになる。このような問題を回避するためには、属性アサーションの内容を暗号化することが必要である。従来の実装でも暗号化はなされているが、通信経路上での情報漏洩を回避するためのものであり、送信先がサービス提供サーバかプロキシであるかに関わらず送信先において復号させることを前提として通信が行われる。ここでは、属性の送信先が信頼のおけるプロキシであることを認識し、プロキシには必要な部分を暗号化した属性を送るための暗号化機構の設計と実装を行う。

### 2.4.2 暗号化された属性アサーションの内容を SP のみが復号できる機構（１－４－２）

暗号化された情報は、送出先のサービス提供サーバのみが復号できなければならない。復号のための鍵管理機構を中心とし、さらに属性提供サーバに対してどのサービス提供サーバにアクセスしようとしているかというプライバシー情報の保護も考慮した技術の開発を行う。

### 2.4.3 課題３　Privacy Enhancing Proxy の設計と実装（１－４－３）

課題 1-4-1 と 1-4-2 で提供される機能に加えて、関係するサーバと通信を行うプロトコルとディレクトリの設計と実装を行う。これによって両者の機能が利用可能になる。さらに全体計画中の属性サーバとのデータ交換が可能になることで、Privacy Enhancing Proxy が実現される。

これら 3 つの課題のユースケースのひとつとして、学割の提供モデルを図２４に示す。このモデルは、従来からある学割を以下のシナリオにまとめて、「学生証提示モデル」として提示し、そのオンライン版を作るものである。

1．大学は学生に学生証を発行する。学生証はその保持者が学生であることを保証する。
2．学生は、ソーシャルなサービスを学割で購入することができる。その時に学生は大学が学生の購入履歴等のプライバシー情報を取得しないことで、安心して購入ができる。
3．サービス提供側は、学生証の提示が強い意味での「属性」保証であることを認めて、学割を提供する。

オンライン版では、「学生」の属性は上記３．の保証のためには大学の IdP から送出される必要がある。これは２．と本質的に矛盾するが、われわれの Privacy Enhancing Proxy を使うことで、上記のシナリオが実現可能になる。

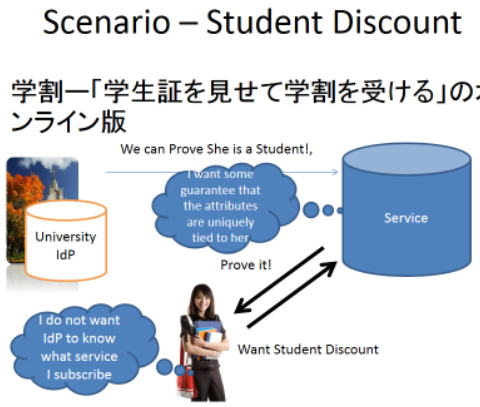



図２４　「学生証を見せて学割を受ける」シナリオのオンライン版

一方、サービスを提供する側からは、Privacy Enhancing Proxy によってプライバシーを保護することには同意しつつも、二重契約の防止等のビジネス上の要件を満たさなければ現実的にはサービス提供ができない。Proxy の運用にあたってこの問題を解決することが求められる。そのために、以下の課題を追加で設定した。なお、この課題は計画時には京都大学が分担したものであるが、以上の理由で機能的にオーバーラップするところが多く、東京大学でも、Privacy Enhancing Proxy に関係するシナリオで研究開発を行うことにした。

### 2.4.4 課題４　利用者が SAML で提供される仮名性を担保しつつ、サービス提供者がユーザ同一性を確認するための機構

現在、広く用いられている ID 連携プラットフォームでは、個々のユーザの認証を行った後、そのユーザに関する属性情報を ID 管理側からサービス提供側に提示し、それに基づいて認可判断を行う形態をとる。属性情報には個人情報が含まれるので、その提示には本人同意が前提となるのは当然として、その開示範囲が必要最小限となるように配慮すべきである。また、誰がどのようなサービスにアクセスしたかという情報もプライバシー情報として保護されるべきものであり、開示範囲を最小化する必要がある。本研究開発では、これらの属性情報を適切に暗号化、仮名化(スードニマイズ,pseudonymize)した上で流通させることにより、たとえば、ユーザがどのサービスにアクセスしたかが ID 管理側に分からない、あるいは、どのユーザがアクセスしてきたかがサービス提供側に分からない、とい

ったプライバシー保護機能を、認証の信頼性を落とすことなく実現することを目的とする。このユースケースを図２５に示す。

具体的には、仮名化されたユーザ ID を利用しながら、二重登録の防止のために、IdP から送出された属性の送出回数を、仮名化を損なわない形でカウントできる機能を実現する。これにより、複数のアカウントから属性提供サーバに同一 ID の属性送出がなされた時に、カウント機能を利用してチェックができる。この機能を実現するサーバ（カウントサーバ）を実現することを目的とする。

この課題を解決することで、サービス提供者にとっては、たとえば二重契約の防止などが可能になり、より広い範囲のサービスが安心して提供できるようになる。図２５のシナリオは、前述の（シナリオ２）を学割に適用したものである。

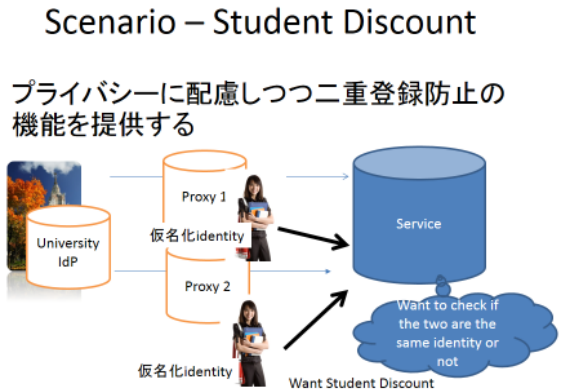



図２５　プライバシーに配慮しつつ、ビジネス側の要件を満たすシナリオ

## 2.4.5 成果及び結果

以下に示すソフトウェアを開発し、目標とする機能が実現されていることを確認した。

### 2.4.5.1 Privacy Enhancing Proxy

1-3-1 で採用された方式とは異なる、ユーザエージェントを用いたアーキテクチャを採用し実装した。モバイルデバイスの普及により、ユーザはそれぞれモバイルデバイスを所有することが一般的になってきていることから、モバイルデバイスの持つブラウザへのプラグイン等の形態でユーザエージェントを実現し、鍵管理その他の機能を受け持たせることとした（図２６）。

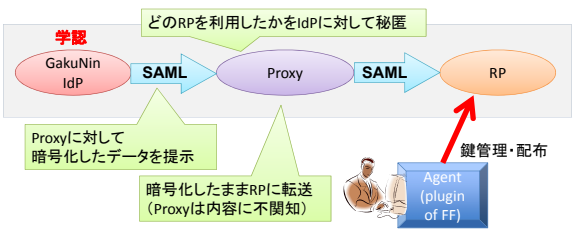



図２６

### 2.4.5.2 利用者が SAML で提供される仮名性を担保しつつ、サービス提供者がユーザ同一性を確認するための機構

あるユーザが、課題 1-4-3 等で実現されるようなプロキシを複数利用して同一サービスにアクセスすると、サービス提供者側では、同一ユーザであるかどうかが判別できなくなる。これでは、ユーザごとに割引回数の上限を設けたいような場合に対応できない。そこで、ユーザの仮名性を保ったまま、ユーザ毎のサービス利用回数をカウントできるようなカウントサーバを実現した。カウントサーバは SP のリクエストに応じて、あらかじめ ID を暗号化しておいたカウント用の ID（CID）の管理を行う。さらに、IdP から SP を隠ぺいする機能を持つ Proxy を実装した。IdP には、自身の管理するユーザ情報から暗号化された CID を生成する機能を付加した（図２７）。

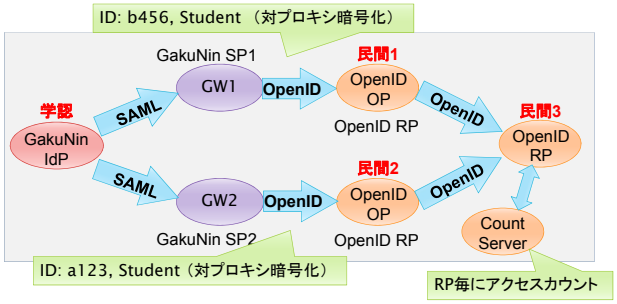



図２７

なお、前述のとおり、本課題 1-4-1～1-4-3 については、前項の課題 1-3-1 の成果と併せて IEEE の COMPSAC 国際会議の MidArch ワークショップに“Privacy Enhancing Proxies in Attribute Release:Two Approaches”という題目で投稿し採択された（７月に京都にて開催）。

## 2.5 属性提供サーバ個人設定に関する研究開発（課題２－１）

ユーザが複数の属性提供サーバを使い分けるような環境が提供されるようになると、ユーザが自分の利用したい属性提供サーバを簡単に選択し、情報提供のコントローラー（PDP, Policy Decision Point）に提示する仕組みが必要となってくる。

しかし、OpenID Connect には、OpenID プロバイダ（ID プロバイダ）とは異なるサーバとして存在する属性提供サーバからエンドユーザの属性を提供する仕組みとして、分散クレームと集約クレームの 2 つのモデルが存在するが、ID プロバイダと属性提供サーバとの間のやりとりの具体的な仕組みは、現在標準化されたプロトコルが存在せず、エンドユーザが ID プロバイダから属性提供サーバを任意に使い分ける方法が一般的には存在しない。

そのため、ユーザにとってわかりやすい文字列での属性情報提供サーバの検索と候補リストの作成の仕組み、候補リストからユーザが選択した属性情報提供サーバからのメタデータの取得の仕組み、候補リストで選択した属性情報提供サーバと PDP の ID の紐付けとユーザ許諾の仕組みについて、開発を行う。

なお、当該課題については、Kantara Initiative における UMA WG において、若干の検討が行われているが、UMA モデルは属性情報提供サーバと PDP の間のやり取りがプッシュ/プルのいずれも想定しているのに対し、当研究開発ではプルモデルに専念することで、より軽量で実

装性の高いものとする。

### 2.5.1 成果及び結果

ID プロバイダと属性提供サーバとの間のやりとりの具体的な仕組みは、現在標準化されたプロトコルが存在せず、エンドユーザが ID プロバイダから属性提供サーバを任意に使い分ける方法が一般的には存在しなかった。そこで、民間における具体的なユースケースとして病院間での診断情報のやりとりを想定し、このユースケースにおいて利用可能な仕組みを実装し検証した。

想定したユースケースは次のとおりである。病院 A は WEB サイト上で患者に対して過去の診断情報を要求し、その診断情報をもとに診察予約等のサービスを提供する。ここで、病院 A は患者に直接診断情報を入力させるのではなく、他の病院に存在する診断情報を、OpenID Connect をもとに拡張した仕組みを用いて受け取る仕組みを想定する（図２８）。

1. 患者は病院 A で診療をうけるため、病院 A の WEB サイト（サービス提供サーバ）にアクセスし、病院 A から過去の診断情報を要求される。
2. 病院 A（サービス提供サーバ）は患者を ID プロバイダに誘導する。
3. ID プロバイダは既知の属性提供サーバを列挙し、患者は自分の診断情報が存在している病院 B（属性提供サーバ）を選択する。わかりやすい文字列で表現された属性提供サーバを選択することになる。
4. ID プロバイダは、病院 B（属性提供サーバ）に対してどのような種類の属性を返す能力があるかを問い合わせる。
5. 病院 B（属性提供サーバ）は、患者の許諾のもとに病院 B 上（属性提供サーバ）の患者のアカウントと ID プロバイダ上のユーザのアカウントを紐付け、ID プロバイダの発行する許可に含まれるユーザ ID から病院 B 上のアカウントを特定できるようにする。
6. ID プロバイダは患者に対し、病院 B から提供されうる診断情報（属性）の種類を提示した上で、病院 A（サービス提供サーバ）に提供しても良いか許諾を取る。
7. ID プロバイダは病院 A（サービス提要サーバ）に対し、診断情報取得の許可を発行する。
8. 病院 A（サービス提要サーバ）は発行された許可を用いて、病院 B から診断情報を取得する。

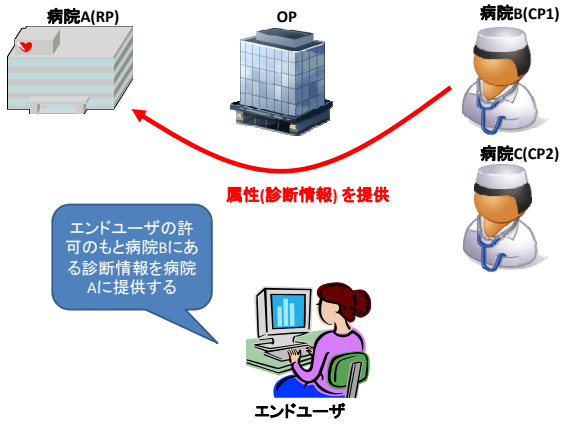



図２８

このユースケースに基づいて開発の概要は次のとおりである。属性提供サーバを URI ではなく固有名詞及びカテゴリで分類したうえで一覧する機能として、ユーザにとってわかりやすい文字列での属性情報提供サーバの検索と候補リストの作成の仕組みを実装させた。

一連のユースケースの実証の中で、その候補リストからユーザが選択した属性情報提供サーバからのメタデータの取得の仕組み、候補リストで選択した属性情報提供サーバと PDP のアカウントの紐付けとユーザ許諾の仕組みを検証した。

また、アカウントの紐付けは WEB アプリケーション間で自動的に行われるものであり背後で実際に紐付いている具体的な情報はエンドユーザには直接見せないものであるが、アカウントの紐付けが必須となる一連の流れで属性取得が可能であることから、アカウントの紐付けの仕組みが実現できたことを確認した。

これにより、研究開発 2-1 において、エンドユーザが利用したい属性提供サーバが容易に選択でき、ID プロバイダと属性提供サーバが連携して属性を提供できるようにする仕組みを、新たにプロトコルを開発することで達成できた。

ユースケースをもとに検討した属性提供サーバ（CP）、サービス提供サーバ（RP）、ID プロバイダ（OP）に必要な機能を表 9 に示す。

表 9　研究開発 2-1 で実装した機能

| 実装箇所 | 機能 No. | 機能 |
|---|---|---|
| CP | 機能 1 | OpenID Connect の RP のとして振る舞うことで OP とエンドユーザの識別子（PPID）を共有し、それを自らの管理するエンドユーザの識別子と紐づける機能（アカウントリンク機能） |
| | 機能 2 | OP が発行したアクセストークンを用いた要求にこたえて属性を応答する機能 |
| | 機能 3 | 提供可能な手段および提供可能な属性情報を示す meta data の提供機能 |
| OP | 機能 1 | アカウントを管理し、エンドユーザの認証を行う機能 |
| | 機能 2 | CP の一覧を有し、エンドユーザに属性提供元を選択させる機能 |
| | 機能 3 | エンドユーザが選択した CP に対し、次節で記載するアカウントリンク機能の開始を要求する機能 |
| | 機能 4 | RP に提供される属性を、エンドユーザが選択可能とする機能 |
| RP | 機能 1 | OpenID Connect の OP と連携し、属性の提供を受ける機能 |
| | 機能 2 | エンドユーザが選択した OpenID Connect の OP と接続する機能 |

表に示すこれらの機能を PHP で備えた WEB アプリケーションを属性提供サーバ（CP）、サービス提供サーバ（RP）、ID プロバイダ（OP）のそれぞれにセットアップした。図２９に研究開発 2-1 の実証環境の概要図を示す。いずれも Amazon EC2 上の Linux サーバを使用した。また、データベースとして MySQL を使用した。

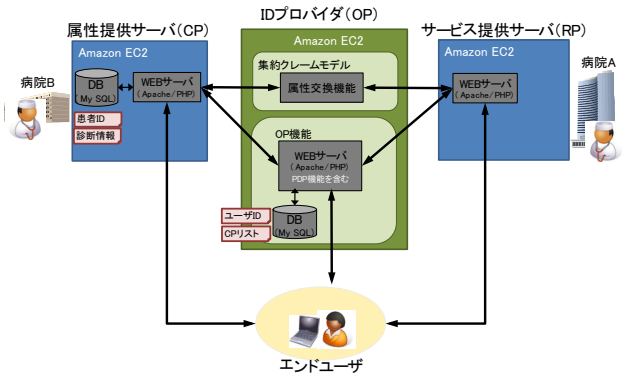



図２９　研究開発 2-1 の実証環境概要図

あらかじめ ID プロバイダに登録されている属性提供サーバの名称、プロパティ（業態による分類）、ホスト名等を元に、エンドユーザにとってわかりやすい属性提供サーバの選択方法として以下の機能を ID プロバイダ上に実装した。

- 文字列入力での検索機能による候補リストの提供
- 属性提供サーバのプロパティによる候補リストの提供
- 過去に利用した属性提供サーバの候補リストの提供

また、候補リスト以外にも

- 属性提供サーバのホスト名の入力による直接選択

を実装した。

### 2.5.2 GÉANT のテストハーネスによる試験

研究開発 2-1 では、GÉANT のテストハーネスでの検証試験を行った。

GÉANT のテストハーネスは、多くの国のプロジェクトで使用されている実績があり、本研究開発のプロトコルの検証環境を GÉANT のテストハーネス上に展開することは、本研究開発のプロトコルの改定すべき点や運用上注意すべき点等がより見つけられるだけでなく、今後の本研究開発の拡張機能の国際標準化につながることが期待される。

今回は、GÉANT のテストハーネスの作成者と本研究開発の目的や作成したプロトコル等の情報を共有し、本研究開発が既存のプロトコルに影響を与えずに追加機能が実装できているかについて検証を行った。試験対象は、OP-RP 間と OP-エンドユーザ間のプロトコルとした。試験結果は下記の通りである。

SUCCESS（+）：　49 件／全 74 件中　…成功
ERROR（！）：　1 件／全 74 件中…軽度なエラー
CRITICAL（X）：　24 件／全 74 件中　…未実装等の重度なエラー

24 件あった CRITICAL（X）のエラーは下記の要因により、評価対象外となる。

- 本件開発の目的に必要とされていない機能を試験された
- 本件開発で対象外とした異常系の処理が試験された

1 件あった ERROR（！）は、OpenID Connect の仕様上実装すべきであるが必須ではない項目がエラーとして報告されているものである。この ERROR（！）は具体的には、

52! (mj-45)Registration with policy_url and logo_url - ERROR

で、OpenID Connect Dynamic Client Registration で規定されている、RP のロゴおよびポリシーとしてそれぞれ指定された URI を OP のページに含む動作が実装されていないことが要因であり、本件開発においては評価対象外となる。

この結果、既存プロトコルに影響を与えず本研究開発の追加機能を実装できていることを確認した。

テストハーネスの作成者からは追加依頼への了承を得ており、今後は、GÉANT のテストハーネス上に、本研究開発で研究開発した追加プロトコルの検証環境を展開する予定である。

## 2.6 属性提供サーバに対するプライバシー保護にする研究開発（課題２－２）

本研究開発は、ユーザが認証を経てサービス提供サーバ(SP、サービスプロバイダ)にアクセスする際に、どのサービス提供サーバにアクセスしたかというプライバシー情報を ID 管理側(IdP、ID プロバイダ)あるいは属性提供サーバ(AP、属性プロバイダ)にわからないようにしようとするものである。

本研究開発は１－３）で行われる内容と課題は同等であるが、その解決手段としてのインターフェース実装を OpenID Connect にて行うものするものである。

IdP によるユーザ認証に基づいて属性提供サーバ（あるいは IdP）が発行する属性アサーションをサービス提供サーバに伝達する際に、具体的なサービス提供サーバの存在を属性提供サーバ（あるいは IdP）に見せないようにすることで、プライバシー保護を実現する。ここでのサービス提供サーバは任意のものを許すわけではなく、事前に信頼関係を構築しているサービス提供サーバのうちの何れかであることが前提であることから、認証連携プロキシ等を導入した新たなアーキテクチャの開発が要求される。

認証連携プロキシを導入することにより、属性情報を用いてサービス提供サーバが認可判断を行う場合に、属性提供サーバは個々の属性情報を秘匿しておきたい、またサービス提供サーバは認可の条件を秘匿しておきたいという要求に対応することも比較的容易になる。

さらにこのようなプライバシー保護を実現した上で、属性提供プロバイダにおける問題の検討として、属性提供サーバからは仮名識別子を提供しつつ、同一のサービスに対して重複した割引サービスを受けようとした場合に、それが検出可能な仕組みについても研究開発する。

### 2.6.1 成果及び結果

課題 2-1 に加えて、属性提供サーバがサービス提供サーバに属性を提供する際に属性のリクエストを中継するプロキシ機能を導入することで、エンドユーザのプライバシー情報を属性提供サーバから秘匿する技術開発を実施した。本課題の内容は、次の 3 つから構成される。

2-2-1) 属性提供プロセスにおけるプロキシ機能の導入
2-2-2) プロキシ上の条件判断による結果の提供
2-2-3) 利用回数制限を設けたいサービスに対する利用回数カウンタの提供

以下、それぞれについて説明する。

#### 2.6.1.1 属性提供プロセスにおけるプロキシ機能の導入

ユースケースとしては、研究開発 2-1 のユースケースで、エンドユーザが診断を受けようとする病院 A が病院 B から診断情報を取得する際に、病院 B の診断情報の提供先

が病院 A であることが病院 B に知れてしまうとプライバシー上の問題が発生するという場合である。この問題を避けるため、プロキシを経由して情報提供先を秘匿する仕組みを考えた（図３０）。

#### 2.6.1.1.1 プロキシの実装方針

OpenID Connect では、サービス提供サーバが属性提供サーバから属性を取得する方法は分散クレームと集約クレームの 2 種類が存在する。

表１０ 属性取得モデル

| 属性取得モデル | 属性情報の所在 | 属性提供の動作 |
|---|---|---|
| 分散クレーム | 属性提供サーバ（CP） | OP が属性の取得方法（取得先 CP 情報）を RP に渡し、RP が CP から直接取得する。 |
| 集約クレーム（研究開発 2-1） | 属性提供サーバ（CP） | OP は、一旦、CP から属性を取得し、RP に渡す。 |

属性情報サーバにどのサービス提供サーバに属性情報を引き渡すかを秘匿する方式は、技術的には分散クレームにて属性提供サーバとサービス提供サーバの間に中継プロキシを挿入する方法と集約クレームを用いて ID プロバイダを中継プロキシとして利用する方法が考えられる。今回は実装を単純化できる利点を重視し、集約クレームを採用した。

実装したシステムでは、プロトコル仕様上、属性提供サーバに渡される情報は Account Linking に関わる情報とアクセストークンのみであり、その中に含まれる情報のうちサービス提供サーバに関連する情報は issued_for claim のみである。集約クレームの場合、issued_for claim が存在する場合には ID プロバイダを指すこととなっているので、属性提要サーバにサービス提供サーバの情報が秘匿される。また、間接的な情報として、属性を取得したパーティの IP アドレスをもとにサービス提供サーバを知ることが考えられるが、集約クレームにおいては属性提供サーバと通信を行うのは ID プロバイダ及び Account Linking 時の user-agent のみであり、実際に本課題の実証においてアクセスログにはサービス提供サーバの IP アドレスは存在しない。

以上より、OpenID Connect の集約フレームを用いて ID プロバイダをプロキシ機能として利用することにより、属性提供サーバに対してサービス提供サーバを秘匿できることが確認された。

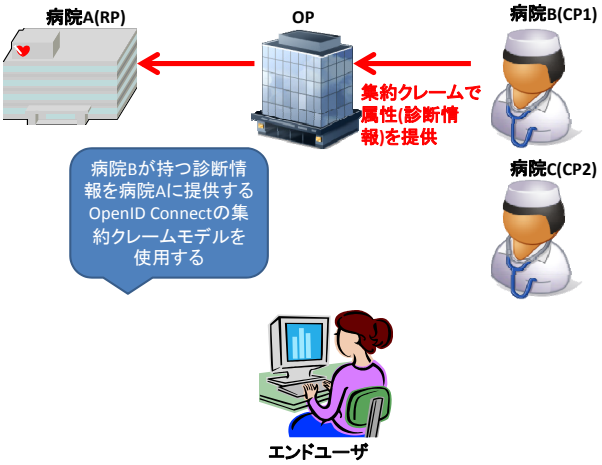



図３０

#### 2.6.1.2 プロキシ上の条件判断による結果の提供

研究開発 2-2-1 を達成した状態で、サービス提供サーバが何らかの条件式に基づいてサービス提供の可否を判断したい場合に、サービス提供サーバが条件式を属性提供サーバから秘匿でき、かつ属性提供サーバが属性の値そのものをサービス提供サーバから秘匿できるように、プロキシ上で属性に対して条件式を適用する仕組みの技術開発を実施した。

ユースケースとしては、年齢に制限のある予防接種**サービスの提供**が考えられる。病院 A は予防接種サービスを実施するにあたって、患者が 20 歳以上であることを要求するが、その具体的な判断基準を健康保険組合 B には秘匿したい。また、健康保険組合 B は患者の年齢にかかわる情報を提供可能としているが、具体的な生年月日を他の病院等に提供しないポリシーが設定されている。

#### 2.6.1.2.1 プロキシにおける属性条件式評価の実装方針

研究開発 2-2-2 は、研究開発 2-2-1 の目的を達成した上で、さらにプロキシ上で属性にサービス提供可否判定の条件を適用し、サービス提供サーバが属性の値そのものを知ることなくサービス提供判定条件の適用結果のみを知り得る、且つサービス提供判定条件を属性提供サーバに秘匿する仕組みを実現することになる。そこで、研究開発課題 2-2-1 でプロキシとみなした ID プロバイダ上で、取得した属性にサービス提供判定条件式を適用し、その結果を通知する機能の技術開発を実施した。具体的には、次の 3 つの機能を実現した（図３１）。

- 条件式を Request Object に含めた Authorization Request を発行する機能
- 条件式を含んだ Request Object にかかわる UserInfo Request を受けた際に、CP には条件式への入力となる属性をリクエストし、その属性を条件式に適用した結果のみを RP に提供する機能
- アクセストークン中の claims_indirect 要素を判定し、サービス提供サーバに直接提供したくない属性の取得が要求された時に属性提示を拒否する機能

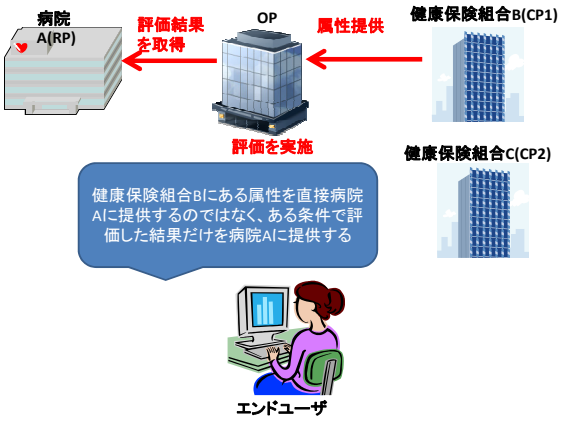



図３１

研究開発 2-2-2 の実証環境は、研究開発 2-1 と同等であるが、サービス提供サーバ（RP）、ID プロバイダ（OP）、属性提供サーバ（CP）のそれぞれに表１１に示す機能の追加を行った。

表１１ 研究開発 2-2-2 で新たに実装した機能

| 実装箇所 | 機能 No. | 機能 |
|---|---|---|
| RP | 機能 1 | 条件式を Request Object に含めた Authorization Request を発行する機能 |
| OP | 機能 1 | 条件式を含んだ Request Object にかかわる UserInfo Request を受けた際に、CP には条件式への入力となる属性をリクエストし、その属性を条件式に適用した結果のみを RP に提供する機能 |
| CP | 機能 1 | アクセストークン中の claims_indirect メンバを判定し、サービス提供サーバに直接提供したくない属性の取得が要求された時に属性提示を拒否する機能 |

1）プロトコル

● アクセストークン

研究開発 2-1 のアクセストークンのフォーマットをもとに、以下のフィールドを追加した。

表１２ アクセストークンの追加フィールド

| Member | Type | 必須/任意 | 説明 |
|---|---|---|---|
| claims_indirect | 文字列の配列 | 必須 | claims フィールドで指定された claim 名のうち、OP の中で評価式によって評価され、RP に直接渡されない claim 名のリスト。 |

ID プロバイダは、直接または間接に要求されているすべての属性について属性名を claims フィールドにセットし、さらにその中でも条件式にのみ出現しサービス提供サーバに属性そのものが直接渡されない属性名を claims_indirect フィールドにセットするようにした。なお、条件式に用いられ、かつ属性そのものも要求されている属性については属性名を claims_indirect フィールドにセットしない。

また、claims フィールドに含まれており、かつ claims_indirect フィールドには含まれていない属性名は属性の値が直接サービス提供サーバに渡されるため、それを望まない場合、属性提供サーバは属性の提供を拒否できるようにした。

● Authorization Request

Authorization Request の手順において、以下の要件を設定した。

1. サービス提供サーバは、属性に対する条件式を要求していることを表すため、OpenID Request Object を用いなければならない。
2. サービス提供サーバは、属性そのものではなく条件式への適合の有無のみを知りたい場合、要求する claim name として条件式を送出しなければならない。
3. ID プロバイダはサービス提供サーバから受け取った条件式を解釈し、条件式の評価に必要な属性を抽出できなければならない。

● UserInfo Response

UserInfo Response の手順において、以下の要件を設定した。

- ID プロバイダは、Authorization Request において条件式で指定された属性に関して、条件式の適用をしたうえで true または false で返さなければならない。

● Authorization Request - OpenID Request Object

条件式は、この研究開発では以下のような文法で記述することとした。

- 属性名が"condition("で始まり")"で終わるものを条件式として判定
- 条件式は数値または文字列の順序を判定するものとし、属性値のほかに定数、演算子、括弧を記述も可能
- 条件式の中に"claim("で始まり")"で終わる表記は、括弧内の文字列を属性名として判定し、属性の値に置換え
- 一つの条件式の中に複数の属性も可
- 条件式に使用できる文字種は、OpenID Connect で属性名に使用できる文字種と同一
  - ✧ JSON 形式で扱われるので、JSON の制限（RFC4627)に準じる
  - ✧ OpenID Connect では、'#'をロケール指定の表記で使用する('OpenID Connect Messages 1.0 - draft 14'の'2.1.1.1.3. "claims" member in "userinfo" and "id_token" members'参照)ので、評価式では使用しない
- 条件式は ID プロバイダが評価し、その結果を true または false で返す

なお、実際の実装では、以下の 2 つの条件式のみを受け付けるよう実装した。

- condition(claim(birthdate)<1993-01-01)

- condition(claim(gender)==female)

実現されたシステムにおいて、サービス提供サーバから条件式を ID プロバイダに渡し、ID プロバイダが属性提供サーバから属性を取得して条件式の適用結果を返していることを、属性提供サーバ内の該当する属性の値を変化させて条件式の適用結果が変化することで確認した。
また、ID プロバイダが属性の取得と条件式の適用を行い、サービス提供サーバに対して属性が直接提供されていないこと、属性提供サーバに対して条件式が提供されていないことは、サービス提供サーバの画面に表示される取得した属性一覧に該当する属性がないこと、属性提供サーバの受け取るアクセストークンに条件式が含まれていないことから確認した。以上より、プロキシ上の条件判断による結果の提供を達成した。

### 2.6.1.3 利用回数制限を設けたいサービスに対する利用回数カウンタの提供

研究開発 2-2-1 を達成した状態で、サービス提供サーバが同一のエンドユーザに対するサービスの提供回数制限を設けたいという要求に対し、ID プロバイダがグローバルユニークな識別子情報を出すことなく提供回数が検出可能となる仕組みの技術開発を実施した。
ユースケースとしては、病院 A がエンドユーザに回数制限のある予防接種を実施するときに、病院 A が回数上限に達しているかを問合せることが考えられる。このとき、エンドユーザはどの病院を利用したかを健康保険組合 B に知られずに、かつ健康保険組合 B はエンドユーザの予防接種回数を把握できる。

### 回数カウンタの実装方針

研究開発 2-2-3 は、研究開発 2-2-1 の目的を達成した上で、サービス提供サーバが同一のエンドユーザに対するサービスの提供回数制限を設けたい場合に、利用回数の検出可能な仕組みを目標とする。新たなアクターとして導入する回数カウンタには、サービス提供サーバが ID プロバイダを経由してカウンタ値を取得、設定できる機能を実現する。回数カウンタは属性提供サーバと同一のパーティ内に実装し、内部でデータベース等を共有する。さらに回数カウンタへのアクセス手段を提供するため、属性提供サーバ上で回数カウンタの URI 情報を提供する well-known URI と、ID プロバイダ上でサービス提供サーバからのアクセス手段情報を提供する well-known URI を準備する（図３２）。

具体的には、次の 3 つの機能を実現した

- OP に対してアクセストークンを用いた回数カウンタの操作をリクエストする機能
- アクセストークンに含まれる属性提供元の CP に対して、回数カウンタの操作リクエストを中継する機能
- 回数カウンタの機能

図３３に回数カウンタ部分のシーケンス図を示す。
実現されたシステムにおいて、サービス提供サーバが、アクセストークンを使用して回数カウンタの操作を行うことが可能であり、その値を取得可能であることを検証した。さらにエンドユーザが異なるサービス提供サーバ、ID プロバイダを使用したとしても、属性提供サーバを変更しないかぎり連続してカウントが行われることを確認した。以上より、利用回数制限があるサービスに対する利用回数カウンタの提供を達成した。

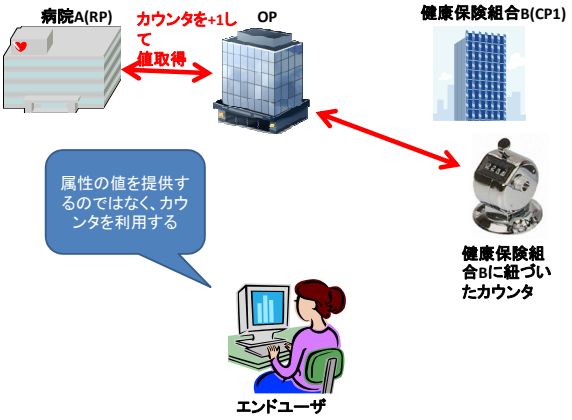


図３２

## 2.7 プロキシに対するプライバシー保護に関する研究開発（課題２－３）

本研究開発は、新たに導入する認証連携プロキシに対して、ユーザのプライバシーを開示しないための仕組みを開発するものである。
本研究開発は１－４）で行われる内容と課題は同等であるが、その解決手段としてのインタフェース実装を OpenID Connect にて行うものするものである。

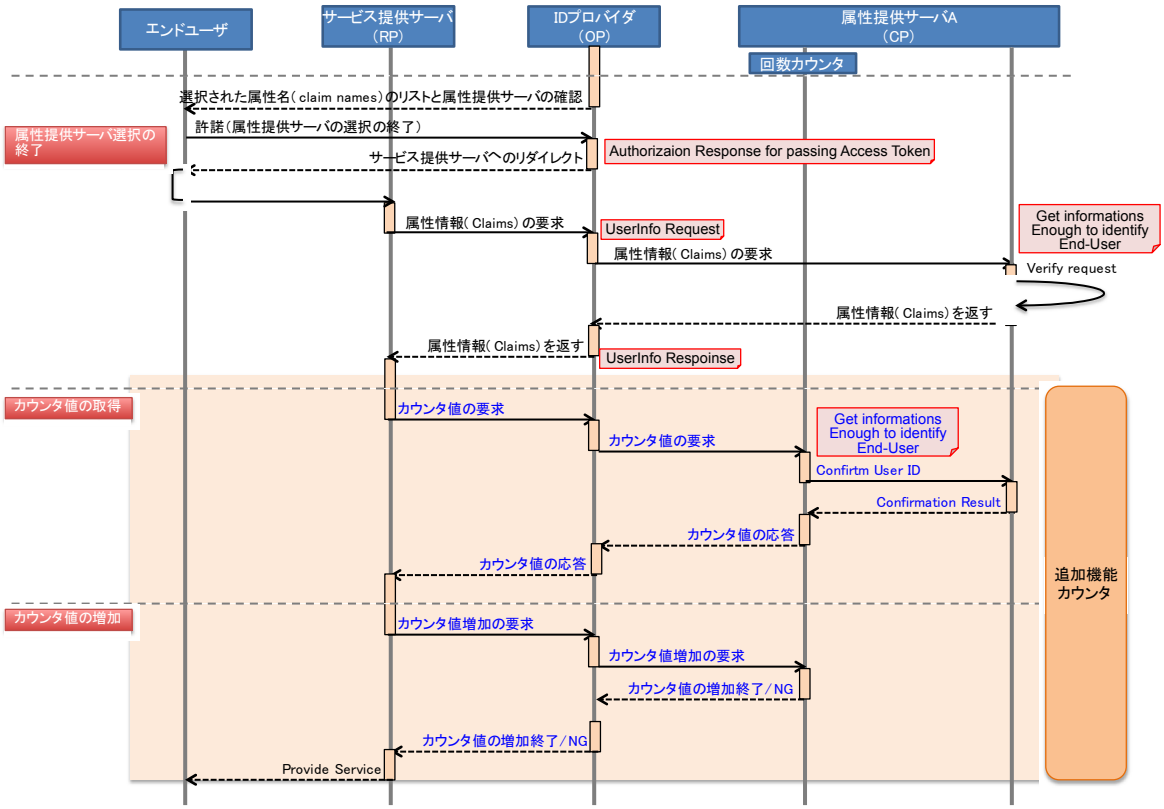



図３３　回数カウンタ部分のシーケンス図

ユーザ認証を経てそのユーザに関する属性情報を属性提供サーバがサービス提供サーバに提供する際、ユーザがどのサービス提供サーバに対してアクセスしたかというプライバシー情報を保護するためには、プロキシの導入が必要となる。しかし、単純にプロキシを導入すると、それぞれのユーザに関する属性情報の内容をプロキシが知ることになってしまい、ユーザのプライバシーがプロキシに知られてしまうことになる。このような問題を回避するためには、属性アサーションの内容を暗号化することが必要である。従来の実装でも暗号化はなされているが、通信経路上での情報漏洩を回避するためのものであり、送信先がサービス提供サーバかプロキシであるかに関わらず送信先において復号させることを前提として通信が行われる。本研究開発では、送信先がプロキシかどうかを認識するとともに、その向こうに存在するサービス提供サーバのみが復号できるような仕組みを、前項の属性提供サーバに対してどのサービス提供サーバにアクセスしようとしているかというプライバシー情報の保護も考慮した技術の開発を行う。

## 2.7.1 実装方針

研究開発 2-3 の目標を達成するため、属性提供サーバが属性を提供する際に、サービス提供サーバのみが復号できるかたちで属性を暗号化する方法を検討した。

1）サービス提供サーバの実装方針

研究開発 2-2-1 の方針に加えて、UserInfo Request において毎回公開鍵・秘密鍵のペア（以下、鍵ペア）を生成して公開鍵を ID プロバイダに渡す機能、および UserInfo Response において集約クレーム中の暗号化された属性を復号する機能を実装した。

ID プロバイダが鍵ペアを生成し、サービス提供サーバのように振る舞って属性を取得することを防げるよう、サービス提供サーバが渡す公開鍵には信頼できるパーティにより署名をすることで ID プロバイダによるなりすましでないことを保証することとした。ただし今回の実証実験では、サービス提供サーバが提供する公開鍵への信頼できるパーティによる署名にかえて、サービス提供サーバが準備した署名用の鍵による署名とした。

また、属性提供サーバによる claim の署名の検証においては、サービス提供サーバは事前に信頼できる属性提供サーバの情報を所持しているものとした。

2）ID プロバイダの実装方針

研究開発 2-2-1 の方針に加えて、UserInfo Request において公開鍵が渡された場合、Claims Endpoint にその公開鍵を渡し、暗号化された属性を集約クレームの形でサービス提供サーバに返す機能を実装した。

3）属性提供サーバの実装方針

研究開発 2-2-1 の方針に加えて、Claims Endpoint において公開鍵が渡された場合、仕様にしたがって検証を行い、属性を暗号化した状態で提供する機能を実装した。

今回の実証実験では、サービス提供サーバが提供する公開鍵への信頼できるパーティによる署名にかえて、サービス提供サーバが準備した署名用の鍵による署名とし、属性提供サーバは署名を検証するための情報を事前に入手しているものとした。

## 2.7.2 成果及び結果

目標を達成するため、属性提供サーバが属性を提供する際に、サービス提供サーバのみが復号できるかたちで属性を暗号化する方法を検討した（図３４）。

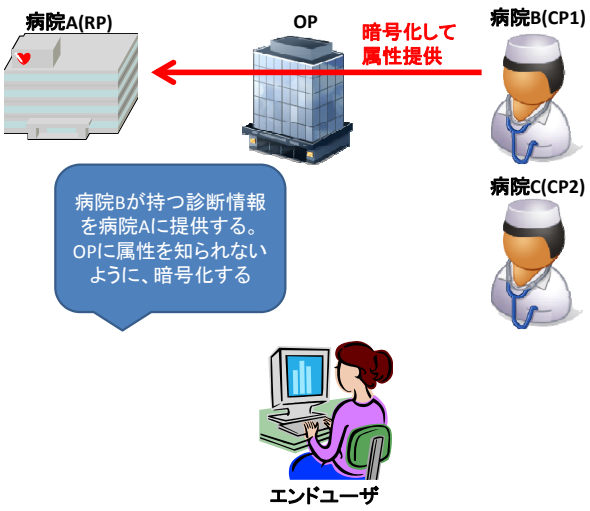



図３４

具体的には、以下の機能を実装した。

サービス提供サーバ（RP）への実装内容

- 公開鍵暗号方式である RSA 暗号に使用する公開鍵・秘密鍵のペア(以下、鍵ペア)を生成し、OP への属性情報のリクエスト時に公開鍵を付加して送信する機能
- 公開鍵を要求する CP の数に対して公開鍵の数が不足する意味の応答を OP が行った場合、必要とされるだけの鍵ペアを再度生成し、リクエストをリトライする機能
- TTP の存在を仮想し、その署名用秘密鍵を用いてリクエストに付加する公開鍵に署名を行う機能

ID プロバイダ（OP）への実装内容

- 属性情報のリクエストに付加された公開鍵群から CP 毎に異なった 1 個の公開鍵を選択し、属性情報のリクエストを行う機能
- リクエストに付加された公開鍵の数が CP の数より少ないときは、公開鍵を付加しないリクエストを先行して行い、鍵の使用を節約する機能
- CP が暗号化を行わない属性情報の提供を拒否した場合、その旨を応答し、RP により多くの鍵ペアの生成を促す機能

属性提供サーバ（CP）への実装内容

- 属性情報のリクエストに公開鍵が付加されている場合、それを用いて暗号化した属性情報を JWE フォーマットで提供する機能
- 属性情報のリクエストに公開鍵が付加されていない場合、属性情報の提供を拒否し、その旨を応答する機能
- TTP の存在を仮想し、RP の署名用公開鍵を用いて、リクエストに付加された公開鍵の署名を検証する機能

実装したシステムを用いて以下の事項について実証し、正常に動作することを確認した。

- サービス提供サーバが属性情報のリクエストに際して 1 つ以上の公開鍵を付加していること
- 属性提供サーバから返却される情報が JWE のフォーマットで暗号化されていること。またその鍵識別子がリクエストに用いられた公開鍵の鍵識別子と同一であること
- サービス提供サーバが属性情報を取得できていること
- 同時に属性情報のリクエスト対象となる複数の属性提供サーバに対して、ID プロバイダから同一の公開鍵が提供されないこと
- デフォルトで生成する鍵ペアの数よりも多い属性提供サーバが同時に属性情報のリクエスト対象となる場合、ID プロバイダのエラー応答を受けたサービス提供サーバが十分な数の鍵ペアを生成し、リトライを行うこと

以上の検証項目がすべて達成できたことから、本研究開発の目的・目標は達成できた。

## 2.8 WEB 以外のサービス提供サーバに同様の仕組みを提供する仕組みの研究開発（課題２－４）

WEB サーバにおいては、ID の氾濫とそれに伴うセキュリティ上の問題から、これまでに挙げたようないわゆる No Password の取り組みが始まっているが、メール等の他のプロトコルについては以前サービスごとの ID、パスワードを使う事が前提となっている。これらのプロトコルにおいても、No Password の仕組みを構築することで、ID 窃盗の問題等への対応を行う事が可能となる。

しかし、WEB 以外で認証を必要とする IMAP、SSH などのプロトコルでは、OpenID Connect 等の ID 連携の仕組みは現状では利用できない。

そのため、SMTP, IMAP 等のパスワードベースのプロトコルに対して、アクセストークンを受け入れてログインすることが出来るような技術を開発する。

### 2.8.1 成果及び結果

WEB 以外のサービス提供サーバ（以下、リソースサーバ）のサービスとしてメールサービスを選択し、ID 連携の仕組みとしてアクセストークンによる認証を実現することで、WEB 以外のサービスにおいても OpenID Connect 等の ID 連携の仕組みを実現可能であることを実証した。具体的には、下記の技術開発を実施した（図３５）。

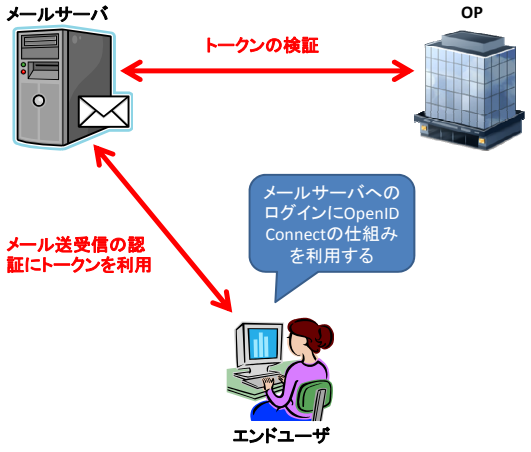



図３５

- 同一サービスへのリソースサーバ用の複数トークン（複数のデバイスやクライアントソフトウェア用）の発行

- リソースサーバへのトークンによるログイン
- トークンの失効の一元管理

パスワードによる認証が行われるサービスにおいてパスワードの代わりにトークンを使用するには、OpenID Connect の範疇にない部分についてプロトコルの仕様を決定する必要がある。
本研究開発では、パスワードに代わって使用されるトークンについて、以下の性質を満たす仕様とした。

- クライアントソフトウェアから受け取ったトークンをリソースサーバが検証可能
- クライアントソフトウェアから受け取ったトークンの scope をリソースサーバが確認可能
- クライアントソフトウェアから受け取ったトークンの失効をリソースサーバが確認可能

ID プロバイダは新たな性質のトークンを発行することになるため、新規の scope を定義し、サポートした。また、エンドユーザがクライアントソフトウェアを頻繁に再セットアップすることは、エンドユーザの利便性やシステム処理の面から好ましくないので、トークンにはデバイスの寿命といったスケールの有効期限を持たせた。通常 OpenID Connect で提供されるアクセストークンはセキュリティ上の理由から短寿命であることを求められるが、新規の scope を定義することによって通常のものと区別する。

2）リソースサーバのトークン検証方式

クライアントソフトウェアからリソースサーバへのアクセス時に、WEB インタフェースで紐付けた情報をもとにアクセストークンを検証させるため、リソースサーバ用に PAM サービスモジュールを作成し、検証機能を持たせた。
また、エンドユーザがクライアントソフトウェアを搭載するデバイスを紛失した場合等、即座にトークンの失効を反映させる必要性を考慮し、毎回リソースサーバが ID プロバイダへトークンの検証をリクエストする方式を用いた。

3）リソースサーバの WEB インタフェース

ID プロバイダからリソースサーバで利用可能なトークンを発行するためには、エンドユーザの操作に基づいて ID プロバイダと連携し、エンドユーザを認証して ID プロバイダのアカウントとリソースサーバのアカウントとを紐付ける手段をリソースサーバに実装しなければならない。

そこで、リソースサーバと密に連携する、エンドユーザがアクセス可能な WEB インタフェースを作成し、既存の ID とパスワードによってエンドユーザを認証し、使用する ID プロバイダの選択、トークンの取得と表示、ID プロバイダのアカウントとリソースサーバのアカウントの紐付け等を実行する方法を用いた。

4）クライアントソフトウェア

既存サービスへの影響を可能な限り小さくとどめ汎用性の高いサービスとするため、本研究開発ではクライアントソフトウェアに対する変更を行わないこととした。
本研究開発においては、ID プロバイダで発行されたアクセストークンを WEB ブラウザに表示し、コピー&ペーストによってクライアントソフトウェアのパスワード欄に、パスワードの代わりに入力する方法を用いた。

## 2.8.2 実装

メールサーバとその WEB インタフェースおよび ID プロバイダ（OP）は、いずれも Amazon EC2 上の Linux サーバを使用した。図３６に研究開発 2-1 の実証環境の概要図を示す。

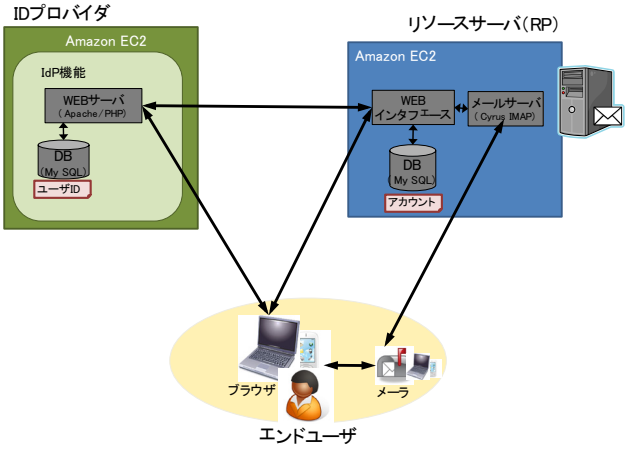



図３６　研究開発 2-4 の実証環境概要図

メールサーバには Cyrus IMAP をセットアップし、新たに導入するトークンによる認証と従来からのパスワードによる認証の双方を有効とするため、表１３に示す機能を備えた 2 つの PAM サービスモジュールをそれぞれ認証の十分条件として設定した。

メールサーバの WEB インタフェースおよびメールサーバ、ID プロバイダ（OP）には、表１３に示す機能を ruby で実装した WEB アプリケーションをセットアップした。また、データベースとして sqlite3 を使用した。

表１３　研究開発 2-4 で実装した機能

| 実装箇所 | 機能 No. | 機能 |
|---|---|---|
| WEBインタフェース | 機能 1 | ログイン ID およびパスワードの組でユーザを認証する機能 |
| | 機能 2 | エンドユーザに OP を選択させる画面を提示する機能 |
| | 機能 3 | OpenID Connect の RP として振る舞い、OP より取得したアクセストークンをエンドユーザに表示する機能 |
| | 機能 4 | Authorization Request を行う際に、リソースサーバに関連した scope を要求する機能 |
| | 機能 5 | ユーザにアクセストークンを発効した OP を紐づける機能（今回の研究開発では、1 つの OP のみ紐づくことが可能） |
| メールサーバ | PAM サービスモジュール 1 | |
| | 機能 1 | WEB インタフェースで使用するユーザ名およびパスワードの組でユーザを認証する機能（従来のパスワードログインを提供する機能） |
| | PAM サービスモジュール 2 | |
| | 機能 2 | パスワードをアクセストークンとして用い、WEB インタフェースで紐づけられた OP に対して UserInfo Request を発行する機能 |
| | 機能 3 | UserInfo Response より、PAM コ |

| 実装箇所 | 機能 No. | 機能 |
|---|---|---|
| | | ンシューマの提供するサービスが認可されているか判断する機能 |
| OP | 機能 1 | ログイン ID およびパスワードの組でユーザを認証する機能 |
| | 機能 2 | OpenID Connect の OP として振る舞い、リソースサーバのウェブインタフェースに対してアクセストークンを発行する機能 |
| | 機能 3 | エンドユーザが行った個々の認可に対応するアクセストークンを、エンドユーザが個別に無効化することを可能にする機能 |
| | 機能 4 | アクセストークンをエンドユーザが個別に無効化する操作を容易にするため、個々の認可にエンドユーザが自由に入力するラベルを紐づける機能 |
| | 機能 5 | UserInfo Response に、Authorization Request で要求された scope を含める機能 |

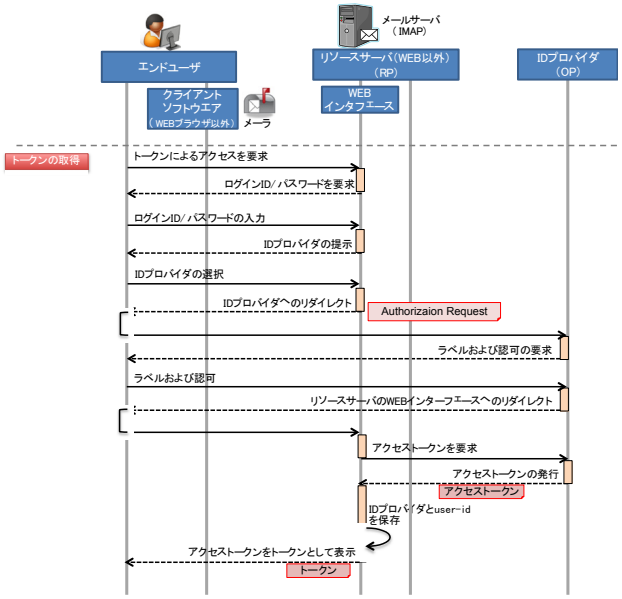



図３７　トークン取得部分のシーケンス図

本研究開発のシーケンス図を、メールサーバへのログイン認証時に送付するトークンの「トークンの取得部分」と、そのトークンを使ってメールサーバにアクセスする「IMAP アクセス部分」に分けて、それぞれの図３７、図３８に示す。
メールサーバへのトークンを発行するために、エンドユーザがメールサーバの WEB インタフェースでトークンを発行要求すると、ID プロバイダの選択を求められ、ID プロバイダ上で任意のデバイスの端末を識別するためのラベルと関連付けてトークンを発行し、メールサーバ上の WEB インタフェースでそのトークンが表示される。
その表示されたトークンを、メールサーバへのログイン時にパスワードの代わりに設定することにより、トークンによる認証が行われメールサービスの利用できる。

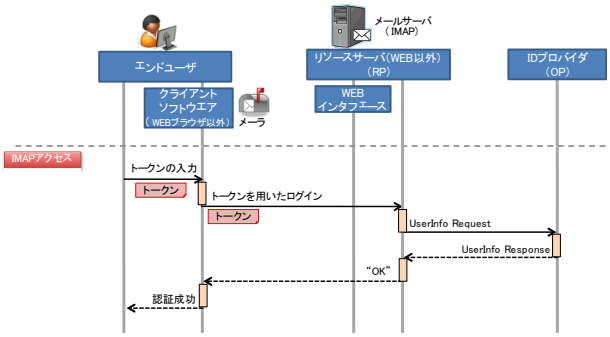



図３８　IMAP アクセスのシーケンス図

ID プロバイダには、Authorization Request の scope に auth_で開始するものが含まれる場合、通常の ID プロバイダとしての動作に加えて以下の動作をするように実装させた。

- openid scope と ID プロバイダでサポートする auth_で始まる scope のみを許可
- エンドユーザがすでに認可済みのリソースサーバの WEB インタフェースであっても新たな認可を要求（これにより、別々のデバイスなどで実行された複数の認可は別々のものとして扱い、任意のトークンの取り消しをエンドユーザができるようにした）
- 認可を行う際にエンドユーザが個々の認可を区別するための任意のテキスト（このテキストを「ラベル」と呼ぶ）の付加が可能（例えば、"Cellular"や"Notebook"など。このラベルにより、失効させたい任意のトークンをエンドユーザが容易に判断できるようにした）
- エンドユーザが行った個々の認可に対応するアクセストークンを、エンドユーザが個別に無効化することが可能
- UserInfo エンドポイントへのリクエストに対して表１４のように、その scope を含めて応答

表１４ ID プロバイダの UserInfo エンドポイントへの応答に含まれる Member（imap の場合の例）

| Member | Type | 説明 |
|---|---|---|
| auth_imap | JSON object | JSON object 内の Member は規定せず、空の{}とする（本研究開発で独自に定義） |

リソースサーバには、PAM サービスモジュールを用いて、トークンを認証に利用した場合、以下の動作をするように実装した。

1. ログイン ID に紐づいた ID プロバイダおよびユーザ ID を得る。ログイン ID に対して、リソースサーバの WEB インタフェースを用いた ID プロバイダの関連付けがなされていなければ認証失敗とする
2. ID プロバイダの UserInfo エンドポイントを、トークンを添付して呼び出す。呼び出しが失敗した場合は、認証失敗とする
3. UserInfo エンドポイントから取得した UserInfo Response に含まれる user_id がログイン ID に関連付いたものと一致していなければ、認証失敗とする
4. UserInfo エンドポイントから取得した UserInfo

Response に自らのサービス名である imap に auth_を前置した auth_imap クレームが含まれていなければ、認証失敗とする。

5. 以上のすべてにおいて認証失敗とならなかった場合は認証成功とする

実装したシステムにおいて、実際にトークンを発行し、取得した以下の 4 つのトークンを、それぞれのデバイスのクライアントソフトウェアに設定した。これらのクライアントソフトウェアから、メールサーバへの認証を行い、全てのデバイスのクライアントソフトウェアでメールサーバへの認証が成功することおよび、メールの送受信が行えることを確認した。

- Thunderbird の IMAP 用のトークン
- Thunderbird の SMTP 用のトークン
- iPhone の IMAP/SMTP 用のトークンの発行
- Putty の SSH 用のトークン

また、特定のクライアントソフトウェア用のトークンを失効させた場合、即時にそのクライアントソフトウェアだけが認証ができなくなるが、他のクライアントソフトウェアのトークンは並行して、有効な状態を保っていることを確認し、ID プロバイダ上で全てのトークンを一元管理する機能が実現できていることを確認した（表１５）。

表１５ トークン失効時のソフトウェアクライアントの認証状況

| 認証状況 ＼ 失効トークン | クライアントソフトウェア | | |
|---|---|---|---|
| | Thunderbird IMAP | Thunderbird SMTP | iPhone IMAP/SMTP |
| Thunderbird IMAP | × | ○ | ○ |
| Thunderbird SMTP | ○ | × | ○ |
| iPhone IMAP/SMTP | ○ | ○ | × |

○：認証成功　×：認証失敗

また、当初目標で想定していなかった SSH においても、ID プロバイダ（OP）ならびにリソースサーバ（RP）への‘auth_sshd’スコープの設定ならびに OpenSSH の設定変更を行うだけでトークンによる認証が可能となった。リソースサーバに今回開発した PAM サービスモジュールを利用する設定を行うことで、IMAP および SMTP 以外のプロトコルに当研究開発での認証認可を行えることが確認できた。

なお、本課題の実施において策定したプロトコルは、Internet-Draft として次の 2 つの規格を IETF において提案中である。

- draft-sakimura-oidc-structured-token-01, “Structured Access Token for Sharing Authorization Grant between a Resource Server and an Authorization Server”
- draft-sakimura-oidc-extension-nonweb-01, “Access Token as per Client Password for Non-Web Protocols”

## ３．今後の研究開発成果の展開及び波及効果創出への取り組み

研究開発成果の国際標準化としては、課題２－４「WEB 以外のサービス提供サーバに同様の仕組みを提供する仕組みの研究開発」において国際標準の提案を行っているところであり、引き続き国際標準獲得に向けての取り組みを行う。また、課題１－３「属性提供サーバに対するプライバシー保護にする研究開発」、課題１－４「プロキシに対するプライバシー保護に関する研究開発」等においてやりとりされる暗号化された属性情報については、従来の属性値として、暗号化された値をそのまま埋め込む実装となっているため、暗号化された値であることを示すデータ構造について検討を行い、必要に応じて標準化に向けた取り組みを行う。

研究開発成果の実用化については、課題１－１で実現した SAML-OpenID Connect 連携の仕組みを実際のサービスに適用し、学術向けの学割サービスの実現を目指す。すでに、このようなサービスの実現に向けて、OpenID Foundation Japan と NII が共同して SITF（Student Identity Trust Framework）を平成 24 年 3 月に立ち上げており、いくつかのサービス提供事業者と検討を始めている。1 年以内を目処に、サービスの試験運用を開始したい。また、課題１－２で実現した「フロントチャネルを用いた AP からの属性情報取得」および「AP セレクタ」については、学認 mAP と呼ぶ、組織をまたがったグループ情報を管理・提供する属性情報提供サーバに組み込む形での活用を予定している。SSO 技術をベースとした ID 連携プラットフォームが広がってきてはいるが、グループ情報はサービス提供サーバ側に閉じて管理されている場合が多く、ベンダーロックイン的な要因となっている。このグループ情報を、汎用的な属性情報提供サーバに分離することができれば、様々なサービスに共通な情報として利用できるようになり、サービス間の密な連携が促進されると考えられる。

本研究開発の成果は、SAML/Shibboleth や OpenID のミドルウェアの拡張として実装し、オープンソースとして公開することで日本や欧州のみならず、世界各国の ID 連携プラットフォームの構築およびその高度化に活用して頂けると期待できる。各国のフェデレーションでは、オープンソースの実装が広く利用されており、それぞれの事情に応じて研究開発へのとりくみが行われている。本研究開発の事例を広く紹介するとともに、研究開発の成果をオープンソースとして提供することは非常に価値の高い貢献である。

## ４．むすび

本研究開発の取り組みは、マイナンバー制度を始めとする、社会での様々な個人情報の管理に密接な関連がある。本研究開発の成果や考え方が、個人情報を取り扱う様々なシステムの開発・構築において参考になることにより、国民生活や社会システムにおける安全・安心に寄与できると幸いである。

**【誌上発表リスト】**

[1]Hiroyuki SATO, Yasuo OKABE, Takeshi NISHIMURA, Kazutsuna YAMAJI, Motonori NAMAMURA, “Privacy Enhancing Proxies in Attribute Release:Two Approaches”, Proceedings of The 7th IEEE International Workshop on Middleware Architecture in the Internet (MidArch

2013), (in Proceedings of The 37th Annual International Computer Software & Applications Conference (COMPSAC 2013)), (2013/7/22)
[2]Motonori NAKAMURA, Takeshi NISHIMURA, Kazutsuna YAMAJI, Hiroyuki SATO, Yasuo OKABE, “Privacy Preserved Simple Attribute Aggregation to Avoid Correlation of User Activities Across Shibboleth SPs”, Proceedings of The 7th IEEE International Workshop on Middleware Architecture in the Internet (MidArch 2013), (in Proceedings of The 37th Annual International Computer Software & Applications Conference (COMPSAC 2013))、(2013/7/22)
[3] 岡部 寿男、佐藤 周行、西村 健、山地 一禎、中村 素典、“属性提供サーバに対してサービス提供サーバを秘匿する匿名化プロキシ”、情報処理学会 DICOMO2013 シンポジウム予稿集, pp. 1958-1963 (2013/7/12)

**【国際標準提案リスト】**

[1]IETF, draft-sakimura-oidc-structured-token-01, “Structured Access Token for Sharing Authorization Grant between a Resource Server and an Authorization Server”, 提案:平成 25 年 2 月 18 日, 修正提案:平成 25 年 2 月 25 日, 採択:平成 25 年 2 月 25 日
https://datatracker.ietf.org/doc/draft-sakimura-oidc-structured-token/
[2]IETF, draft-sakimura-oidc-extension-nonweb-01, “Access Token as per Client Password for Non-Web Protocols”, 提案:平成 25 年 2 月 18 日, 修正提案:平成 25 年 2 月 25 日, 採択:平成 25 年 2 月 25 日

**【参加国際標準会議リスト】**

[1]86th IETF (Internet Engineering Task Force)、Orland (Florida, USA)、March 10-15, 2013

**【本研究開発課題を掲載したホームページ】**

http://peofiamp.nii.ac.jp/