Vol.2020-SPT-36 No.52
2020/3/3

属性ベース暗号方式を用いた FIDO2の拡張による代理認証の実現

大川 悠人[1,a)]　猪俣 敦夫[2,3]　上原 哲太郎[1,b)]

概要：オンラインサービスの認証においてパスワード認証方式が用いられてきたが，安全性と利便性の面から課題が指摘されている．この課題に対して，生体認証等を用いたパスワードレスな認証仕様である FIDO が存在する．FIDO では公開鍵暗号方式を採用しており，クレデンシャル情報がネットワーク上を流れないため安全な認証方式であるとされている．しかし，FIDO 認証において秘密鍵や生体情報などのクレデンシャル情報を共有することが難しく，利用者本人でない代理人の認証を行うのは難しい．そこで，秘密鍵に属性を付与することでアクセス制御を行うことができる属性ベース署名を用いて FIDO の仕様を拡張することで，代理人の代理認証を行う仕組みを提案する．

Implementation of delegated authentication by extending FIDO2 using attribute-based encryption

1. はじめに

様々なサービスで，認証手段として以前からパスワード認証方式が用いられてきた．しかし，安全性と利便性の面から課題があると指摘されている．

このような課題に対して，パスワードへの依存度を減らしつつ，利便性と安全性の両面を向上させる認証方式として FIDO Alliance が提案したパスワードレス認証方式の仕様である FIDO[1] が存在する．パスワードの代わりに利用者のデバイスなどの認証器にて生体認証等のローカル認証を行うことで，パスワードを用いずに認証を行う．FIDO 認証では公開鍵暗号方式を用いていて，利用者の指紋などのクレデンシャル情報がネットワーク上を流れず安全な認証方式であるとされている．

FIDO は安全な認証仕様であるが，利便性の面から見ると課題が存在している．それは，利用者本人ではない代理人がアクセスする際に，生体情報や秘密鍵などクレデンシャル情報の共有が難しいことにより，代理人による認証が難しい．パスワード認証ではパスワードを代理人と共有することで認証を行うことができるが，FIDO 認証の場合，デバイスに格納された秘密鍵を取り出すことは難しい．また，パスワード認証においてパスワードを人と共有することにはリスクが存在し，危険とされているため，安全に代理認証を行う必要がある．

そこで，本研究では FIDO が公開鍵暗号方式を用いることに着目し，鍵に属性を付与してアクセス制御ができる属性ベース署名を用いて FIDO の仕様を拡張することで，代理人による認証を実現するための提案し，実現性を検証した．

2. 背景

2.1 FIDO

FIDO[1] は，オンラインサービスの認証において指紋認証や顔認証等の生体認証を用いて，簡単で安全な認証を行うための仕様である．**図 1** に，FIDO 認証モデルを示す．FIDO 認証では，利用者のスマートフォンや市販の USB セキュリティキー [2], [3] などのデバイスを認証をするための認証器として扱う．認証器にて生体認証等を用いて利用者本人であることをローカルで認証することで，オンライ

[1] 立命館大学 情報理工学部
College of Information Science and Engineering, Ritsumeikan University
[2] 立命館大学 総合科学技術研究機構
Research Organization of Science and Technology, Ritsumeikan University
[3] 大阪大学
Osaka University
[a)] ookawa@cysec.cs.ritsumei.ac.jp
[b)] t-uehara@fc.ritsumei.ac.jp

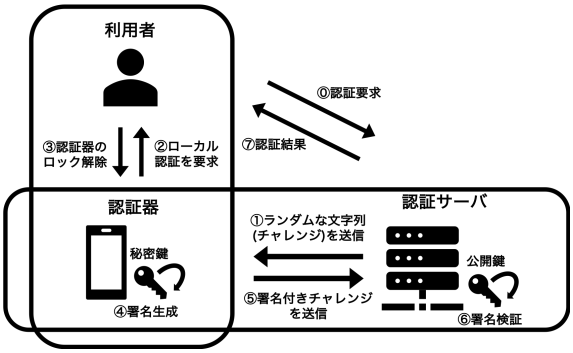

図 1　FIDO 認証モデル
Fig. 1　Model of FIDO authenticate.

ンサービス上で認証が行える．認証器は，ローカル認証の結果のみを認証処理を行う Web サーバ (以下，FIDO サーバ) へ送る．ここで，公開鍵暗号方式による電子署名が用いられている．

サービス利用登録時に，認証器内部にて公開鍵と秘密鍵のペアを生成する．通常秘密鍵は TEE(Trusted Execution Environment)[4] などの安全な領域へ保存され，公開鍵は FIDO サーバにて利用者の ID と関連付けて登録される．

サービス認証時に利用者が認証器にてローカル認証を行い，認証器内の秘密鍵を用いて FIDO サーバから送られたランダムな文字列 (チャレンジ) に対して署名を生成し，FIDO サーバに送信する．その後，登録されている公開鍵を用いて署名検証を行い，その結果を認証の結果としている．これは，FIDO サーバが公開鍵を用いて適切な署名であることを検証できれば，その公開鍵に対応した秘密鍵を認証器が持っていることを確かめることができるためである．このように公開鍵暗号方式を用いることで，従来のパスワード認証とは異なり，ID と関連づいたパスワードなどのクレデンシャル情報がネットワーク上を流れず，安全な認証が行える．

また，この FIDO 認証には FIDO UAF，U2F，FIDO2 と複数の仕様が存在する．

2.1.1　FIDO UAF

FIDO UAF は，利用者が対応したデバイスを FIDO サーバに登録することで，デバイスのローカル認証によってパスワードレスな認証を実現する仕様である．主にスマートフォンなどを想定している．対応したデバイス内で公開鍵と秘密鍵のペアを生成し，秘密鍵をデバイス内に保存して，公開鍵を FIDO サーバに登録する．

2.1.2　FIDO U2F

FIDO U2F はパスワード認証方式に加えて，二要素目として FIDO 認証を用いる仕様である．従来の認証方式に簡単に組み込むことができ，パスワード認証の脆弱性に対して対策が行える．U2F では，USB セキュリティキーや，BLE(Bluetooth Low Energy) や NFC(Near Field Communication) の無線方式を搭載したデバイスを用いる．

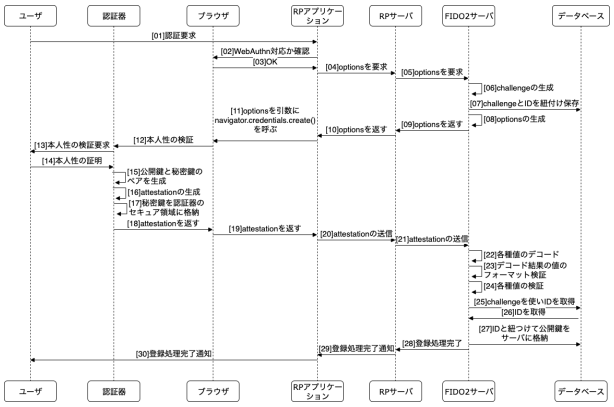

図 2　FIDO2 登録シーケンス図 [7]
Fig. 2　Sequence Diagram of FIDO2 registration.[7]

2.1.3　FIDO2

FIDO2 は，UAF と U2F を統合したものであり，W3C の Web 認証仕様である WebAuthn[5] と，FIDO Alliance が策定したデバイス間連携仕様である CTAP[6] で構成された仕様である．WebAuthn は，Web ブラウザに FIDO 認証を組み込むための JavaScript の API を定義しており，現在 Chrome，Edge，Firefox，Safari にて実装されている．この API を呼び出すことで，Web ブラウザが利用者に認証器との接続を要求し，FIDO 認証を行うことができる．これにより，サービス事業者が専用のアプリケーションを用意することなく，FIDO 認証を扱うことができる．CTAP は外部または内部のデバイスの認証器を用いて認証を行うための通信プロトコルである．これにより，認証を行う際に，外部にあるスマートフォンなどのデバイスの認証器を用いて認証を行える．

図 2 は，FIDO2 における登録時のシーケンス図である．

(1) Web ブラウザ上のアプリケーションは FIDO サーバに対して登録要求を行う．
(2) FIDO サーバは，ブラウザに実装されている JavaScript 登録関数の引数として必要な値を生成してアプリケーションに送信する．
(3) アプリケーションは受け取った値を引数にブラウザに実装されている JavaScript の登録関数を実行する．
(4) 利用者は Web ブラウザから要求された方法で認証器を接続する．
(5) 利用者は生体認証や PIN などを用いて認証器のロックを解除する．
(6) 認証器内にて鍵ペアが生成され，秘密鍵をセキュアな領域に格納する．
(7) 公開鍵と検証用の値を生成し，関数の返り値として Web ブラウザに返す．
(8) アプリケーションが FIDO サーバにその返り値を送信する．
(9) FIDO サーバは値の正当性を検証後に公開鍵を登録する．

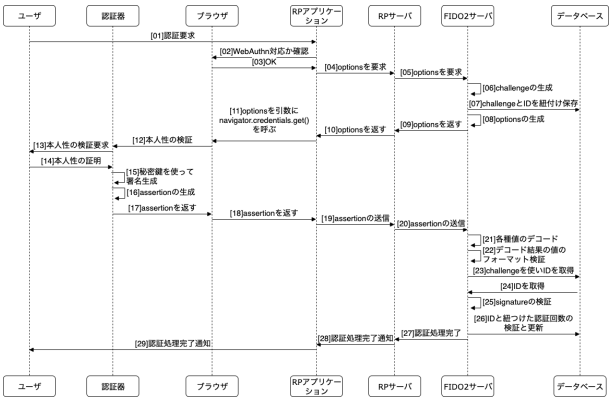

図 3 FIDO2 認証シーケンス図 [7]
Fig. 3 Sequence Diagram of FIDO2 authenticate.[7]

FIDO サーバは，登録時に必要な値としてチャレンジを生成する．このチャレンジは一時的にデータベースに保存し，登録関数の返り値を受け取った際に検証用の値として用いる．

図 3 は，FIDO2 における認証時のシーケンス図である．

(1) Web ブラウザ上のアプリケーションは FIDO サーバに対して認証要求を行う．
(2) FIDO サーバは，ブラウザに実装されている JavaScript の認証関数の引数として必要な値を生成してアプリケーションに送信する．
(3) アプリケーションは受け取った値を引数にブラウザに実装されている JavaScript の認証関数を実行する．
(4) 利用者は Web ブラウザから要求された方法で認証器を接続する．
(5) 利用者は生体認証や PIN などを用いて認証器のロックを解除する．
(6) 認証器は認証器内に格納されている秘密鍵を用いて署名を生成する．
(7) 署名と検証用の値を生成し，関数の返り値として Web ブラウザに返す．
(8) アプリケーションが FIDO サーバにその返り値を送信する．
(9) FIDO サーバに登録されている公開鍵を用いて署名の検証を行う．

2.2 属性ベース署名

属性ベース署名 (Attribute-Based Signature，ABS)[8] は，属性ベース暗号方式を用いた電子署名の一種である．属性ベース暗号方式は公開鍵暗号方式の一種であり，公開鍵暗号方式が公開鍵と秘密鍵が 1 対 1 の関係になることに対し，公開鍵と秘密鍵が 1 対多や多対 1 の関係となる暗号方式である [9]．属性ベース署名では，1 つの公開鍵に対して 1 つのマスター秘密鍵が存在し，そのマスター秘密鍵に属性情報を付与することで，属性情報が埋め込まれたユーザ秘密鍵を生成する．ここで，公開鍵とユーザ秘密鍵が 1 対多の関係となる．ユーザ秘密鍵を用いてある値に対して署名を生成する際に，属性の and や or の組み合わせによる論理式で表されたアクセス条件が必要となり，生成された署名にはこのアクセス条件が埋め込まれている．署名検証では，署名に埋め込まれたアクセス条件を満たすような属性を持つ秘密鍵で作成された署名であることを検証することができる．これにより，アクセス制御に相当するものを電子署名に組み込むことができる．

ABS のアルゴリズムは，以下の 5 つからなる．

ABS.TSetup: $TPK \leftarrow ABS.TSetup()$
公開参照情報鍵 TPK を生成する．

ABS.ASetup: $ASK, APK \leftarrow ABS.ASetup(TPK)$
マスター秘密鍵 ASK，公開鍵 APK の鍵ペアを生成する．

ABS.AttrGen: $SK_A \leftarrow ABS.AttrGen(ASK, A)$
属性の集合 A，マスター秘密鍵 ASK を入力とし，A と対応したユーザ秘密鍵 SK_A を出力する．

ABS.Sign: $\sigma \leftarrow ABS.Sign((TPK, APK), SK_A, m, S)$
公開鍵ペア (TPK,APK)，ユーザ秘密鍵 SK_A，メッセージ m，属性集合 A を満たすようなアクセス条件 S を入力とし，署名 σ を出力する．

ABS.Ver: $1 or 0 \leftarrow ABS.Ver((TPK, APK), m, S, \sigma)$
公開鍵ペア (TPK,APK)，メッセージ m，アクセス条件 S，署名 σ を入力とし，真偽値を出力する．

ABS では，鍵生成局を利用する必要がある．TSetup で署名受諾者が公開参照情報鍵 TPK を生成し，属性発行機関が ASetup を実行してマスター秘密鍵 ASK と公開鍵 APK を生成する．また利用者の要求を受け，鍵生成局が AttrGen でマスター秘密鍵 ASK と属性集合 A を用いて，ユーザ秘密鍵 SK_A を生成する．

2.3 関連研究

大森らは，FIDO 認証において端末の安全な領域に ID プロバイダが証明する所有者情報を埋め込み，これを検証することで同一所有者の端末間での権限共有や移譲を実現する方法として，異なる ID プロバイダが連携をして同一性を検証する方法を提案した [10]．この研究では FIDO 認証に用いる鍵ペアをキーストアと呼ばれる Web サーバ内で一括管理することで他のデバイスと鍵ペアを共有するキーストア方式，あるデバイスの TEE 領域に格納している鍵ペアを別デバイスの TEE 領域にセキュアにコピーするコピー方式がある．これらの方式について，異なる ID プロバイダであっても同一所有者であることを検証している．

本研究では，FIDO 認証自体に鍵共有に近い手段を組み込むことを提案する．

3. 提案手法

3.1 FIDO の拡張性

FIDO は，従来のパスワード認証方式と比較して，簡単で安全な認証ができる認証方式である．しかし FIDO において，パスワード認証方式と比べて，クレデンシャル情報を共有するのが困難である．FIDO ではクレデンシャル情報である秘密鍵を TEE など認証器内の安全な領域に格納しているので，取り出して他の認証器と共有することは一般には難しい．オンラインサービスの認証において，利用者に管理が必要になる場合に，管理者の認証器からアクセスできた方が利便性は高まる．また，緊急時に利用者以外のアクセスが必要になった場合には，利用者本人の認証器でなければ認証できない場合がある．これらのことから，FIDO には利便性を向上する余地があると考える．

また，FIDO は暗号技術として公開鍵暗号方式を用いている点に着目し，公開鍵暗号方式には様々な種類があり [9]，現在 FIDO で採用されている暗号方式以外を扱うことで仕様拡張を行うことができると考える．

3.2 本研究で提案する手法

3.1 節で述べたような拡張を行うために，鍵に属性情報を与えることができる属性ベース署名を用いる．これにより，利用者の秘密鍵に属性を付与することで，利用者本人ではない代理人のアクセスが可能な仕組みを FIDO 認証に組み込む．そこで本提案手法では，多くの利用例を考えるために，Web ブラウザ上で扱うことができる FIDO の仕様の一つである FIDO2 を想定する．

WebAuthn は属性ベース署名のアルゴリズムには対応していないため，Web ブラウザに実装されている関数をそのまま扱うことができない．したがって，本研究では，WebAuthn を模した登録模擬関数と認証模擬関数を JavaScript を用いて認証を必要とするアプリケーション上に作成し，属性ベース署名のアルゴリズムに対応した認証器を模したプログラムを作成する．作成したアプリケーションからそれぞれの関数を呼び出すことで，直接認証器と接続して FIDO 認証の処理を行う．

FIDO 認証に属性ベース署名を組み込むにあたり問題がある．それは，通常の属性ベース署名の運用と同様に一つのマスター秘密鍵と公開鍵のペアで運用すると，ID が異なっていてもユーザ秘密鍵の持つ属性が一致したことにより署名検証に成功してしまう．また，属性ベース署名は予め属性集合を定義し，属性集合を更新する場合には全ての鍵を更新する必要がある．そのため，ID を属性集合に加えると利用者が増える度に属性集合を更新する必要がある．また，属性ベース署名のアルゴリズムにかかる時間が属性の数に応じて増えていくので，属性集合に利用者の ID を加えることは難しい．そこで本提案手法では，鍵生成局は利用者の ID ごとにマスター秘密鍵と公開鍵を発行し，予め鍵生成局で受け入れる属性集合を定めておくことで代理人の認証を実現することとする．また，認証段階における署名生成時のアクセス条件はアプリケーション上で決めておくものとする．

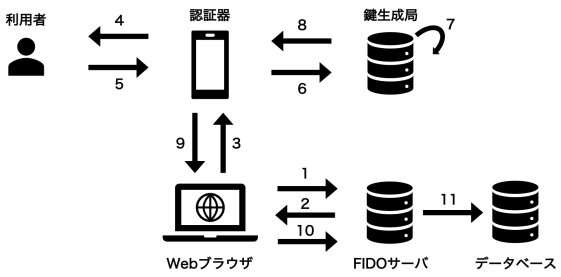

図 4 提案手法による登録モデル
Fig. 4 Proposed Registeration model.

3.3 構成

本提案手法の構成について，登録段階と認証段階の二段階について述べる．構成要素として以下の 5 つが挙げられる．

FIDO サーバ: FIDO 認証に必要な値を生成し，公開鍵登録や署名検証などの処理を行う Web サーバ．

データベース: 利用者の ID と紐つけて，公開鍵や登録属性などを格納するデータベース．

認証器: 利用者が FIDO 認証を行うために用いるデバイス．

鍵生成局: 属性ベース署名のアルゴリズムにしたがって鍵を生成する Web サーバ．

アプリケーション: Web ブラウザ上で動作する Web アプリケーション．

3.3.1 登録段階

利用者の登録は以下のようなフローで行われる．登録モデルを図 4 に示す．ここで，登録模擬関数の引数を options，返り値を attestation とする．

(1) アプリケーションにて ID と属性を入力し，それらを FIDO サーバに送信する．
(2) FIDO サーバはランダムな文字列 (以下，チャレンジ) を生成し，チャレンジと属性を含んだ options を生成してアプリケーションに返す．
(3) アプリケーションは，options を引数に登録模擬関数を実行して，認証器と接続する．
(4) 認証器は利用者にローカル認証を要求する．
(5) 利用者は生体認証等を用いて認証器のロックを解除する．
(6) ロックを解除した認証器は鍵生成局に対して属性を送信して鍵生成要求を行う．
(7) 鍵生成局は新規 ID であればマスター秘密鍵・公開鍵

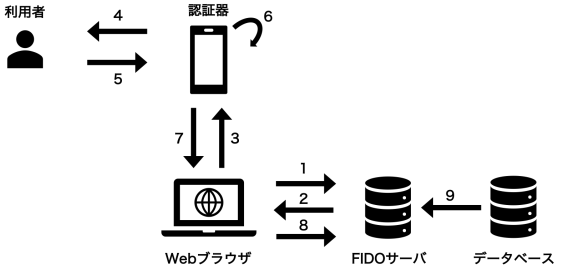

図 5　提案手法による認証モデル
Fig. 5　Proposed Authentication model.

を生成し，受け取った属性とマスター秘密鍵からユーザ秘密鍵を生成する．

(8) 鍵生成局は認証器に公開鍵とユーザ秘密鍵を送信する．

(9) 認証器は公開鍵の値と自身の正当性を検証するための値を含んだ値である attestation を生成し，アプリケーションに関数の返り値として返す．

(10)アプリケーションは，attestation を FIDO サーバに送信する．

(11)FIDO サーバは attestation の正当性を検証し，公開鍵ペアと属性をデータベースに登録する．

3.3.2　認証段階

利用者の認証は以下のようなフローで行われる．認証モデルを図 **5** に示す．ここで，認証模擬関数の引数を options，返り値を assertion とする．

(1) Web ブラウザ上のアプリケーションにて ID を入力し，アプリケーション側で設定したアクセス条件と ID を FIDO サーバに送信する．

(2) FIDO サーバはチャレンジを生成し，チャレンジとアクセス条件を含んだ options を生成してアプリケーションに送信する．

(3) アプリケーションは，options を引数に認証模擬関数を実行して，認証器と接続する．

(4) 認証器は利用者にローカル認証を要求する．

(5) 利用者は生体認証を用いて認証器のロックを解除する．

(6) ロックを解除した認証器は登録段階で保存したユーザ秘密鍵を用いて署名を生成し，その署名を含んだ assertion を生成する．

(7) 認証器は assertion を関数の返り値としてアプリケーションに返す．

(8) アプリケーションは，assertion を FIDO サーバに送信する．

(9) FIDO サーバは assertion の検証を行い，データベースに保存されている公開鍵を用いて assertion に含まれる署名を検証しアプリケーションに結果を送信する．

3.4　運用について

提案手法である属性ベース暗号方式による電子署名を用いた FIDO 認証について，運用する上で注意しておく点を以下に述べる．

まず，FIDO 認証の特徴として，以下の 4 つが挙げられている．[11]

- FIDO 認証プロトコルは end to end であり，第三者の介在はない
- サーバ側で秘密情報が生成されたり保存されることはない．(FIDO 認証の鍵ペアのうち，秘密鍵は FIDO 認証器の外に出ない)
- 生体情報は FIDO 認証器に保存され，外に出ない
- 異なるサービス・アカウントに対して，FIDO 認証の鍵ペアは独立

今回の提案手法である属性ベース署名を用いた FIDO 認証では，鍵生成の際に鍵生成局を用いる．認証器は鍵生成局からユーザ秘密鍵を受け取るため，認証器の外に秘密鍵が出ないということに反する．このため，認証器と鍵生成局の通信は SSL/TLS を利用するなどして，暗号化して運用するものとする．

次に，本提案手法では属性ベース署名を FIDO に適応させるために，鍵生成局はマスター秘密鍵を利用者の数発行することとしている．これにより，利用者の数のマスター秘密鍵を管理することとなるため，鍵管理コストが膨大になってしまう恐れがある．マスター秘密鍵が漏洩した場合，ユーザ秘密鍵を偽造することができてしまうため，厳重に管理をする必要がある．万が一漏洩した場合，そのマスター秘密鍵から作られたユーザ秘密鍵や対応した公開鍵を失効する必要があり，再発行の手間がかかるなどの問題を考慮する必要がある．このため，鍵生成局においても，通常アクセスできない安全な領域にて鍵を管理して十分漏洩に気を付ける必要がある．

4.　実装

属性ベース署名を用いた FIDO 認証の実現性検証のため，認証を実行する Web アプリケーション・FIDO サーバ・認証器・鍵生成局のプログラムを実装した．

4.1　実装環境

FIDO サーバ，鍵生成局，認証器のプログラムは Node.js を用いて実装した．認証を実行する Web アプリケーションは，Vue.js を用いて実装した．また，属性ベース署名のアルゴリズムの実装は Python を用いて実装し，Node.js 側から呼び出すこととした．属性ベース署名のアルゴリズムは，[8] のアルゴリズムを用いて実装した．このアルゴリズムにはペアリングと呼ばれる演算が必要となるため，ペアリング計算用のライブラリとして，Python のペアリングライブラリである Charm-Crypto[12] を用いた．属性ベース署名のアルゴリズムの実装にあたり，[13] を参考にした．FIDO サーバの実装と WebAuthn の処理については，WebAuthn のドキュメント [5] と共に [7] を参考にし

表 1　使用した主な言語・ソフトウェア・OS
Table 1　Main language, software, and OS.

ソフトウェア	バージョン
Node.js	12.14.0
Vue.js	3.11.0
Python	3.6.9
Charm-Crypto	0.50
Ubuntu	18.04.3

た．今回実装にあたって使用した主な言語・ソフトウェアの詳細を**表 1** に示す．

4.2　仕様

実装した各プログラムの仕様を述べる．

4.2.1　認証を必要とする Web アプリケーション

登録段階にて実行する実行模擬関数，認証段階にて実行する認証模擬関数をアプリケーション上に実装した．登録模擬関数の返り値 (以下，attestation) と認証模擬関数の返り値 (以下，assertion) はそれぞれ WebAuthn にて定義されている API の返り値と同じ構成としている．その構成は，[5] の 5.1 に記述されている．Web アプリケーション上にて登録及び認証を行う際に，FIDO サーバに対してそれぞれの関数を実行するために必要な値を要求して受け取り，各模擬関数を実行する．また，この関数の返り値を受け取った Web アプリケーションはその返り値を FIDO サーバに送信し，登録及び認証の結果を受け取る．

4.2.2　FIDO サーバ

FIDO サーバでは，4 つのエンドポイントを用意した．

/attestation/options:　登録段階において，登録模擬関数の引数として必要な値である options を生成して返す．ここではリクエストパラメータとして，利用者の ID と属性集合を受け取る．今回の実装では，FIDO2 の登録関数に必要な options に利用者の指定した属性を加えている．これは，属性ベース署名ではユーザ秘密鍵を作成する際に属性が必要となり，認証器に属性を渡す必要がある．また，生成したチャレンジと受け取った属性は ID と紐つけてデータベースに一時的に格納しておく．

/attestation/result:　登録段階において，attestation の正当性を検証し，公開鍵を取り出してデータベースに登録する．ここではリクエストパラメータとして，attestation を受け取る．最初に，含まれている各種パラメータを検証する．検証する値としてチャレンジと origin がある．チャレンジはデータベースに一時的に格納していたものと attestation に含まれるものが同一かどうかを検証し，origin はそのハッシュ値が attestation に含まれているので取り出して同一 origin か検証する．パラメータの検証後，attestation 自体の検証を行う．attestation の検証方法には複数のフォーマットが存在し，その中でも今回は WebAuthn 仕様に最適化されたフォーマットである packed[14] を対象とした．全ての検証が成功したのち，公開鍵をデータベースに格納する．

/assertion/options:　認証段階において，認証模擬関数の引数として必要な値である options を生成して返す．ここではリクエストパラメータとして．利用者の ID と属性の論理式で表されたアクセス条件を受け取る．今回の実装では，FIDO2 の認証関数に必要な options にアクセス条件を加えている．属性ベース署名では署名を生成する際にアクセス条件が必要となるため，認証器にアクセス条件を渡す必要がある．また，生成したチャレンジと受け取ったアクセス条件は ID と紐つけてデータベースに一時的に格納しておく．

/assertion/result:　認証段階において，assertion を検証し，署名を取り出して検証する．ここではリクエストパラメータとして，assertion を受け取る．最初に，含まれている各種パラメータを検証する．検証する値は，登録段階と同様にチャレンジと origin である．パラメータの検証後，登録段階にてデータベースに格納した公開鍵を用いて，assertion に含まれる署名を検証する．この時のアクセス条件は，一時的に ID と紐つけてデータベースに格納したアクセス条件を取り出して用いる．署名検証は Python のプログラムを呼び出して実行している．署名を検証後，検証結果を返す．

4.2.3　認証器

認証器は，Web アプリケーション上にて実行された登録模擬関数及び認証模擬関数によって呼び出され，それぞれに対応した処理を行う．そのため，今回は CTAP[6] の仕様を用いた実装は行っていない．

登録段階:　登録模擬関数によって呼び出される．FIDO サーバにて生成された options を受け取り，鍵生成局へ鍵生成要求を行いユーザ秘密鍵と公開鍵ペアを受け取る．受け取ったユーザ秘密鍵を認証器内に保存し，attestation を生成して Web アプリケーションに値を返す．認証器は options から属性を取り出し，鍵生成局に対して鍵生成要求を行う．鍵生成局から公開鍵ペアとユーザ秘密鍵を受け取った認証器は，鍵を認証器内に保存する．この 2 つの鍵は，認証段階の署名生成に用いられる．なお，公開鍵ペアは公開参照情報鍵と公開鍵の 2 種類で構成されており，attestation を生成する際には公開参照情報鍵と公開鍵を空白でつないだものを 1 つの公開鍵としている．

認証段階:　認証模擬関数によって呼び出される．FIDO サーバにて生成された options を受け取り，登録段階で保存したユーザ秘密鍵と公開鍵を用いて署名生成を行う．署名生成後に assertion を生成し，Web アプリ

図 6　登録画面
Fig. 6　Snapshot of registration.

ケーションに値を返す．署名生成は Python のプログラムを呼び出して実行している．署名生成の際，アクセス条件は options に含まれている policy を用いて署名を生成する．

4.2.4　鍵生成局

鍵生成局では，認証器からのリクエストを受けて，属性ベース署名のマスター秘密鍵と公開鍵の生成するアルゴリズムである ASetup，そしてユーザ秘密鍵の生成を行うアルゴリズムである Attrgen を実行する．また，鍵生成局では事前に TSetup を実行し，公開参照情報鍵を予め生成しておく．鍵生成局にて鍵生成を行う際には，Python のプログラムを呼び出して実行している．

ASetup:　マスター秘密鍵を生成し，公開参照情報鍵とマスター秘密鍵から公開鍵を生成する．その後，マスター秘密鍵と公開鍵のペアに対して固有の ID を生成し，その ID を返す．

Attrgen:　リクエストパラメータとして鍵ペアの ID と利用者が指定した属性を受け取る．受け取った鍵の ID に紐ついたマスター秘密鍵と，リクエストパラメータとして受け取った属性を用いてユーザ秘密鍵を生成する．その後，生成したユーザ秘密鍵と公開鍵ペアを返す．

5.　評価

評価として，実装した FIDO サーバ，鍵生成局，認証器を用いた認証モデルを作成した．また，属性ベース署名の各アルゴリズムに要した時間を計測し，評価を行った．

5.1　認証モデル

認証モデルとして，保護者が子供のアカウントにアクセスするモデルを作成した．用意した属性集合は PARENT，CHILD，OTHERS とした．また，アプリケーション上で決めるアクセス条件は PARENT OR CHILD とした．これにより，CHILD または PARENT を含んだ属性集合で生成された秘密鍵を用いて生成された署名のみ署名検証が成功するようになる．

図 6 にて作成した認証モデルにて登録する様子を表す．利用者は登録する際に自分の ID を入力し，自分の属性を選択して登録要求ボタンを押す．これにより登録段階による一連の処理が行われ，起動している認証器プログラム内にユーザ秘密鍵が保存される．

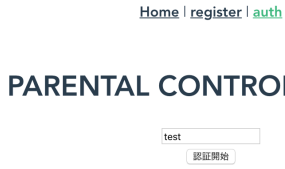

図 7　認証画面
Fig. 7　Snapshot of authentication.

図 7 にて認証する様子を表す．利用者は認証を行う ID を入力し，認証要求ボタンを押す．これにより認証段階による一連の処理が行われ，認証器プログラム内で作成した署名を FIDO サーバで検証する．今回はアクセス条件を CHILD OR PARENT としているので，CHILD または PARENT の属性を含んだ秘密鍵を持っていれば認証を成功することができる．

このように，子供の認証器に CHILD 属性を持つ秘密鍵を，保護者の認証器に PARENT 属性を持つ秘密鍵を生成することによって，例えば子供のアカウントに保護者がアクセスできるようになることを実現した．

5.2　処理時間の計測

実装した各プログラムについて，それぞれ処理時間を計測した．計測に関しては，5.1 節のモデルにて 10 回登録と認証を行った結果を平均している．また，比較対象として，FIDO2 に対応した FIDO サーバ [15] を用いて，同様に署名検証を行う処理を計測した．**表 2** に，計測に使用した環境を示す．

表 2　処理時間の計測環境
Table 2　Environment of processing time.

OS	OS バージョン	CPU	メモリ
Ubuntu	18.04.3 LTS	intel Core i7-7700	32GB
MacOS	Mojave 10.15.6	intel Core i5-5250U	4GB

鍵生成局，FIDO サーバは計算能力の高いものを想定し上記 Ubuntu 環境下で計測した．認証器のプログラムは計算能力による署名生成にかかる時間の差異を見るため，Ubuntu と MacOS 共に計測を行った．**表 3** に計測結果を示す．

FIDO2 対応の FIDO サーバと今回の実装の FIDO サーバと比較すると，処理にかかる時間は増加している．また，認証器を模したプログラムは，結果として 2 つのデバイスにおける計測結果に差が生じた．

6.　考察

評価として保護者が子供のアカウントにアクセスするシ

表 3　処理時間の計測結果
Table 3　Measurement result of processing time.

プログラム	計測ポイント	平均処理時間
FIDO2 対応	/attestation/result	71.0578ms
FIDO サーバ	/assertion/result	20.844ms
ABS 対応	/attestation/result	174.5988ms
FIDO サーバ	/assertion/result	155.7288ms
鍵生成局	ASetup	3.4336ms
	Attrgen	1.3499ms
認証器	登録	359.9383ms
(Ubuntu)	認証	177.5458ms
認証器	登録	1380.5638ms
(MacOS)	認証	1038.4797ms

ステムを構築し，実際に利用者本人でない代理人の認証の実現性を確認することができた．

FIDO サーバの計測結果から，FIDO2 の仕様より署名検証にかかる時間が増加したと考えられる．このことから，FIDO2 の仕様に近づけるためには用いたペアリングライブラリの選定や実装の最適化を行う必要があると考える．また，認証器プログラムの計測結果から計算能力による差が大きくみられた．本来は生体認証等を用いて認証器のロックを解除する過程を含み，その時間を考慮すると認証器の処理にかかる時間は無視できると考える．しかし，幅広い利用を考えるためには FIDO サーバと同様の対策を行う必要があると考える．

属性ベース署名を運用するにあたり鍵生成局が必要となり，認証器が鍵生成局と通信を行う必要があるため，認証器自体がネットワークに接続されている必要がある．今回は認証器のプログラムを Node.js を用いて実装したが，実際の利用用途を考えるために，主にスマートフォン上の認証器について焦点を当てて実装を行っていくことが考えられる．さらに，今回はアプリケーション上に模擬関数を実装したが，実際に Web ブラウザ上に今回の模擬関数を API として実装することで，CTAP[6] を用いて外部の認証器と通信することも考えられる．

7. おわりに

本研究では FIDO2 の仕様に属性ベース署名を用いて，利用者本人でない代理認証を行うための拡張を提案し，実装を行った．FIDO 認証において，システム的に代理認証を行うのではなく FIDO 認証自体に代理認証を組み込むことで，利便性の向上を提案できたと考える．今後の課題として，Node.js のライブラリである mcl-wasm[16] を用いることで Python のプログラムを呼び出している時間を抑えることや，様々なペアリングライブラリを用いた実装をすることで実行時間の評価をすることが考えられる．

また，属性ベース署名において鍵の失効を行うことができるアルゴリズムの実装を行うことで，FIDO 認証においてアカウントリカバリーなどの問題に対処することも考えられる．そのため，様々な属性ベース署名のアルゴリズムを追求し，より利便性の向上した FIDO2 仕様の拡張ができるのではないかと考える．

参考文献

[1] FIDO Alliance: Spacifictions Overview, https://fidoalliance.org/specifications/. [最終閲覧日:2020-01-16].
[2] 飛天ジャパン 株式会社: 飛天ジャパン製品紹介, https://ftsafe.co.jp/products/fido/. [最終閲覧日:2020-01-18].
[3] yubico: yubico products, https://www.yubico.com/products/. [最終閲覧日:2020-01-16].
[4] GlobalPlatform: Trusted Execution Environment(TEE) Committee, https://globalplatform.org/technical-committees/trusted-execution-environment-tee-committee/. [最終閲覧日:2020-01-23].
[5] W3C: Web Authentication, https://www.w3.org/TR/webauthn/. [最終閲覧日:2020-01-16].
[6] CTAP: Client To Authnenticator Protocol(CTAP), https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html. [最終閲覧日:2020-01-16].
[7] ヤフー株式会社: Yahoo! JAPAN での生体認証の取り組み（FIDO2 サーバーの仕組みについて）, https://techblog.yahoo.co.jp/advent-calendar-2018/webauthn/. [最終閲覧日:2020-01-20].
[8] Maji, H. K., Prabhakaran, M. and Rosulek, M.: Attribute-Based Signatures, *Topics in Cryptology – CT-RSA 2011* (Kiayias, A., ed.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 376–392 (2011).
[9] 光成滋生: クラウドを支えるこれからの暗号技術，秀和システム (2019-04-04).
[10] 大森芳彦，西村豪生，山下高生: ID プロバイダと連携した端末所有者検証を基にした端末間権限共有と移譲 (ネットワークシステム)，電子情報通信学会技術研究報告= IEICE technical report : 信学技報， Vol. 117, No. 459, pp. 403–408（オンライン），入手先 ⟨https://ci.nii.ac.jp/naid/40021521981/⟩ (2018).
[11] FIDO Alliance: パスワードのいらない世界へ FIDO 認証の最新状況, https://www.slideshare.net/FIDOAlliance/fido-178936595. [最終閲覧日:2020-01-20].
[12] Johns Hopkins University ISI: Charm-Crypto 0.50 documentation, https://jhuisi.github.io/charm/. [最終閲覧日:2020-01-20].
[13] Mamietti: Mamietti/ABS, https://github.com/Mamietti/ABS. [最終閲覧日:2020-01-20].
[14] W3C: Packed Attestation Statement Format, https://www.w3.org/TR/webauthn/\#packed-attestation. [最終閲覧日:2020-01-25].
[15] apowers313: apowers313/fido2-server-demo, https://github.com/apowers313/fido2-server-demo. [最終閲覧日:2020-01-28].
[16] 光成滋生: mcl-wasm, https://www.npmjs.com/package/mcl-wasm. [最終閲覧日:2020-01-25].