2021年度　卒業論文

ProVerifを用いたBluetoothの安全性解析:Passkey Entryモードの検証に向けて

関東学院大学

理工学部理工学科

情報ネット・メディアコース

218K6023　河合　悠斗

指導教員　塚田恭章

2022年　3月

概要

　暗号プロトコルの自動検証ツールとしてProVerif[1]がある。本研究では、BluetoothのセキュアシンプルペアリングのYehら[2]のNumeric ComparisonモードについてProVerifを用いた安全性検証の再現実験を行った。検証の結果、先行研究[3]と同様の結果を得られた。さらに改良されたPasskey Entryモードの検証に向けて形式化する上での難しさや予想される攻撃法について考察した。

目次

第1章　はじめに 1

第2章　クレジットカード決済 2

2-1　クレジットカード決済の課題 2

2-2　クレジットカード決済における認証方式 2

2-3　検証プロトコルの動作 2

2-4　類似プロトコルとの違い 3

第3章　プロトコルのモデル化 4

　3-1　メインプロセス 6

3-2　各プロセス 9

第4章　検証結果 13

第5章　プロトコル記述の反省・改善点 14

第6章　ProVerifの注意点 15

第7章　まとめ 16

謝辞 17

参考文献 18

第1章　はじめに

近年、インターネットなど技術の発達により、セキュリティの重要性が日に日に増してきている。また、プロトコルの設計などの複雑なシステムの開発において、高度なセキュリティを保証する技術の確立が求められてきている。そこでProVerif[1]などの自動形式検証ツールを用いたプロトコルの安全性の検証が行われている。

近年，電子決済サービスの実装プロトコルの不備を突く形で，攻撃者が顧客の資金を不正に盗む事件が相次いでいる．これらの事件では，運営者による問題点の総括は行われるものの，それらのサービスに用いられたプロトコルの全体像が公開されることはほぼない．その後，対策として「SMSを用いた二段階認証」などアウト・オブ・バンド方式認証がサービスに追加されることはある．しかし，ユーザに対し，それがどのようにして以前の脆弱性を塞ぎ，有効に働くかという点まで説明されることもほぼなく，またすぐさま理解できるユーザも少ないと思われる．そこで本研究では，前述のモデル検査器であるProVerifを用いて，電子決済サービスに用いられる，代表的な認証プロトコルを検証し，それが実際に安全と言えるかどうか，および，ProVerifが身近に使われる認証方式の検証に有効であるかどうかを検証する．検証ケースを代表的な電子決済サービスであるクレジットカード決済とし，そのプロトコルの重要な性質である認証性が保たれているか検証することを目的とした．

ProVerifとは，INRIAが開発した形式検証を行うツールであり，通信プロトコルの検証に特化したモデル検査器の一つと言える．ProVerifは，検証したいプロトコルの様々な通信パターンを仮想的かつ網羅的にシミュレートし，プロトコルに対する具体的な攻撃手法などを検出し提示することができる．文献 [2]では，電子投票プロトコルの匿名性(観測等価性)の検証が，当ツールによって行えることが確認されている．また文献[3]では，日本の代表的メッセージアプリである「LINE」の暗号化プロトコルに，前方秘匿性があるかどうかを検証し，プロトコルに対する攻撃方法があることを発見している．今回はこのProVerifを用いて，セキュリティの特性の一つである認証性について検証を行う．「認証性が保たれている」とは，サービスの利用者が，実は攻撃者がなりすました者ではなく，確実に正規の登録を済ませた利用者本人であることが確実に確認できる状態を指す．認証性が必要となるプロトコルは様々なものが存在するが，本研究ではクレジットカード決済を扱うこととする．なお，本研究に使用したProVerifのバージョンは2.00である．

　本研究ではまず，決済に用いられる認証プロトコルのモデル化を行った．今回扱うプロトコルは，物理型ないしソフトウェア型トークンを使用する「タイムシンクロナス認証方式」と呼ばれるものであり，ユーザID，トークン固有のシード値，現在時刻を元に，トークンを以て生成したトークンコードを利用した認証方式である．これを4つのステップから構成されるプロトコルとしてモデル化し，ProVerifのコードとして表現した．認証性を「最終的にプロトコルの実行が完了したならば，ユーザの正規のIDが送信されているはずである」と定式化し，ProVerifにクエリとして与えて実行した結果，認証性が保たれていることの検証に成功した．一方，プロトコルを正しくコード化できているかという  
点については課題が残った．また，コード化の正しさといった面から，その反  
省・改善点やProVerifを使用する際の注意点をまとめた．

　本研究の構成は以下の通りである．第二章ではクレジットカード決済，認証性，認証プロトコルの説明を行う．第三章ではプロトコルのモデル化について説明する．第四章では検証結果について説明する．第五章ではプロトコル記述の正しさについて述べる．第六章ではProVerifを使用する際の注意点をまとめ，第七章では本研究の総括を行い，今後の課題について述べる．

第2章　クレジットカード決済

**2-1 クレジットカード決済の課題**

　近年，電子商取引の流行により，それにクレジットカード決済が用いられることが多くなっている．クレジットカード決済は，迅速に支払いが済むことや，支払いのために銀行やコンビニなどの実店舗に行く必要がないなど，電子商取引と相性が良い．その反面認証性が不確かである，といった問題があるとされている．

例えば，悪意のある実店舗の店員が，わずかな時間でユーザから渡されたクレジットカードの裏表両面の認証情報を暗記してしまい，自分の電子商取引に悪用する事例が確認されている．

**2-2 クレジットカード決済における認証方式**

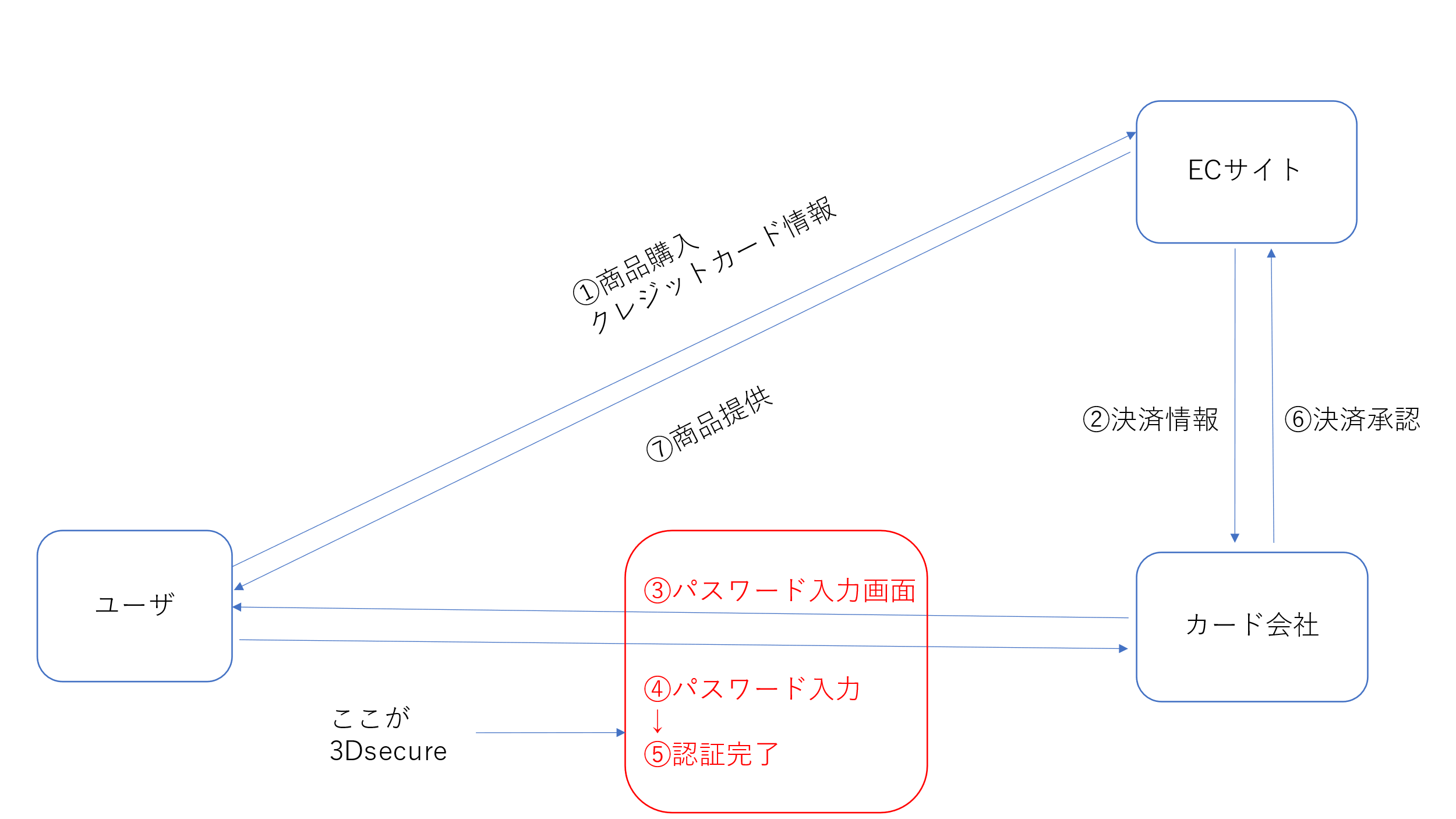
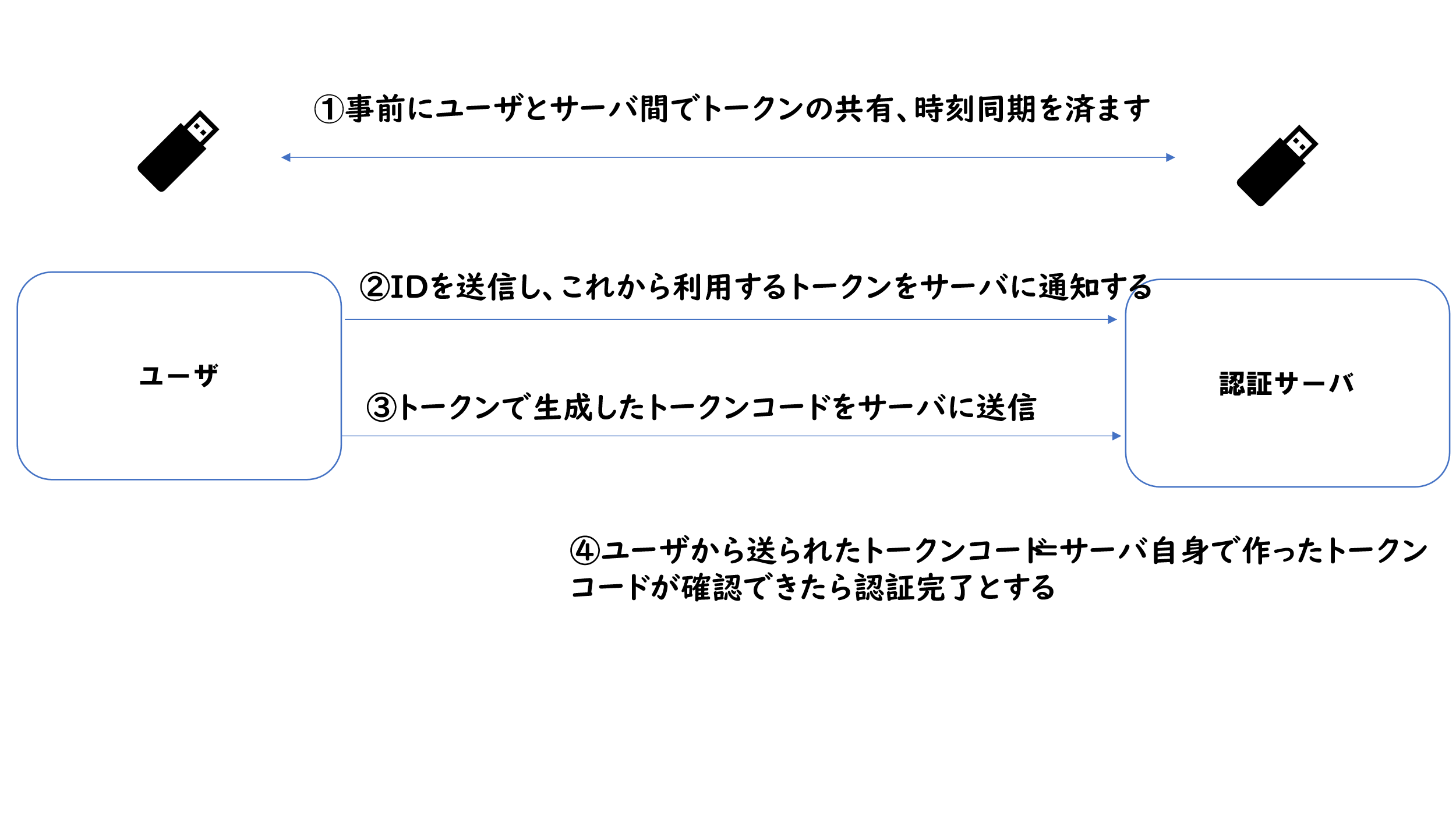


図1.クレジットカード決済のプロトコル図

　クレジットカード決済に用いられる代表的な認証方式として，「3Dsecure」があり，これは決済プロトコルの一部として機能する．この方式が機能する決済プロトコル内では，まずユーザから商品購入の依頼を受けた決済事業者が，ユーザから渡されたカード情報をカード会社に転送する．そして，カード会社はユーザがカードの持ち主であるか確認をするが，この部分が「3Dsecure」に該当する．本研究では，この「3Dsecure」に絞ってモデル化し，ProVerifのコードとして記述し，安全性の検証を行う．なお今回扱うプロトコルは，三井住友銀行カードなどで用いられる，物理型ないしソフトウェア型トークンを使用する「タイムシンクロナス認証方式」 [4]と呼ばれるものである．

**2-3 認証プロトコルの動作**



**図2.認証プロトコルの動作図**

　今回検証する認証プロトコルは，事前にトークンをユーザに渡し，サーバ側でもユーザのトークンと同等の動作をするものを所持しているという前提のものである．プロトコルは4つのステップに分けられる．

* ステップ1

ユーザとサーバ間でトークンの共有と，トークン内部における時刻の同期を済ませる．

* ステップ2

ユーザがサーバにIDを送信し，IDと紐付けられたトークンの使用を通知する．

* ステップ3

ユーザID，トークン固有のシード値，現在時刻を元に，トークンを以てトークンコードを生成し，サーバに送信する．トークンコードTは，関数fとIDとシード値および同期時刻tから

T = f(ID,s,t) (1)

によって計算される．

* ステップ4

サーバ側でもユーザからトークンコードが送られ次第，IDと紐付けられたトークンでトークンコードを生成し，それが送られたコードと一致するか確かめる．一致した場合，認証完了とする．

**2-4 類似プロトコルとの違い**

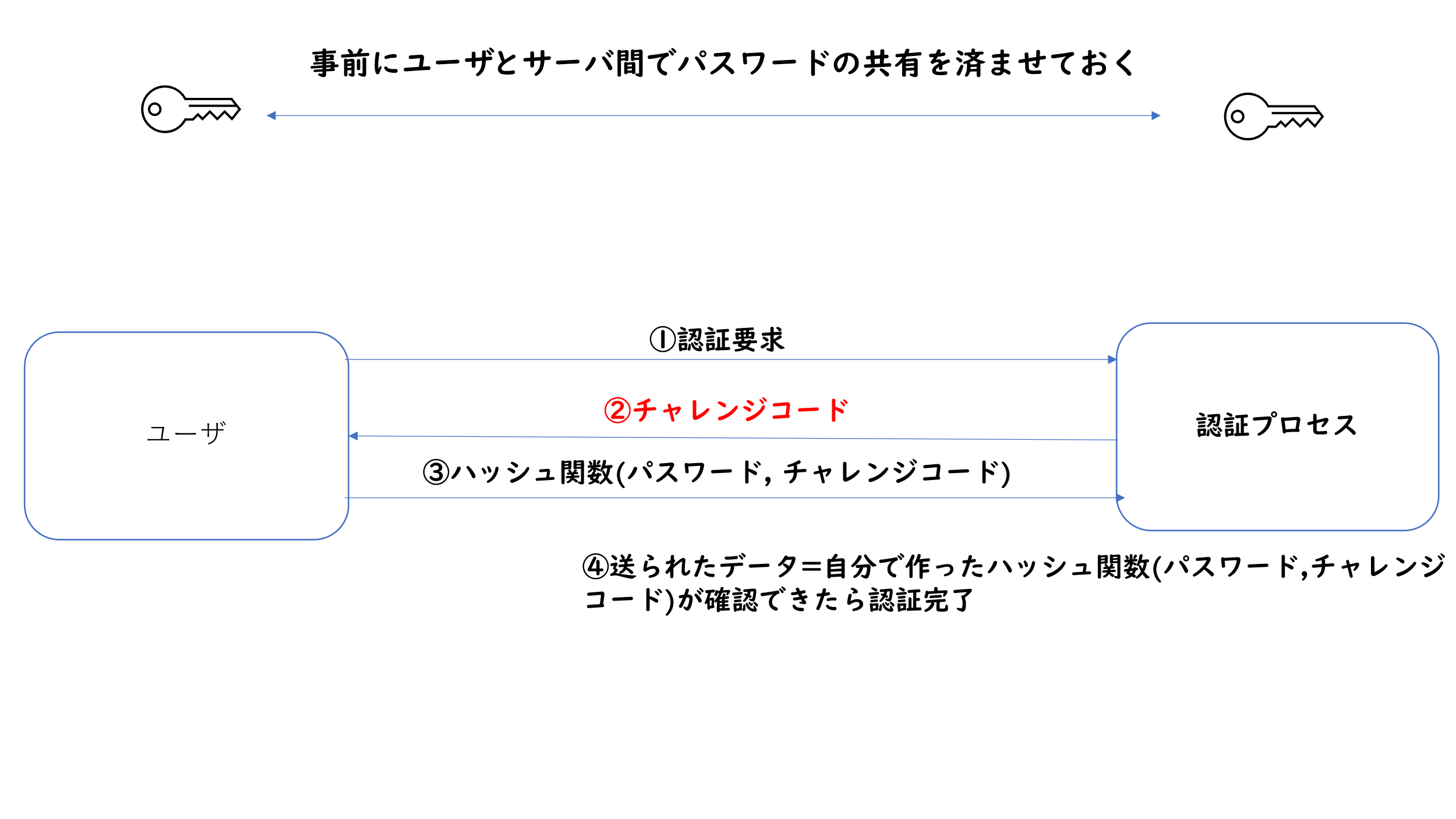


図3.CHAPの動作図

　今回検証するタイムシンクロナス型プロトコルと類似のプロトコルに，「チャレンジアンドレスポンス型認証方式」[5] がある．前者と後者の違いは，プロトコル開始後のサーバとの通信回数が，前者が一回分少ないことにある．その理由は，事前にトークンの共有を済ませてあるためで，実質的に図3の②にある「チャレンジコードの送信」をする必要がない(既に終えている)からである．

第3章　プロトコルのモデル化

　ProVerifにて認証プロトコルのモデル化を行った．本研究では認証性を検証するために，図2のプロトコルにおいて少なくとも満たされなければならない性質を考えた．それは「最終的に認証が完了したならば(図2の④)，ユーザの正規のIDが送信されている(図2の②)はずである」ということである．それをProVerifの式で仕様として書き表すと，以下のようになる．

query otp:bitstring; event(Success(otp)) ==> event(PROB1). (2)

　(2)でのSuccess(otp)は図2における④，PROB1は②の成立をそれぞれ示している．またこれと逆の性質である「最終的に認証は完了したにも関わらず，ユーザの正規のIDが送信されていない(不正なIDが送信された)」という仕様も以下のように書き表し，検証することができる．

query otp:bitstring; event(Success(otp)) ==> event(FAIL). (3)

(3)におけるFAILは，正規でないIDを送信した場合のイベントラベルを表している．この仕様は，当プロトコルにおいて，満たされないことが確認されなければならない性質である．

**3-1 開発環境**

本研究では，WSL(Windows Subsystem for Linux)と呼ばれる，Windows10に標準で付属するUNIX環境を土台として，その上にProVerifをインストールし開発環境とした．従来の仮想マシン上のUNIX環境と比較すると，動作がやや軽く，インストールの手順が簡略化され，ある程度PCの知識のある者であれば，ProVerifの環境構築を容易に行える点で優位である．

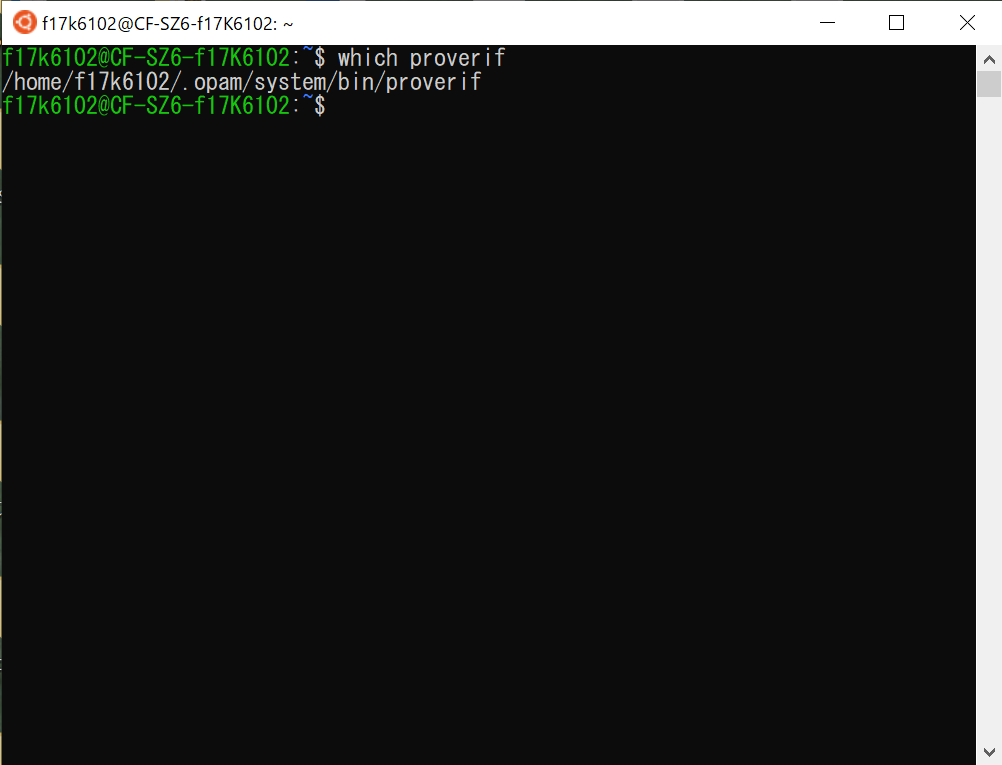
****

図4.ProVerifをインストールしたWSL

**3-2 メインプロセス**

　今回，図2の①の部分に相当するメインプロセス部分のソースコードを示す．

1. process
2. new time:Time; (\* 同期時刻の生成 \*)
3. User(id,seed,time) | !AuthServer(id,seed,time)

　このメインプロセスでは各プロセス(今回はユーザと認証サーバ)に必要なものを作成し引数としている．2行目ではユーザと認証サーバで同期する時刻を生成している．3行目では事前に生成したユーザのIDとトークンのシード値を，ユーザと認証サーバそれぞれへ配布している．

**3-3 各プロセス**

　次にユーザ，認証サーバの各プロセスを説明する．以下に各プロセスのソースコードを示す．

1. (\* ユーザ \*)
2. let User (id:ID, s:bitstring, syntime:Time) =
3. if id <> ngid then
4. out(ch, id);
5. event PROB1;
6. out(ch, genCodebyToken(id, s, syntime))
7. else
8. let id = ngid in
9. out(ch, ngid);
10. event FAIL.

12.

13. (\* 認証サーバ \*)

14. let AuthServer (PID:ID, s:bitstring, syntime:Time) =

15.

16. in (ch, (userid:ID));

17. if (userid = PID) then event ACC;

18. in(ch, otp:bitstring);

19. if otp = genCodebyToken(PID, s, syntime) then

20. event Success(otp).

ユーザのプロセスから説明する．2~12行目では，ユーザの動作を規定している．2行目でユーザがID，トークン固有のシード値，同期時刻を持つことを規定している．3行目でそのIDが間違っていないこと，不正なIDでないことを確認し，4行目でサーバに送信している．この動作を「PROB1」としてラベル付けし，図2における②としている．7行目では，サーバに(1)で規定したトークンコードTを作成し送信している．9行目からはIDが間違っていること，不正なものである場合を規定している．10行目で，正規でないIDをサーバに送信している．これでは当然認証は進まないことを想定しているので，この動作を「FAIL」とラベル付けしている．これでユーザのプロセスは終了する．

　次にサーバのプロセスを説明する．14~20行目では，サーバの動作を規定している．14行目でサーバもユーザと同様に，ID，トークン固有のシード値，同期時刻を持つことを規定している．16行目で，ユーザから送信されたIDを受信している．17行目で，その受信したIDが，事前にサーバとユーザで共有していたIDと一致しているかチェックしている．18行目でユーザから送信されたトークンコードTを受信し，19行目でサーバ自身で計算したトークンコードと一致するか確認している．一致した場合，20行目でイベント「Success(otp)」としてラベル付けし，図2における④としている．これでサーバのプロセスを終了する．

**3-4 ソースコードの全体像**

以下にソースコードの全体像を示す．

(\* 時刻同期型認証のモデル化 \*)

1. free ch:channel.

2.

3. type ID.

4. type Time.

5.

6. free id:ID [private].

7. free seed:bitstring [private].

8. free time:Time [private].

9.

10. free ngid:ID.

11.

12. fun genCodebyToken(ID, bitstring, Time): bitstring[private]. (\* トークンコード生成器 \*)

13.

14. event PROB1.

15. event PROB2.

16. event ACC.

17. event Success(bitstring).

18. event FAIL.

2. (\* ユーザ \*)
3. let User (id:ID, s:bitstring, syntime:Time) =
4. if id <> ngid then
5. out(ch, id);
6. event PROB1;
7. out(ch, genCodebyToken(id, s, syntime))
8. else
9. let id = ngid in
10. out(ch, ngid);
11. event FAIL.

31.

32. (\* 認証サーバ \*)

33. let AuthServer (PID:ID, s:bitstring, syntime:Time) =

34.

35. in (ch, (userid:ID));

36. if (userid = PID) then event ACC;

37. in(ch, otp:bitstring);

38. if otp = genCodebyToken(PID, s, syntime) then

39. event Success(otp).

40. process

41. new time:Time; (\* 同期時刻の生成 \*)

42. User(id,seed,time) | !AuthServer(id,seed,time)

第4章　検証結果

　上記のコードの検証を行い，認証性が保たれていることが証明できた．以下に検証結果を示す．

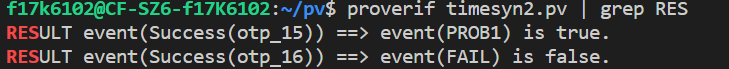


図4.ケース1検証結果

　図2のように(2)は成り立つが，(3)は成り立たないことが示された．これで当初予定していたプロトコルの性質が確認された．

第5章　プロトコル記述の反省・改善点

　ProVerifはOCamlというプログラミング言語を元に作られており，関東学院大学で学んだJava言語のように，オブジェクト指向型プログラミングには向いていない．そのため，フローチャートからプログラムを書き上げるといった定石がなく，習得するのが困難であると感じた．今回のプロトコルは比較的単純であったために，検証結果もイメージしやすいものであった．しかし，プロトコルをProVerifのコードとして記述した際に，それが正常に動作しているか評価する方法は基本的にない．そのため，今後複雑なプロトコルを検証する場合，検証結果が真であってもあまり信用が出来ない結果となる可能性がある．なので，コードを記述する際の手順をマニュアル化することや，検証結果を併用・比較し，同じプロトコルで他の性質を検証するということが必要だと感じた．

第6章　ProVerifの注意点

　今回，ProVerifにて検証を行ったが，以下のように注意が必要な部分があった．

* + コードを書き上げるための定石がないため，

記述者によってバラツキが出やすい．

* + コードが長くなりがちで，記述にミスが出やすい．また記述ミス時に出るエラーがあいまいで，どこを修正すればよいのかわかりにくい．
  + ｢安全である｣という結果が出力された場合でも，  
    入力したプロトコルにミスがあれば，結果は信用できない．
  + 仕様を細かく記述できるが，  
    記述者によってバラつきが出やすい．
  + 様々な結果と比較を行わないと，

プロトコルが正しく動作しているかわかりづらい．

　これらの注意点があるが，以下の方法で改善ができると考えられる．

* 結果の検証・考察を行う．
* 複数の性質を検証し，併用・比較を行う．
* プロトコル記述方法・検証手順のマニュアル化を行う．

　まだこれらの方法は実現できていないため，今後はProVerifのプロトコルの記述の正しさについての改善策の検証も行っていきたい．

第7章　まとめ

　クレジットカード決済に用いられる認証プロトコルのモデル化を行い，ProVerifにて検証を行った．その結果，最低限満たされなければいけない認証性が保たれることが証明された．よって，ProVerifは身近に用いられるプロトコルの認証性の検証，および動作の理解に有用であることがわかった．一方で，プロトコルのコード化の正しさの評価が課題として残った．

今後は今回証明したログイン部の検証に加え，よりプロトコルの実態に近いユーザとECサイト間の動作も含めたプロトコルをモデル化し，検証することを目標としたい．

謝辞

　本研究を進めるにあたり，ご指導，ご助力を頂いた指導教員の塚田恭章教授には深く感謝いたします．また，ProVerifのインストール環境の構築を手伝ってくださった先生方，ご協力して頂いたネットワークセキュリティ研究室の皆様にも深く感謝いたします．ここで深謝の意を表し，謝辞とさせて頂きます．

参考文献

[1] Bruno Blanchet：“ProVerif”,  
<https://prosecco.gforge.inria.fr/personal/bblanche/proverif/>

[2] 松山健人: “ProVerifを用いた電子投票プロトコルの匿名性の検証”, 関東学院大学 理工学部 卒業論文 (2020)

[3] Cheng Shi, Kazuki Yoneyama: “Verification of LINE Encryption Version 1.0 Using ProVerif”, IWSEC, Advances in Information and Computer Security,pp 158-173 (2018)

[4] ワンタイムパスワードサービス「One Time Pass」アプリ,  
https://www.smbc-card.com/mem/service/sec/otp.jsp

[5] 岡崎 裕之, 紫村 彰吾, 宮本 樹, 渡邊 樹, 布田 裕一, 村上 恭通： “形式的安全性検証ツールを用いた暗号教育の実践とそのe-Learning教材化の課題について”, コンピュータソフトウェア, Vol.37, No.1, pp.99-113 (2020)