Verification of Integrity and Secrecy Properties of a Biometric Authentication Protocol

A. Salaiwarakul and M.D.Ryan

School of Computer Science,
University of Birmingham, UK
{A.Salaiwarakul, M.D.Ryan}@cs.bham.ac.uk

Abstract. In this paper, we clarify and verify an established biometric authentication protocol. The selected protocol is intended to have three properties: effectiveness (integrity checks are carried out on all hardware before enabling transmission of biometric data), correctness (the user is satisfied that integrity checks have been executed correctly before transmission of biometric data occurs), and secrecy (unauthorized users cannot obtain biometric data by intercepting messages between the system's hardware components). We analyse the clarified protocol using applied pi calculus and the ProVerif tool, and demonstrate that it satisfies the intended properties of the protocol. Moreover, this paper shows that the verification result between the naive interpretation and the clarified interpretation is different.

1 Introduction

1.1 Biometric Authentication Protocols

Biometric authentication complements other methods of authentication such as passwords or smartcards. It may be used as an alternative to these, or in combination. Passwords used on their own are known to have certain weaknesses: users are liable to choose easily guessable passwords, transfer passwords between each other in ways not desired by the system owners, use the same password on multiple systems, or forget their passwords. Authentication using smart cards also has some of these weaknesses, such as undesired transfer between, or theft from, users. Biometric user authentication can be utilised in a variety of applications, from logging into a local PC, to passenger identification at a border control, to authentication on a remote server in e-commerce transactions or online banking.

Potential biometric techniques include fingerprint and hand geometry, as well as voice, retina, face and behavioural characteristics. The rapid move towards the use of biometrics in user authentication comes from the method's promise to offer secure and reliable authentication. In well-designed and engineered systems for biometric authentication, user A cannot authenticate as another user B, even with B's cooperation. In contrast to other types of credential, such as password or smartcard, which can be transferred or stolen, biometric authentication promises a ”non-transferability” property, which means that users cannot lose their credentials or acquire those belonging to others.

However, there are many obstacles to overcome before this potential can be realised. Biometric authentication depends on biometric protocols, i.e. the way biometric data is transmitted and stored. But protocol design is known to be very difficult. The well-known paper that identified attacks on the Needham-Schroeder protocol that was believed to be invulnerable and had been used successfully for more than a decade [1]is one example of how accurate protocol verification is crucial. The focus of this paper concerns the handling and storage of biometric data. Biometric data cannot be considered a secret in the way that private keys or passwords can. In contrast with private keys, biometric data is given to potentially hostile hosts when a user wishes to authenticate herself, and unlike passwords, biometric data cannot be replaced - a user cannot conveniently choose different biometric data to present to different hosts in the way that one might use a different (and lower security) password for a webmail account as for a bank account. Moreover, in contrast with keys and passwords, biometric data such as the user's facial characteristics and fingerprints should be considered to be in the public domain, and can be captured without the user's consent or knowledge.

In spite of this, we take the view that biometric data should be kept private as a matter of good practice. In this respect, it is rather like credit card numbers; they are not really private, as we voluntarily cite them on the phone and by unencrypted email and allow restaurant and other retail staff to handle the cards, often in our absence. Nevertheless, it seems sensible not to allow such data to be spread around without restriction. The same idea applies to biometric data; even if a user's biometric data could be captured by agents having access to smooth surfaces the user touches, or agents to whom the user authenticates, it should not be made unnecessarily easy for malicious agents to acquire it. However, biometric authentication methods cannot assume that biometric data is secret. Such an assumption is false.

$^{+}$**Long version.** The full paper that presents the ProVerif model is available at http://www.cs.bham.ac.uk/˜mdr/research/papers/

1.2 Our Contribution

To demonstrate verification of biometric authentication protocols, we use an established protocol, CPV02 [2], as a case study, and prove whether its intended properties are satisfied.

In order to verify the properties of the biometric authentication protocol, we need to clarify the operation of the protocol. We show that it is easy to interpret the protocol incorrectly, and that this would affect the security properties. An example of a naive interpretation and its verification result is given in a later section.

We set out and formalise the intended properties of the protocol. We present the verification result of the naive interpretation, and clarification of the protocol's detail that is necessary in order to achieve successful verification. Moreover, we show that the verification outcome of the naive interpretation of the protocol identifies an attack while the result of the clarified one is different.

To obtain our findings, we use the verification tool ProVerif [5]. We explain the protocol, clarify it, and provide a formal model of the protocol and its properties. We then give the outcome of the verification, and some analysis.

2 The CPV02 Protocol

In [2], Chen, Pearson and Vamvakas present a protocol for biometric authentication that we call CPV02. This protocol prevents disclosure of biometric data both during data transmission and within all system hardware. This is achieved through integrity metric checking. The protocol is a generic protocol for biometric authentication. It can be used as a protocol in applications that require authentication before the user is allowed to proceed.

The system under consideration is composed of three connected components: a smartcard (SC), a trusted computing platform (TCP) and a trusted biometric reader (TBR).

The SC is used for storing credential information such as the user's biometric code or the user's signature. The TBR is a device for reading the user's biometric data for use later in the matching process. In this protocol, the TBR and the SC generate session keys to transfer the user's submitted biometric data (BD) and the user's stored biometric code (BC).

A TCP is a device that behaves in an expected manner for the intended purpose and is resistant to attacks by application software or viruses [3]. This is achieved because the TCP contains a Trusted Platform Module (TPM), which stores keys and can perform cryptographic operations. The TPM can check the integrity of the TCP. Specifically, it can create an unforgeable summary of the software on the TCP, allowing a third party to verify that the software has not been compromised. This can be accomplished by presenting a certificate to the third party to confirm that it is communicating with a valid TPM. Table 1 summarises notations and meanings that will be used through out the paper. Figure 1 shows the basic system for this model. Informally, it can be

Table 1. Notations and Meanings

Notation	Meaning
BC	User's stored biometric code
BD	User's submitted biometric data
TPM	Trusted Platform Module
TCP	Trusted Computing Platform
TBR	Trusted Biometric Reader

described as a user holding a smart card that contains her previously stored biometric code, e.g. fingerprint code. To authenticate herself to the system, she first inserts the smart card into a smart card reader. This triggers part of the protocol during which the integrity of the computing platform and the biometric reader are checked and the result is returned to the smart card. If the smart card is satisfied that the computing platform and biometric reader have not been tampered with, it indicates this to the user, e.g. by releasing a special image to be displayed by the computing platform. The user recognises that image as an indication that the integrity checks have been successful and proceeds to the second step, which is biometric authentication. To achieve that, she submits her biometric data, e.g. by placing her fingerprint on a biometric reader.

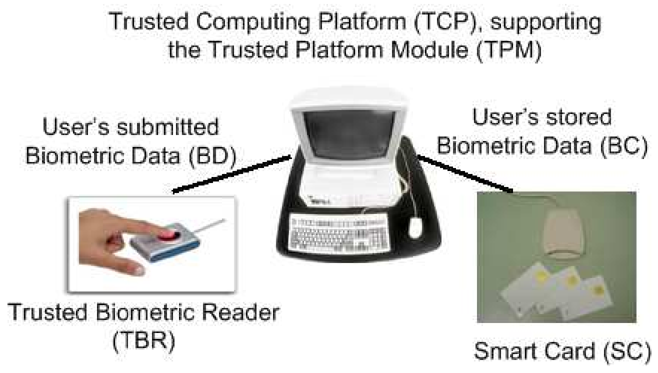
Trusted Computing Platform (TCP), supporting the Trusted Platform Module (TPM)
User's submitted Biometric Data (BD)
User's stored Biometric Data (BC)
Trusted Biometric Reader (TBR)
Smart Card (SC)

Fig. 1. The basic setup for CPV02 consists of a trusted biometric reader (TBR), a trusted computing platform (TCP) that supports a trusted platform module(TPM), and a smart card device (SC)

The biometric code stored on the smart card and the submitted biometric data from the biometric reader are then sent to the computing platform, which will validate whether they match. If they match, the smart card will release the user's credential data, e.g. her signature on a message, to the computing platform.

The BC is stored in the SC and will be transferred to the TPM for comparison with the BC. However, before this transmission is performed, the TPM and the SC must authenticate each other by sending an authentication message, which includes a nonce and integrity metric. The integrity metric is a measurement of the trustworthiness of the component. Depending on its policy, the challenger will decide, based on this value, whether to trust or allow any action to be performed.

The SC sends a nonce n1 and its identity to the TPM. The TPM generates a nonce n2 and a message including n1, n2, the identity of the SC and integrity metric D3. The integrity metric D3 is used to trigger the components involved in the communication to do the integrity checking. The message sent back to the SC will be signed by the TPM so that the SC can check its origin and the correctness of n1. After the authentication's success, the SC generates the session key SK1, shared by the SC and the TPM, for encrypting the BC, before sending it together with the authentication messages. After the TPM has verified the message, it then stores the BC.

When the TBR is presented to the system, it also performs mutual authentication with the TPM and generates a session key to share between the TBR and the TPM. In the same way as that in which the TPM and the SC authenticated each other, the TBR sends an integrity metric D7 to the TPM. If the TPM has successfully verified the message it receives, it will send back a message MF5. The TBR verifies the message. After the authentication has succeeded, the TBR generates a session key SK2, shared by the TBR and the TPM, for use in encrypting the BD from the TBR to the TPM.

The BD is encrypted by using the session key created in the previous stage to the TPM. This data will be compared with the BC. After the message is verified, the TPM decrypts the encrypted message and verifies the validity of the BD. If they match, the user is allowed to use the system or perform the request. For example, the SC releases the user's signature. The message sequence of this protocol is shown in Figure 2.

2.1 Intended Properties of CPV02 Protocol

The protocol has the following intended properties:

1. *Effectiveness.* The accessed computing platform is given neither the user's stored biometric code nor the user's submitted biometric data until the integrity of both the computing platform and biometric reader are checked by the smart card.
2. *Correctness.* The biometric reader is not given the user's submitted biometric data until the user is convinced of the correctness of both the computing platform and biometric reader integrity checking.
3. *Secrecy.* An unauthorised entity that can listen to a message between the smart card and computing platform, or between the biometric reader and computing platform, cannot obtain either the user's stored biometric code or the user's submitted biometric data.

2.2 Problem Encountered

To verify the three protocol properties presented in 2.1, we need to gain a detailed understanding of how the protocol works and the sequence of messages.

If a naive verifier were to interpret the CPV02 protocol as it is presented in [2] (page 7-9), it would identify an attack. The sub-protocols are presented in a sequence order, and since nothing is said about the order in which they should be run, the reader can assume they are run in the order presented. We performed the verification on that assumption. The result of the verification shows that one of the properties does not hold: the biometric data is released before the TPM is checked.

The ProVerif model of this interpretation is shown in the full paper$^+$. Let us briefly describe this model. The code consists of 4 processes (excluding the main process): TPM, TBR, SC and ProcessK. Processes TPM, TBR and SC perform the operations of the TPM, TBR and SC respectively (as mentioned in section 2). ProcessK distributes the verification key certificates to the three processes TPM, TBR and SC. The main process generates private keys for each component and distributes them via private channels, running these processes concurrently.

The result of the verification shows that the TCP is sent the BC before the TBR is checked. This breaks one of the intended properties of the protocol: *effectiveness*.

2.3 The Clarified CPV02 Protocol

Email discussion with one of the authors of [2], Liqun Chen, has given us further vital information about CPV02. We have learnt that the four sub-protocols can run at any time and in any order. Moreover, the result from one sub-protocol may affect the other sub-protocols. For example, sub-protocol (S1) cannot be run successfully without also running sub-protocol (S2). These facts cannot be easily extracted from the paper without the discussion and they are important in order to successfully verify the protocol.

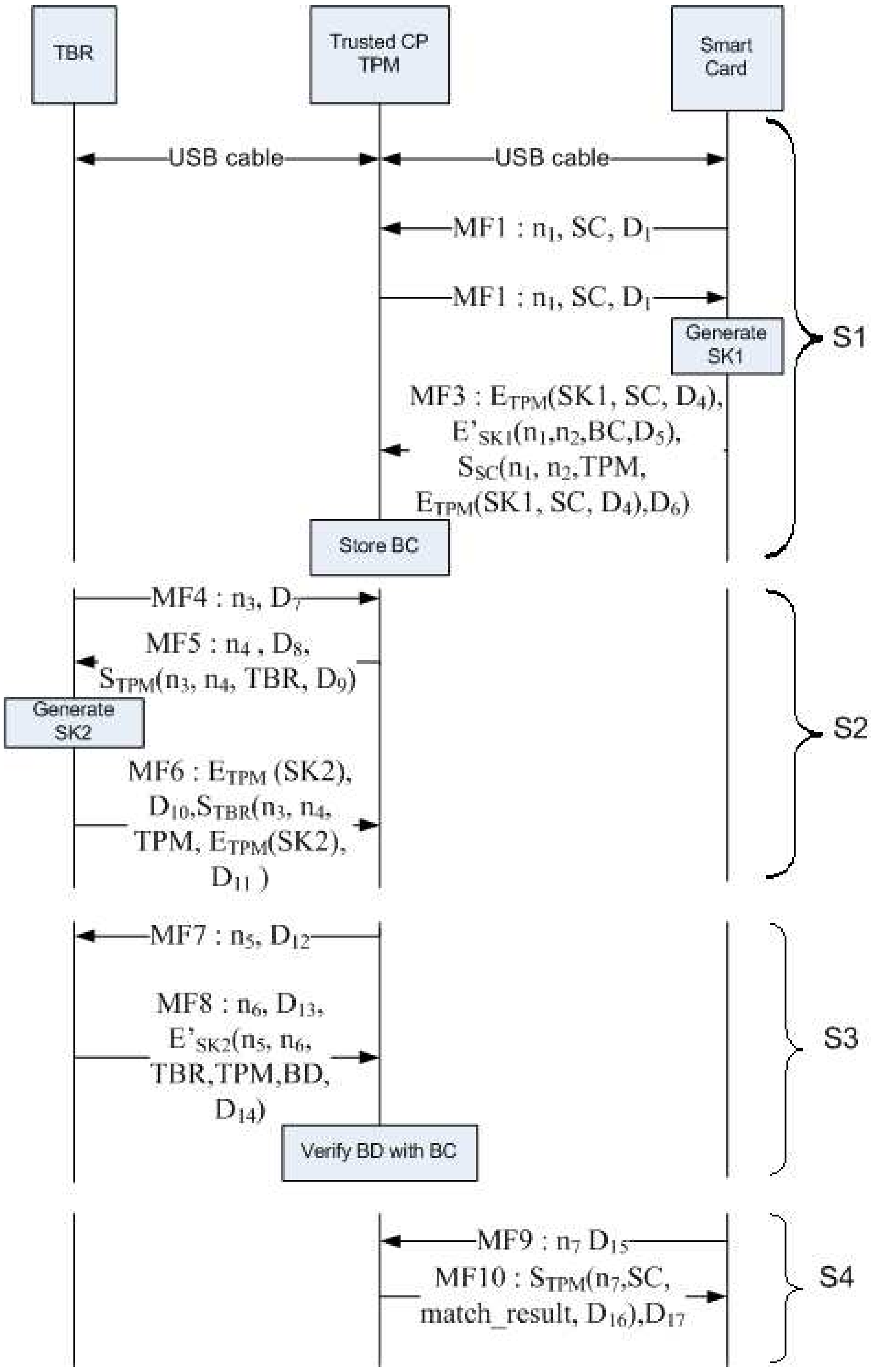
TBR
Trusted CP TPM
Smart Card
USB cable
USB cable
MF1 : n_1, SC, D_1
MF1 : n_1, SC, D_1
Generate SK1
MF3 : E_{TPM}(SK1, SC, D_4), E'_{SK1}(n_1,n_2,BC,D_5), S_{SC}(n_1, n_2,TPM, E_{TPM}(SK1, SC, D_4),D_6)
Store BC
S1
MF4 : n_3, D_7
MF5 : n_4 , D_8, S_{TPM}(n_3, n_4, TBR, D_9)
Generate SK2
MF6 : E_{TPM} (SK2), D_{10},S_{TBR}(n_3, n_4, TPM, E_{TPM}(SK2), D_{11})
S2
MF7 : n_5, D_{12}
MF8 : n_6, D_{13}, E'_{SK2}(n_5, n_6, TBR,TPM,BD, D_{14})
Verify BD with BC
S3
MF9 : n_7 D_{15}
MF10 : S_{TPM}(n_7,SC, match_result, D_{16}),D_{17}
S4

Fig. 2. Message Sequence Chart for CPV02 Protocol

Let us consider the message sequence chart of CPV02 in Figure 2. The protocol consists of four sub-protocols (S1), (S2), (S3), and (S4) which can run in any order and at any time. In (S1), the encrypted BC is sent from the SC to the TPM. In (S2), a session key is created for use between the TPM and the TBR when the BD is encrypted. In (S3), the encrypted BD is sent from the TBR to the TPM. In (S4), a matching result on the BC and BD is sent from the TPM to the SC.

The detailed ProVerif model for verifying the properties according to the clarified protocol is presented in section 4.

3 Applied Pi Calculus and ProVerif

3.1 Applied Pi Calculus

Applied pi calculus is a language for describing concurrent processes and their interactions [4]. It is based on pi calculus, but is intended to be less pure and therefore more convenient to use. Properties of processes described in applied pi calculus can be proved by employing either manual techniques or automated tools such as ProVerif [5]. As well as reachability properties that are typical of model-checking tools, ProVerif can in some cases prove that processes are observationally equivalent [6].

To describe processes in applied pi calculus, one starts with a set of names (which are used to name communication channels or other constants), a set of variables, and a set of function symbol which will be used to define terms. In the case of security protocols, typical function symbols will include **enc** for encryption (which takes plaintext x and a key k, and returns the corresponding cipher text) and **dec** for decryption (which takes cipher text and a key k and returns the plaintext x). One can also describe equations which hold on terms constructed from the function. For example:

$$\mathbf{dec}(\mathbf{enc}(x,k),k) = x$$

Terms are defined as names, variables, and function symbols applied to other terms. Terms and function symbols are sorted, and of course function symbol application must respect sorts and arities. In the applied pi calculus, one has (plain) proceses and extended processes. Plain processes are built up in a similar way to processes in the pi calculus, except that messages can contain terms (rather than just names) [4, 7].

3.2 ProVerif

ProVerif is a protocol verifier developed by Bruno Blanchet [8]. This tool has been used to prove the security properties of various protocols [7, 9]. It can be used to prove secrecy, authenticity and strong secrecy properties of cryptographic protocols. It can handle an unbounded number of sessions of the protocol and an unbounded message space. The grammar of processes accepted by ProVerif is described in the long version of the paper.

In order to verify properties of a protocol, query commands may be made. The query 'attacker: m' is satisfied if an attacker may obtain the message m by observing the messages on public channels and by applying functions to them. The query ev :

$f(x_1,\ldots,x_n) \Rightarrow ev : f'(y_1,\ldots,y_m)$ is satisfied if the event $f'(y_1,\ldots,y_m)$ must have been executed before any occurrence of the event $f(x_1,\ldots,x_n)$.

An advantage of using ProVerif as a verfier is it models an attacker which is compliant with the Dolev-Yao model [10] automatically. We do not need to explicitly model the attacker.

4 Modelling the Clarified CPV02 in ProVerif

Now we model the CPV02 protocol based on the derived message sequence chart (shown in Figure 2) from clarification and the following assumptions:

1. All the components, TPM, SC and TBR, hold the public key of certificate authority.
2. The integrity metric measurements have been made and are stored in the tamper-resistant storage. Therefore we model it, as it is a stored secret value, and verify its correctness with the challenger's stored value.

The ProVerif code consists of signature and equational theory, a main process, a process for certificate distribution, S1 process, S2 process, S3 process, and S4 process. A detailed description of each process will be given in a later section.

4.1 Signature and Equational Theory

Our ProVerif model involves public key and host functions. We model cryptographic function as *enc* and *dec*. Similarly, the symmetric cryptography is modelled as *senc* and *sdec*. In order to introduce digital signature, function *sign* is added and function *checksign* is used to verify the origin of messages.

The public key cryptography is represented in the first equation. To decrypt messages from symmetric cryptography, the second equation permits us to do so. In the interest of verifying the origin of messages; the checksign equation is introduced in our model.

```
equation dec(enc(x,pk(y)),y) = x.
equation sdec(senc(x,k),k) = x.
equation checksign(sign(x,y),pk(y)) = x.
```

4.2 Main Process

In the main process, the public keys, private keys, and the identities of each component are created and distributed in the public channel. Moreover, the components can run at any time and in any order.

```
process

(* create secret keys *)
   new skCA;
   new skSC;
   new skTPM;
   new skTBR;
(* public keys *)
   let pkCA = pk(skCA) in
   out(ch,pkCA);
   let hostSC = host(skSC) in
   let hostTPM = host(skTPM) in
   let hostTBR = host(SKTBR) in
   out(ch,hostSC);
   out(ch,hostTPM);
   out(ch,hostTBR);
   !(TPM1) | !(TPM2) | !(TPM3) | !(TPM4)|
   !(SC1) | !(SC4) |
   !(TBR2) | !(TBR3) |
   !(processK)
```

4.3 Certificate Distribution

This process is intended to distribute the certificates of verification keys for the integrity checking process and distribute them through the private channel to guarantee that each identity will obtain them correctly.

```
let processK =
out(privChCertTPM1,sign((host(skTPM),pk(skTPM)),skCA))|
out(privChCertTPM2,sign((host(skTPM),pk(skTPM)),skCA))|
out(privChCertTPM3,sign((host(skTPM),pk(skTPM)),skCA))|
out(privChCertTPM4,sign((host(skTPM),pk(skTPM)),skCA))|
out(privChCertSC1,sign((host(skSC),pk(skSC)),skCA))|
out(privChCertSC4,sign((host(skSC),pk(skSC)),skCA))|
out(privChCertTBR2,sign((host(skTBR),pk(skTBR)),skCA)).
```

4.4 (S1) Sending the Encrypted Biometric Code

This sub-protocol includes two processes: TPM1 and SC1. The mutual authentication between the TPM and the SC is performed before the encrypted BC is transmitted. Firstly, the TPM and the SC obtain their certificates. The TPM generates a fresh random nonce. Then it sends its integrity metric with this nonce to the SC. The SC checks the certificate it receives from the TPM and retrieves the public key of the TPM. The SC verifies the validity of messages and generates a session key and then sends the BC encrypted by the key to the TPM. The TPM verifies the accuracy of the received message, decrypts it, and stores the BC in its secure storage. Moreover, from email discussion, we have learnt that (S1) cannot run successfully before (S2) has run. So we add state checking to check that (S2) has run.

```
let TPM1 =
(* Receive certificate of the verification keys for
    the integrity checking purpose *)

    in(pState2,P2);
    if P2 <> success then 0
    else
    (
    in(privChCertTPM1,D2);
    in(ch1,(nx1,hostSCx,=D1));
    new n2;
    out(ch2,(n2,sign((nx1,n2,hostSCx,D3),skTPM),D2));
    in(ch3,(m2,m3,m4));
    let(SK1Received,=hostSCx,=D4) = dec(m2,skTPM) in
    let(=nx1,=n2,BCReceived,Dx5) = sdec(m3,SK1Received) in
    let(=hostSCx,pkSCx) = checksign(Dx5,pkCA) in
    let(=nx1,=n2,=hostTPM,m5,=D6) = checksign(m4,pkSCx) in
    let(=SK1Received,=hostSCx,=D4) = dec(m5,skTPM) in
    event tpmgetBC()
    ).

let SC1 =
(* Receive certificate of the verification keys for
   the integrity checking purpose *)
   in(privChCertSC1,D5);
   new n1;
   out(ch1,(n1,hostSC,D1));
   in(ch2,(nx2,m1,Dx2));
   let(hostTPMx,pkTPMx) = checksign(Dx2,pkCA) in
   let(=n1,=nx2,=hostSC,imtpmReceived) = checksign(m1,pkTPMx) in
   if imtpmReceived <> D3 then 0
   else
  (
   event tpmChecked();
   new SK1;
   new BC;
   out(ch3,(enc((SK1,hostSC,D4),pkTPMx),senc((n1,nx2,BC,D5),SK1),
       sign((n1,nx2,hostTPMx,enc((SK1,hostSC,D4),pkTPMx),D6),skSC)));
   out(pState1,success)
  ).
```

4.5 (S2) Creating a Session Key for Encrypting the User’s Submitted Biometric Data

This sub-protocol represents mutual authentication between the TPM and the TBR. It also creates a session key for sharing between the TPM and the TBR. This sub-protocol runs when a TBR has been introduced to the system.

This sub-protocol includes the two processes TPM2 and TBR2. The certificates are obtained via the private channels. Note that the TPM has already obtained this certificate in the previous sub-protocol. TPM1 and TPM2 are indeed the same trusted platform modules but they are run in different sub-protocols and therefore require distinct names. So we model TPM2 to receive the certificate again but the certificate it receives is the same certificate as that received by TPM1.

The TBR has to authenticate itself to the TPM using an integrity checking mechanism. It creates a fresh random number and sends it with its integrity metric. If the TPM is satisfied with the checking result, it will send its certificate along with the authentication message to the TBR. The TBR retrieves the public key of the TPM. It then checks the correctness of the message. If it is valid, the TBR will create a session key SK2 for the encryption and decryption of the BD.

While the processes TPM1, TPM2, TPM3, and TPM4 are on the same trusted platform module, as seen in section 2, in order to fit the CPV02 protocol they need to run as separate sub-protocols. This fact also applies to TBR2 and TBR3. All variables created or received in one TPM process should be known to others. Hence, in the process TPM2, two private channels are set up. One is used for acknowledging that S2 has run and the other is used for transferring the session key SK2 from the process TPM2 to the process TPM3. Similarly, a private channel is set up in process TBR2 to transmit the session key from the process TBR2 to the process TBR3.

```
let TPM2 =
   in(privChCertTPM2,D8);
   in(ch4,(nx3,Dx7));
   let(hostTBRx,pkTBRx) = checksign(Dx7,pkCA) in
   if hostTBRx = hostTBR then
   event tbrChecked();
   new n4;
   out(ch5,(n4,D8,sign((nx3,n4,hostTBRx,D9),skTPM)));
   in(ch6,(m7,=D10,m8));
   let(SK2) = dec(m7,skTPM) in
   let(=nx3,=n4,=hostTPM,m9,=D11) = checksign(m8,pkTBRx) in
   let(=SK2) = dec(m9,skTPM) in
   out(pState2,success);
   out(privSK2TPM2,SK2).

let TBR2 =
(* Receive certificate of the verification keys for
   the integrity checking purpose*)
   in(privChCertTBR2,D7);
   new n3;
   out(ch4,(n3,D7));
   in(ch5,(nx4,Dx8,m11));
   let(hostTPMz,pkTPMz) = checksign(Dx8,pkCA) in
   let(=n3,=nx4,=hostTBR,Dx9) = checksign(m11,pkTPMz) in
   if Dx9 <> D9 then 0
   else
   (
    new SK2;
    out(ch6,(enc((SK2),pkTPMz),D10,sign((n3,nx4,hostTPMz,enc((SK2),pkTPMz),
        D11),skTBR)));
    out(privSK2,SK2)
   ).
```

4.6 (S3) Sending Encrypted User’s Submitted Biometric Data From the Biometric Reader to the Trusted Platform Module

The processes TPM3 and TBR3 run in sub-protocol (S3). Firstly, the TPM obtains the session key SK2 via the private channel. The TBR also obtains the identity of the TPM and the session key via the private channels from TBR2.

The TPM generates a fresh random nonce and sends it to the TBR. Again, from email discussion about the sequence of the processes, (S3) cannot run successfully before (S1) has run. The TBR verifies the message and sends back the BD encrypted by the session key created in the previous stage. The TPM verifies the received message and decrypts it to retrieve the BD. In order to check protocol properties later, after the BD is received, an *event tcpgetBD()* is launched.

```
let TPM3 =
    in(privChCertTPM3,D12);
    in(privSK2TPM2,SK2TPM2);
    new n5;
    out(ch7,(n5,D12));
    in(ch8,(nx6,Dx13,m10));
    let(hostTBRxx,pkTBRxx) = checksign(Dx13,pkCA) in
    let(=n5,=nx6,=hostTBRxx,=hostTPM,BDReceived,=D14) =
         sdec(m10,SK2TPM2) in
    event tpmgetBD().

let TBR3 =
    in(pState1,P1);
    if P1 <> success then 0
    else
    (
    in(privChCertTBR3,D13);
    in(privSK2,SK2TBR3);
    in(ch7,(nx5,Dx12));
    let(hostTPMzz,pkTPMzz) = checksign(Dx12,pkCA) in
    new n6;
    new BD;
    out(ch8,(n6,D13,senc((nx5,n6,hostTBR,hostTPMzz,BD,D14),SK2TBR3)))
    ).
```

4.7 (S4) Sending a Matching Result

The last sub-protocol (S4) represents the transfer of a matching result on the BC and BD from the TPM to the SC. This sub-protocol includes process TPM4 and process SC4. We model it to check the correctness of the messages received.

The TPM acquires its certificate via the private channel. The SC creates a fresh random number and sends it with a request. The TPM verifies the message. It then signs the match result message and sends it to the SC. We model a match result as a fresh value since we are not concerned with the mechanism by which the TPM carries out the matching process. The SC will check the signature and the correctness of the message. If it is correct, the SC may release the user's credential to the TPM. We do not model how the SC releases this credential since it goes beyond the definition of the protocol.

```
let TPM4 =
    in(privChCertTPM4,D17);
    in(ch9,(nx7,Dx15));
    let(hostSCxx,pkSCxx) = checksign(Dx15,pkCA) in
    new matchResult;
    out(ch10,(sign((nx7,hostSCxx,matchResult,D16),skTPM),D17)).

let SC4 =
    in(privChCertSC4,D15);
    new n7;
    out(ch9,(n7,D15));
    in(ch10,(m12,Dx17));
    let(hostTPMxx,pkTPMxx) = checksign(Dx17,pkCA) in
    let(=n7,=hostSC,matchResultReceived,=D16) = checksign(m12,pkTPMxx) in
    out(ch,secret).
```

5 Analysis

As described in section 2.2, if a naive interpretation of the protocol is applied, an attack is found. After the clarification of the protocol is introduced, we intend to analyse the properites of the protocol to see if the result of the verification is different.

We have analysed the three properties of CPV02, *effectiveness*, *correctness* and *secrecy*, using ProVerif. All three properties of the protocol are satisfied.

Using ProVerif as a verification tool means we can model a Dolev-Yao style attacker that can compose and decompose messages (provided it has relevant cryptographic keys), and has full control over messages that pass over public interfaces and networks.

In the case of the CPV02 protocol, the USB cables are considered part of the public network, since an attacker can interfere with them. The smart card interfaces are also considered public. A prototype device is presented in [11] that can listen to the signal between smart card and smart card reader. This sort of device could be used by an attacker to try to capture a user's biometric code.

5.1 Effectiveness

The TCP will not be given either the BC or the BD unless the integrity of the TPM and TBR has been checked by the SC.

According to the protocol, the BC is transferred from the SC to the platform, and the BD is read from the TBR and sent to the platform; then the two are compared. To protect the BD from a malicious attacker, the device holding this data has to be convinced that the destination to which it will transfer the data can be trusted before the transmission is carried out. This is done by means of integrity checks.

To analyse this property, we use the *event* and *query* command. These two commands are used to check the correctness of sequences of events. While the *event* command is used for launching an event when a certain action is executed, the *query* command is used to prompt ProVerif to verify the correctness of the sequence of events that we specify. If the sequence is not correct, an attack is identified.

In order to verify this property in ProVerif, we encode the integrity check which ensures that the SC is satisfied with the integrity metric of the TCP and the TBR before the trusted platform module receives the user's stored biometric code and user's submitted biometric data. The event *tcpChecked()* is inserted after the SC has checked the integrity of the TCP via the TPM, and the event *tcpgetBC()* is inserted after the TCP has received the BC$^+$.

Similarly, to verify that the integrity metric of the TBR is checked by the SC before the BD is transferred, an event *tbrChecked()* is launched after the SC has checked the integrity metric of the TBR.

It should be noted that there is no direct communication between the TBR and the SC, so the TPM is responsible for checking the integrity metric of the TBR on behalf of the SC. To model this situation, we code it in such a way that if the TPM is satisfied with the integrity metric of the TBR, an event *tbrChecked()* is triggered. The TBR then sends the encrypted BD to the TPM. The TPM verifies the message, stores the BD, and then an event *tcpgetBD()* is inserted$^+$.

We need to check that these events are executed in the correct order, i.e. that the TPM's integrity metric and the TBR's integrity metric have been examined before the TPM receives the BD. This should be the case even in the presence of an attacker that can control the order of the subprotocols and the messages on the network. This check is implemented using ProVerif's *query* command:

query ev: tcpgetBD() $\Rightarrow$ *ev: tcpChecked() & ev : tbrChecked().*
query ev: tcpgetBC() $\Rightarrow$ *ev: tcpChecked() & ev : tbrChecked().*

5.2 Correctness

The TBR is not given the BD until the user is satisfied with the integrity checks on both the TCP and TBR.

This property aims to protect the BD from being read by a malicious biometric reader, the user places her biometric data only on the biometric reader that she trusts. This property is important because if the BD is stolen or accidentally disclosed, it cannot be altered, replaced or regenerated.

To verify this property, we check that the biometric reader (TBR) receives the BD after the integrity metric of the TCP and the integrity metric of the TBR have been checked.

To achieve this, we launch an event *tbrgetBD()* after the BD is created in the process TBR3$^+$.The event would not be triggered without satisfactory integrity checking. To check the correct order of events, we use the query command:

query ev: tbrgetBD() $\Rightarrow$ *ev: tcpChecked() & ev : tbrChecked().*

5.3 Secrecy

An unauthorised entity that can listen to a message between the SC and TCP, or between the TBR and TCP, cannot obtain either the BC or the BD.

As we remarked in section 1, the secrecy of biometric data cannot be relied upon. The security of a protocol should not depend on the secrecy of biometric data. Indeed, this protocol does not depend on it, since it uses a trusted biometric reader to guard against disclosure. Nevertheless, it is good practice to prevent widespread dissemination, and this property verifies that the protocol does not give an attacker easy access to that data.

To model this property we use the *query* command to ask ProVerif whether an attacker can access the BC or the BD. The commands for this verification are

query attacker : BC.
query attacker : BD.

Using these commands to check whether the specified arguments are secret, ProVerif will exhaustively check whether there is any way that an attacker could obtain the information, BD and BC, that we want to protect. If an attacker can obtain the data, then a potential attack has been identified.

6 Conclusion and Future Work

We have presented a specification of the CPV02 biometric authentication protocol, obtained after clarifying details of the protocol through email discussion with one of the authors. We modelled the clarified protocol using the applied pi calculus and the

ProVerif verification tool. We have encoded three intended properties of the protocol, namely *effectiveness*, *secrecy* and *correctness*. The positive results from the verification show that the properties of the protocol hold.

The protocol is successfully verified against the properties. Without this clarification, verification of one of the properties fails.

The CPV02 protocol uses trusted computing platform and involves integrity checking. The trusted computing platform module is an essential part of the protocol in order to guarantee that the components that involved in biometric authentication data cannot be tampered by an intruder. Similar to other classical protocols, nonces are used for checking the freshness of message received and encryption and decryption are also used for the secrecy of message content.

In future work, we will select other protocols with different properties and verify that they hold in a similar way. We would also like to investigate biometric authentication protocol which can be used for unsupervised remote authentication, such as in online banking.

Acknowledgement. Many thanks to Liqun Chen, one of the authors of [2], for detailed email discussion, which was crucial in clarifying the protocol and our understanding of it.

References

1. Lowe, G.: An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters 56* (1995) 131-133
2. Chen, L., Pearson, S., Vamvakas, A.: Trusted Biometric System. Available from http://www.hpl.hp.com/techreports /2002/HPL-2002-185.pdf. (2002)
3. Pearson. S.: How Can You Trust the Computer in Front of You? http://www.hpl.hp.com/techreports/2002/HPL-2002-222.pdf (2002)
4. Abadi, M., Fournet, C.: Mobile values, new names, and secure communications. Proceedings of the 28th Annual ACM Symposium on Principles of Programming Languages (2001) 104-115
5. Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In Steve Schneider, editor, 14th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press (2001) 82-96
6. Blanchet, B.: Automatic Proof of Strong Secrecy for Security Protocols. In IEEE Symposium on Security and Privacy (2004) 86-100
7. Kremer, S., Ryan, M.: Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. Proceedings of the European Symposium on Programming. Lecture Notes in Computer Science 3444. Springer Verlag (2005) 186-200
8. Blanchet, B.: ProVerif Automatic Cryptographic Protocol Verifier User Manual (2005)
9. Delaune, S., Kremer, S., Ryan, M.: Coercion-resistance and Receipt-freeness in Electronic Voting. In 19th Computer Security oundations Workshop. IEEE Computer Society Press (2006)
10. Dolev, D. and Yao, A.C.: On the Security of Public Key Protocols. Proceedings of 22nd IEEE Symposium on Foundations of Computer Science (1981) 350-357
11. Bond, M.: Chip and Pin (EMV) Point-of-Sale Terminal Interceptor. Available from http://www.cl.cam.ac.uk/˜mkb23/interceptor/). (2007)