TELKOMNIKA, Vol. 11, No. 2, February 2013, pp. 855~863
ISSN: 2302-4046

Authentication of the Command TPM_CertifyKey in the Trusted Platform Module

Donglai FU*[1,2], Xinguang PENG[2], Yuli YANG[2]
[1]School of Electronics and Computer Science and Technology, North University of China, Taiyuan, Shanxi, 030051, China
[2]School of Computer Science and Technology, Taiyuan University of Technology, Taiyuan, Shanxi, 030024, China
*Corresponding author, e-mail: hhluci@163.com

Abstract

Trusted Platform Module (TPM) is a key component designed to enable computers achieve greater security. Several vulnerabilities discovered in the TPM highlight the necessity of formal analysis. The procedure invoking an API may be regarded as several interactive processes between the TPM and a user. As a result, the current study formalized the API specifications proposed by Trusted Computing Group (TCG) using applied pi calculus. Meanwhile, two authentication properties between them were also described in a formalized way. With the help of the tool ProVerif, the flaw of the command TPM_CertifyKey was discovered. It was also confirmed on the TPM emulator. Subsequently, the modified API was presented and its authentication properties could be satisfied after verifying again. Results show the model is valid.

Keywords*: trusted computing, trusted platform module, security analysis, authentication*

1. Introduction

Trusted Computing is a solution to resolve some inherent defects in an open-distributed-computing environment. It only embeds a hardware chip called Trusted Platform Module (TPM) on the motherboard to enable computers achieve greater security without changing the current computer architecture. In order to promote the technology, Trusted Computing Group (TCG) proposed the TPM specifications which have been ISO/IEC standards [1-4]. More and more applications based on it have been developed [5-6]. However, the application programming interfaces (APIs) of the TPM have been still described in natural language up to now. Therefore, when implementing them different people have different interpretation, which results in some security problems. Especially, several attacks discovered in a few last years against it highlight the necessity of formal analysis of the API specification. As a result, the current study focuses on the problem.

In our study, a series of behaviors that application software invokes a certain TPM API are regarded as an interactive procedure. Therefore, the problem of the TPM security can be discussed as another problem about security protocols between a user and the TPM. At first, we modeled the TPM APIs using the applied pi calculus and formalized a bidirectional authentication between the user and the TPM. After then, the Object Independent Authorization Protocol (OIAP) and the command TPM_CertifyKey were symbolized in order to describe them accurately. And their formal models described using the applied pi calculus were also given. Moreover, to verify the authentication, the tool ProVerif was adopted and a flaw of the command TPM_CertifyKey was found. Because of the defects of the tool ProVerif, the attack discovered by it was simulated on the TPM emulator 0.7.1. At last, we discussed some fixes to the command TPM_CertifyKey, and proved the authentication properties for the modified API.

The main contributions include:

(1) A complete formal model with supporting verification was given about the TPM APIs and related authentication properties, which established the foundation for analyzing abundant TPM APIs;
(2) To obtain a finer and real formal model, we proposed a modeling idea using an integrated API scenario instead of an individual API. The method can avoid the defects of modeling a

Received August 18, 2012; Revised December 30, 2012; Accepted January 14, 2013

single API;
(3) For the imperfectness of ProVerif, the verification method under the real environment was used, which guaranteed the validity of such attacks discovered by ProVerif;
(4) We formalized the scenario that the command TPM_CertifyKey was executed under the OIAP context and identified its authentication properties. Through the tool ProVerif, we found a flaw. And, in order to prove its authenticity we also stimulated such attacks which took different key combinations as input on the TPM emulator 0.7.1.

2. Releated Works

At present, the study on the TPM safety can be divided into two aspects: trusted application and the TPM APIs security itself. In our study, the latter was concentrated. As for the former, most previous studies assume that the TPM APIs are correct and consistent. For instance, Datta Anupam analyzed formally the security of remote attestation protocols, which thought on both the dynamic trust root and the static, under presuming the TPM APIs were secure [7]. Wang Dan, Wei Jinfeng and Zhou Xiaodong analyzed security properties of current remote attestation protocols through communicating sequential processes (CSP) which is also a formal analysis method [8]. Yang Li, Ma Jianfeng designed a direct anonymous attestation scheme in cross trusted domain for wireless mobile networks based on Canetti-Krawczyk model (CK) under assuming the TPM APIs were safe [9].

On the latter, the current achievements have also two branches: Aiming at single vulnerability and based on a formal model. For example, Bruschi found a leak of OIAP in the TPM which was intended to a replay attack in 2005 [10]. Moreover, Chen also found two attacks against weak authdata secrets [11] as well as shared authorization data [12] respectively. However, lower level analyses of the TPM APIs based on a formal model are rarer. They include: Lin described an analysis of various fragments of the TPM APIs using Otter and Alloy [13]. However, his model omitted some details such as sessions, HMAC and authdata, but included the state. Although he discovered a possible attack on the delegation model of the TPM, experiments with a real TPM had shown that the attack is not possible [14]. In addition, Delaune modeled the TPM APIs based on process algebra [15]. In China, Xu Shiwei and Zhang Huanguo described a formal security analysis on trusted platform module based on applied pi calculus [16].

The current study is inspired by the above results. The model of the TPM APIs was described formally through applied pi calculus as well as its authentication properties between the user and the TPM. We modeled the scenario that the command TPM_CertifyKey is used in the OIAP context. And the authentication of the model was identified and verified by the tool ProVerif. Fortunately, a flaw of the API was founded. To guarantee its authenticity, the attack program was executed on the TPM emulator 0.7.1. It verified all kinds of circumstance under different key combinations. At last, we discussed the fixed method and verified its authentication.

3. TPM Analysis

3.1. Functions and Key Management

TPM provides three kinds of functionality: secure storage, platform identity authentication and platform measurement and reporting. All keys in the TPM are stored in the shielded memory. They only are accessed through its APIs. Keys are organized in a tree hierarchy, with the Storage Root Key (SRK) at its root. And, each key is associated with an authorization data named as authdata. To use the key, a user must offer the relevant authorization data. In addition, the trusted computing specification only allows users to access them by certain authorization session protocol to obtain greater security.

3.2. Analyzing and Symbolizing OIAP

An authorization session created by the OIAP can manipulate any objects, but some commands can not be executed in the session. The complete TPM OIAP session can be seen in the Figure 1. The session can be divided into two phrases: session creation and command execution. After the session is created, the TPM returns one even nonce and one session

handle. During the subsequent procedures, the TPM communicates with the user with a rolling nonce protocol to obtain fresh messages.

Adding the following four points for Figure 1：

(1) The parameter named inParamDigest is the result of the following calculation: inParamDigest = SHA1(ordinal||inArgOne||inArgTwo);
(2) The parameter named inAuthSetupParams refers to the result of the following calculation: inAuthSetupParams = SHA (authLastNonceEven ||nonceOdd ||continueAuthSession);
(3) The parameter named outParamDigest refers to the result of the following calculation:outParamDigest = SHA1(returnCode||ordinal||outArgOne);
(4) The parameter named outParamDigest refers to the result of the following calculation: outAuthSetupParams = SHA1(nonceEven||nonceOdd ||continueAuthSession).

To obtain a relative accuracy description for the OIAP session, it was symbolized. Let U be a generic subject that wishes to create the OIAP session with the TPM T. Moreover, let

(1) returnCode be a return code of the operation;
(2) S_{UT} be a secret shared between U and T;
(3) nonceEven be an even nonce generated by T;
(4) nonceOdd be an odd nonce generated by U;
(5) RQU be a request command type;
(6) RSD be a response command type;
(7) ordinal be a command ordinal;
(8) || be the concatenation of the data;
(9) H be a hash calculation;
(10) paramSize：the size of all parameters;
(11) inArgOne and inArgTwo be two different input parameters;
(12) outArgOne be an output parameter.

Figure 2 shows the procedure of the OIAP session in some abstract symbols. Initially, U requests to open an authorization session by sending the command with parameters including RQU, paramSize and ordinal to T. After then, T sends back to U the session information to handle the authorization session itself. If these steps are correctly performed, U can send to T a command to execute, which embeds the proof that she knows S_{UT}. On receiving it, T will verify the message about authenticity and integrity, and if they are satisfied, T will execute the command on behalf of U, sending back to U the result. Otherwise, the connection will be closed by T.

3.3. Analyzing and Symbolizing the Command TPM_CertifyKey

The operation TPM_CertifyKey allows one key to certify the public portion of another key. As such, it allows the TPM to make the statement “this key is held in a shielded location, and it will never be revealed.” From the TPM specification, an identity key may be used to certify non-migratable keys, but is not permit to certify migratory keys or certified migration keys. However, Signing and legacy keys may be used to certify both migratable and non-migratable keys.

The processing of the command TPM_CertifyKey is described in the Figure 3 in detail. On receiving the command TPM_CertifyKey, the TPM does as follows: Validate the key to be used to certify another key has a signature scheme→Validate two authorization data associated to two keys are true→Check the protocol version (1.2 or 1.1)→Create C1 that a structure TPM_CERTI_INFO and Sign it and return. It should be noted that C1 includes the signature information of the key to be certified.

To describe the command TPM_CertifyKey in symbols, let (See the section 3.2, if there is a symbol not to be introduced):

(1) certHandle be a handle which points to a key to be used to certify another key.
(2) keyHandle be a handle which points to a key to be certified.
(3) antiReplay be a random number to be used to prevent a replay attack.
(4) certAuthHandle be a session handle used for certHandle.
(5) authLastNonceEven be an even nonce lately generated by the TPM in the session used for certAuthHandle.
(6) nonceOdd be an odd nonce generated by the user and associated with certAuthHandle.
(7) continueAuthSession be the flag that the session used for certAuthHandle is continue or not.

(8) certAuth be the authorization session digest for inputs and certHandle, it equals: HMAC(certKey.auth,SHA1(ordinal||antiReplay||authLastNonceEven||nonceOdd ||continueAuthSession).
(9) keyAuthHandle be a session handle used for keyHandle.
(10) keylastNonceEven be an even nonce lately generated by the TPM in the session used for keyAuthHandle.
(11) keynonceOdd be an odd nonce generated by the user associated with keyAuthHandle.
(12) continueKeySession be the flag the session used for keyAuthHandleis continue or not.
(13) keyAuth be the authorization session digest for inputs and keyHandle, it equals: HMAC(key.usageAuth,SHA1(ordinal||antiReplay||keylastNonceEven||keynonceOdd ||continueKeySession).

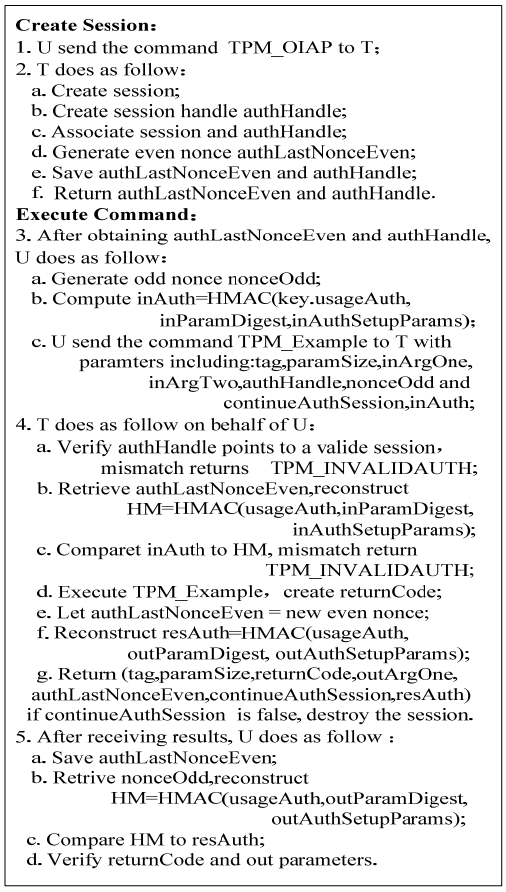

Figure 1. OIAP Processing

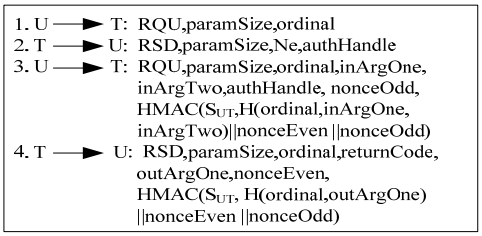

Figure 2. OIAP Processing in Symbols

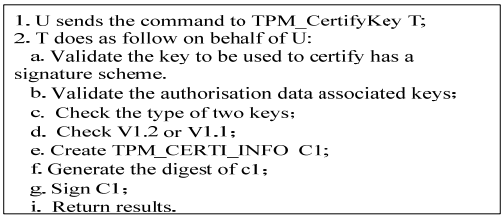

Figure 3. TPM_CertifyKey Processing

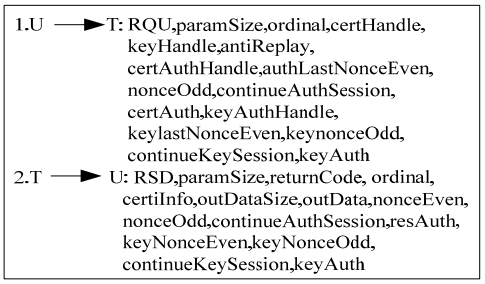

Figure 4. TPM_CertifyKey Processing in Symbols

(14) certifyInfo be a structure TPM_CERTIFY_INFO or TPM_CERTIFY_INFO2 that provides information relative to keyHandle.
(15) outDataSize be the used size of the output area for outData ;
(16) outData be the signature of certifyInfo.
(17) resAuth be the authorization session digest for the returned parameters and certHandle, it equals: HMAC (certKey. auth, SHA1 (returnCode||ordinal||certifyInfo||outDataSize ||outData) ||nonceEven||nonceOdd|| continueAuthSession).
(18) keyAuth be the authorization session digest for the target key, it equals: HMAC (key.usageAuth, SHA1 (returnCode||ordinal|| certifyInfo||outDataSize ||outData) ||keyNonceEven ||keynonceOdd||continueKeySession).

Figure 4 shows the procedure of the command TPM_CertifyKey in some symbols. Initially, U requests to certify a key by sending the command with parameters including RQU, paramSize, ordinal, certHandle and keyHandle and so on to T. Afterwards, T will verify a series of conditions, if they are satisfied, T will execute the command on behalf of U, sending back to U the result. Otherwise, the connection will be closed by T.

4. Modeling TPM

4.1. Definition of the TPM Model

Practically, when developing a trusted application, a user always invokes the TPM APIs to achieve certain security aims. The invoking procedure can be abstracted as an interactive problem between two processes. With reference to the literature [14], the set of the commands included by the TPM specification can be defined as: Tpm = {Tpmi| i∈N+}, each Tpmi corresponds to a specific command, namely an API function. Of course, the set of the user processes can be defined as: User = {Useri| i∈N+}, each Useri is a set of actions executed by a user when invoking an API. Therefore, the TPM model can be defined as:

Definition 1 TPM model TpmModel = !Useri | !Tpmj | Userk | Tpml, Useri,Userk∈User, Tpmj,Tpml∈Tpm, and i≠k, j≠l, the symbol ! notes that the process can be created repeatedly.

4.2. Formal Definition of Authentication

Actually, the authentication is bidirectional between the user and the TPM during invoking a certain API.

The authentication that the TPM certifies the user can be described as: If the TPM has executed a certain command, then the user in possession of the relevant authdata has previously requested the command. It can be defined as formally:

Definition 2 The authentication of the user can be expressed by the correspondence properties:TpmConsider(M1,M2,…Mi,…Mn)==>UserRequest(M1,M2,… Mi,…Mn).

The event TpmConsider(M1,M2,…Mi,…Mn) expresses the TPM has executed a certain API. The event UserRequest(M1,M2,…Mi,…Mn) expresses the user has previously requested a certain API. The symbol Mi represents the ith parameter of the event.

The above property can hold if the event TpmConsider(M1,M2,…Mi,…Mn) occurs, then the corresponding event UserRequest(M1,M2,… Mi,… Mn) has previously appeared.

Similarly, the authentication that the user certifies the TPM can be described as: If the user considers that the TPM has executed a certain command, then the TPM really has executed the command. It can be defined as formally:

Definition 3 the authentication of the TPM can be expressed by the correspondence properties: UserConsider(M1,M2,…Mi,…Mn)==>TpmConsider(M1,M2,…Mi,…Mn).

The event UserConsider (M1,M2,… Mi,…Mn) expresses the user considers the TPM has executed a certain API. The event TpmConsider(M1,M2,…Mi,… Mn) expresses the TPM has really executed the API. The symbol Mi represents the ith parameter of the event.

The above property can hold if the event UserConsider(M1,M2,…Mi,…Mn) occurs, then the corresponding event TpmConsider(M1,M2,…Mi,… Mn) has previously appeared.

In the TPM specification, two authentication properties are achieved by the authorization HMACs which accompany the corresponding command. The HMAC of the former is provided by the user, but the HMAC of the latter is given by the TPM.

4.3. Modeling the Scenario of the Command TPM_CertifyKey

Invoking the command TPM_CertifyKey needs open two authorization sessions, and mutual authentication only depends on HMACs from two parties. Thus, some details to be irrelevant to the authentication are omitted.

Figure 5 shows the whole scenario that the command is used in the OIAP context. Initially, a user sends the OIAP command ordinals S1 and S2 to the TPM in order to obtain two authorization session handles H1 and H2 and two corresponding even-random numbers Ne1 and Ne2. Then, a user sends the command TPM_CertifyKey with parameters including a key to used to certify another key, a key to be certified, a random number aRn to prevent replay attacks, two session handles H1 and H2, two authorization HMACs HMAC(D1, SHA1(aRn || Ne1 || No1)) and HMAC(D2, SHA1(aRn || Ne1 || No1)), D1 and D2 are the authorization data of

two keys K1 and K2. Finally, the user obtains a signature of K2 created by K1 and two authorization HMACs.

Combining Figure 5 with Definition 1, the scenario of the command TPM_CertifyKey can be formally defined as:

Definition 4 CertifyKey = Tpm_OIAP1 | Tpm_OIAP2 | User_OIAP1 | User_OIAP2 | !Tpm_CertifyKey | !User_CertifyKey.

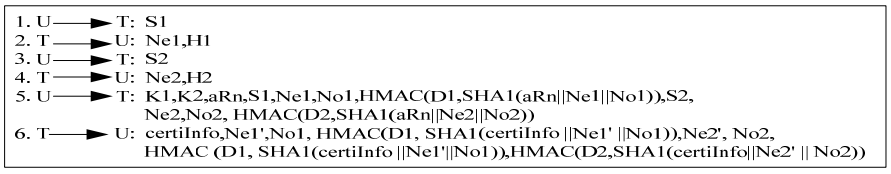

Figure 5. Scenario of TPM_CertifyKey in OIAP

Although there is not a requirement that the session can not be repeatedly used in the TPM specification, only one command is executed in each session. The reason has two points: the non-monotomic state of the TPM and the fault of not modeling a state transition system using the tool ProVerif. The above scenario includes:

(1) The processes Tpm_OIAP1 and User_OIAP1 which are used to create one session only are executed one time;
(2) The processes Tpm_OIAP2 and User_OIAP2 which are used to create another session are also executed one time;
(3) The processes Tpm_CertifyKey and User_CertifyKey which are used to stimulate the command execution can be executed repeatedly.

With the help of Figure 5, definition 2 and 3, two authentication properties included the aforementioned scenario can be defined as definition 5 and definition 6 respectively.

Definition 5 The authentication of the user can be expressed by the correspondence properties: TpmConsider(D1,PK1,D2,PK2,certiInfo)==> UserRequest(D1,PK1, D2,PK2).

In the above definition, D1 and D2 are two authorization data. PK1 and PK2 are two public keys. certiInfo is a signature of K2 singed by K1.The property can hold If the event TpmConsider(D1,PK1,D2,PK2,certiInfo) occurs, then the event UserRequest(D1,PK1, D2,PK2) has previously appeared.

Definition 6 The authentication of the TPM can be expressed by the correspondence properties:UserConsider(D1,PK1,D2,PK2,certiInfo) ==>TpmConsider(D1,PK1,D2,PK2,certiInfo).

In definition 6, D1 and D2 are two authorization data. PK1 and PK2 are two public keys. certiInfo is a signature of K2 singed by K1. The property can hold if the event UserConsider(D1,PK1,D2,PK2, certiInfo) occurs, then the corresponding event TpmConsider(D1,PK1,D2,PK2,certiInfo) has previously appeared.

5. Experiment and Evaluation

5.1. Verification Using ProVerif

The primary goal of ProVerif is the verification of cryptographic protocols. Cryptographic protocols are concurrent programs which communicate using public communication channels such as the Internet to achieve some security-related objectives. These channels are assumed to be controlled powerfully and completely, the attacker may: read, modify, delete, and inject messages. We only model honest parties because the environment can capture the behavior of dishonest participants. Therefore, in the model of the above scenario, there are six processes: two processes named Tpm_OIAP1 and User_OIAP1 are used to create one session; two processes named Tpm_OIAP2 and User_OIAP2 are used to create another session; two processes named Tpm_CertifyKey and User_CertifyKey are used to stimulate a series of behaviors of the command TPM_CertifyKey; More specifically, the process Tpm_CertifyKey models the behavior of the TPM, but the process User_CertifyKey models the behavior of the user. Figure 6 and Figure 7 respectively show their behaviors in detail.

According to Figure 6, the process User_CertifyKey obtains two keys, namely to certify and to be certified, from a channel. Then, she constructs two HMACs based on the authorization data of two keys respectively. The event UserRequestCertifiedKey is triggered in order to declare the user has requested the command TPM_CertifyKey with parameters by prearranged data. At last, after the user receives the response of the command, she checks two HMACs. If match, the event UserConsiderCertifiedKey will be triggered in order to declare the user has considered the TPM has executed the command.

Figure 7 describes the behavior that the TPM executes the command TPM_CertifyKey. Initially, the TPM receives two key-handles. Subsequently, she checks two HMACs. At last, she signs the key h2 with another key h1, and triggers the event TpmConsiderCertifiedKey to declare she has executed the command TPM_CertifyKey. When the above model was input into the tool ProVerif, the results showed two properties could not hold (see Figure 8). And, it discovered the attack track (see Figure 9).

```
let User_CertifyKey=
   in(c,(ne1:TPM_NONCE,h1:TPM_KEY_HANDLE,
        ne2:TPM_NONCE,h2:TPM_KEY_HANDLE));
   new n:TPM_NONCE;
   new no1:TPM_NONCE;
   new no2:TPM_NONCE;
   let (secret1:TPM_AUTHDATA,pk1:TPM_PUBKEY)
                        = (getSecret(h1),getPubKey(h1)) in
   let (secret2:TPM_AUTHDATA,pk2:TPM_PUBKEY)
                         = (getSecret(h2),getPubKey(h2)) in
   let hmac1:bitstring = hmac(secret1,(n,ne1,no1))in
   let hmac2:bitstring = hmac(secret2,(n,ne2,no2))in
   event UserRequestCertifiedKey(secret1,pk1,secret2,pk2);
   out(c,(n,no1,hmac1,no2,hmac2));
   in(c,(ne1':TPM_NONCE,ne2':TPM_NONCE,
          hmac1':bitstring,hmac2':bitstring,certif:bitstring));
   if hmac1'= hmac(secret1,(n,ne1',no1,certif)) then
   if hmac2'= hmac(secret2,(n,ne2',no2,certif)) then
   event UserConsiderCertifiedKey(secret1, pk1, secret2,
            pk2,certif);
```

Figure 6. User_CertifyKey Process

```
let Tpm_CertifyKey=
   in(c,(ne1:TPM_NONCE,h1:TPM_KEY_HANDLE,
       ne2:TPM_NONCE,h2:TPM_KEY_HANDLE));
   let (secret1:TPM_AUTHDATA,pk1:TPM_PUBKEY,
       sk1:TPM_KEY) = (getSecret(h1),getPubKey(h1),
                                    getKey(h1)) in
   let (secret2:TPM_AUTHDATA,pk2:TPM_PUBKEY,
       sk2:TPM_KEY) = (getSecret(h2),getPubKey(h2),
                                    getKey(h2)) in
   in(c,(n:TPM_NONCE,no1:TPM_NONCE,hmac1:bitstring,
                       no2:TPM_NONCE,hmac2:bitstring));
   if hmac1=hmac(secret1,(n,ne1,no1)) then
   if hmac2=hmac(secret2,(n,ne2,no2)) then
   let certif:bitstring=cert(sk1,pk2) in
   new ne1':TPM_NONCE;
   new ne2':TPM_NONCE;
   let hmac1':bitstring = hmac(secret1,(n,ne1',no1,certif))in
   let hmac2':bitstring = hmac(secret2,(n,ne2',no2,certif))in
   event TPMConsiderCertifiedKey(secret1,pk1,secret2,pk2,
                                                certif);
   out(c,(ne1',ne2',hmac1',hmac2',certif)).
```

Figure 7. Tpm_CertifyKey Process

Verification results about definition 5:

Starting query
event(TPMConsiderCertifiedKey(s1_17087,pk1_17088,
s2_17089,pk2_17090,certif_17091)) ==>
event(UserRequestCertifiedKey(s1_17087,pk1_17088,
s2_17089,pk2_17090))
goal reachable:
begin(UserRequestCertifiedKey(secret2[],pk(key2[]),
secret1[],pk(key1[]))) ->
end(TPMConsiderCertifiedKey(secret1[],pk(key1[]),
secret2[],pk(key2[]),cert(key1[],pk(key2[]))))
event(TPMConsiderCertifiedKey(s1_17087,pk1_17088,
s2_17089,pk2_17090,certif_17091)) ==>
event(UserRequestCertifiedKey(s1_17087,pk1_17088,
s2_17089,pk2_17090)) is false.

Verification results about definition 6:

Starting query
event(UserConsiderCertifiedKey(s1,pk1_30,s2,pk2_31,
certif_32)) ==>
event(TPMConsiderCertifiedKey(s1,pk1_30,s2,pk2_31,
certif_32))
goal reachable:
begin(TPMConsiderCertifiedKey(secret1[],pk(key1[]),
secret2[],pk(key2[]),cert(key1[],pk(key2[])))) ->
end(UserConsiderCertifiedKey(secret2[],pk(key2[]),
secret1[],pk(key1[]),cert(key1[],pk(key2[]))))
event(UserConsiderCertifiedKey(s1,pk1_30,s2,pk2_31,
certif_32)) ==>
event(TPMConsiderCertifiedKey(s1,pk1_30,s2,pk2_31
,certif_32)) is false.

Figure 8. Outputs of ProVerif

U→T*: n,ne1,no1,h1, hmac(getSecret(h1), (n,ne1,
no1)),ne2,no2,h2,hmac(getSecret(h2),(n,ne2,no2))
T*→T: n,ne2,no2,h2,hmac(getSecret(h2),(n,ne2,
no2)),ne1,no1,h1, hmac(getSecret(h1), (n,ne1,
no1))
T→T*:certif,ne1',no2,hmac(getSecret(h2),(n,ne1',no
2,certif)),ne2',no1,hmac(getSecret(h1),(n,ne2',no1,ce
rtif)
T*→U: certif,ne2',no1,hmac(getSecret(h1),(n,ne2',
no1,certif)),ne1',no2,hmac(getSecret(h2), (n,ne1',
no2,certif))

Figure 9. Attack Track

In Figure 9, let T* and U* represent the TPM and the user respectively they are all disguised by the attacker. Before the command is received by the TPM, the attacker swaps two keys and their HMACs. Then, when returning the response, they are inversed again. Although the user does not find any errors, she obtains an error signature.

5.2. Verification on the TPM Emulator

Because of the imperfectness of the tool ProVerif, it needs further validation when a security property is not hold. Therefore, we build an environment using TPM Emulator 0.7.1, Ubuntu 10.0.4（2.6.27.38）and jtss 0.4.1 in order to verify our attack. We wrote a program stimulating the attacker with the help of the development tool Eclipse 3.2. According to the TPM specification, the key to certify other key is only one of them including an identity key, a signing key and a legacy key. And the key to be certified is only one of them including a signing key, a storage key, an identity key, a bind key and a legacy key. We performed the experiment which took different key combinations as inputs. Table 1 shows the results.

Table 1. Verification on the attack

Signing key	Key to certified	Attack
identity	signing	success
identity	storage	failure
identity	bind	failure
identity	identity	success
signing	identity	success
signing	storage	failure
signing	bind	failure
signing	signing	success

Only when the key to be certified is also a signing key or an identity key, the attack is true. Otherwise, the attack will fail because of checking the type of a key. However, these factors do not be considered in our applied pi calculus model. Therefore, we conclude the result of ProVerif is true.

5.3 Modified API

Practically, the TCG specification does not disguise two different HMACs, which results in the above attack. Therefore, if we add the key handle corresponding to the HMAC to it when calculating, and then the attack will disappear. The tool ProVerif gives a proof for the modified API.

6. Conclusion

In the study, we model the TPM using applied pi calculus, and formalize the bidirection authentication between the TPM and a user. Subsequently, we analyze the command TPM_CertifyKey in a complete context. Through the tool ProVerif, we find two authentication properties of it can not hold. Because of the imperfectness of the tool ProVerif, we write the attack program on the TPM emulator 0.7.1. At last, we verify the attack again after modifying the formal model of the command TPM_CertifyKey.The results show our model is valid.

References

[1] International Organization Standardization (ISO). ISO/IEC 11889-1:2009. *Information technology—Trusted Platform Module—Part 1: Overview*. Geneva: International Organization Standardization (ISO); 2009.
[2] International Organization Standardization (ISO). ISO/IEC 11889-2:2009. *Information technology—Trusted Platform Module—Part 2: Design principles*. Geneva: International Organization Standardization (ISO); 2009.
[3] International Organization Standardization (ISO). ISO/IEC 11889-3:2009. *Information technology—Trusted Platform Module—Part 3: Structures*. Geneva: International Organization Standardization (ISO); 2009.

[4] International Organization Standardization (ISO). ISO/IEC 11889-4:2009. *Information technology—Trusted Platform Module—Part 4: Commands*.Geneva: International Organization Standardization (ISO); 2009.
[5] Liye TIAN, Changxiang SHEN. Productive Information System Oriented Trust Chain Scheme.*TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(5): 1093-1100.
[6] Teddy M, Andri Z. Securing E-mail Communication Using Hybrid Cryptosystem on Android-based Mobile Devices. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(4): 788-797.
[7] Datta Anupam, Franklin Jason, Garg Deepak. *Logic of secure systems and its application to trusted computing*. Proceedings of the 30th IEEE Symposium on Security and Privacy. Berkeley. 2009; 221-236.
[8] WANG Dan, WEI Jinfeng, ZHOU Xiaodong. Design and validation for a remote attestation security protocol. *Journal on Communication*. 2009; 30(11A):2 9-35.
[9] YANG Li, MA Jianfeng, JIANG Qi. Direct Anonymous Attestation Scheme in Cross Trusted Domain for Wireless Mobile Networks. *Journal of Software*. 2012; 23(5): 1260-1271.
[10] Bruschi D, Cavallaro L, Lanzi A, et al. *Replay attack in TCG specification and solution*. Proceedings of the 21st Annual Computer Security Applications Conference. Los Alamitos. 2005: 127-137.
[11] Chen L, Ryan MD. *Offline dictionary attack on TCG TPM weak authorization data and solution*. Future of Trust in Computing. Wiesbaden. 2009: 193-196.
[12] Chen L, Ryan MD. *Attack, solution and verification for shared authorization data in TCG*. Proc. of the 6th Int Workshop on Formal Aspects in Security and Trust. Berlin. 2010: 201-216.
[13] AH Lin. Automated Analysis of Security APIs. Master's Thesis. Boston: Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology. 2005.
[14] Ables K. An attack on key delegation in the trusted platform module. Master's Thesis. Birmingham: School of Computer Science, University of Birmingham; 2009.
[15] Delaune S, Kremer S, Ryan MD, et al. *A formal analysis of authentication in the TPM*. Proc. of the 7th Int Workshop on Formal Aspects in Security and Trust. Berlin. 2011:111-125.
[16] Xu Shiwei, Zhang Huanguo. Formal Security analysis on trusted platform module based on applied pi calculus.*Journal of Computer Research and Development*, 2011; 48(8): 1421-1429.