The Verification of Merchant Registration in SET Protocol

Qingfeng Chen, Chengqi Zhang, and Jie Lu

Faculty of Information Technology
University of Technology, Sydney
P.O. Box 123, Broadway, NSW 2007, Australia
qchen@it. uts. edu. au, chengqi@it. uts. edu. au, jielu@it.uts.edu.au
Tel: +61 2 95144534 FAX: +61 2 95141807

Abstract: *The Merchant Registration in the Secure Electronic Transaction (SET) protocol is presented and its formal analysis is described. Based on the analysis, this paper unveils some potential vulnerabilities of SET. Such vulnerabilities have been identified when ENDL (extension of non-monotonic logic) is applied to verify the Merchant Registration in SET protocol. To our knowledge, this is the first attempt to analyze the Merchant Registration. Also, we present the modeling of the Merchant Registration for the purpose of mechanically model checking.*

Keywords: *Security; Secure transaction protocol; Confidentiality; Electronic Commerce; Verification.*

1. Introduction

Security is a necessary concern in e-commerce. Nearly everyday, a news organization reports a potential security violation on the Internet. Thus, a number of secure protocols have been developed to meet the user's expectation for secure business transactions. For example, toward achieving secure on-line transaction, *Visa* and *MasterCard* developed the SET protocol to safeguard the secure payment card transactions over the Internet. The secure protocols [1][6][7] widely use encryption and decryption functions to achieve the secure requirements. But the cryptographic protocol is prone to error and a number of well-known and largely used cryptographic protocols have been proved to have some potential flaws [17]. Recently, using formal methods to verify such protocols has received increasing attention.

Currently, there are many remarkable efforts on developing methodologies, theories, logics, and other supporting tools. These efforts are effective to overcome the weakness and reduce the redundancies in the protocol design stage. Some typical approaches are the BAN logic [9], GNY logic [10], BGNY logic [11] and AUTOLOG logic [13]. They focus on the verification of authentication protocols. E-commerce protocols, however, are inherently complex and a number of published protocols of this nature contain subtle errors. For example, the protocols developed by Needham and Schroeder [17] are vulnerable to attacks by intruders [8].

Kailar's analysis of simple e-commerce protocols has been proven to be successful [12]. After that, there have been attempts to verify more realistic protocols such as Kerberos [14] and TLS/SSL [15]. However, the analysis of complex e-commerce protocols is still out of their reach. Meadows and Syverson [20] give a formal specification for payment transactions in the SET protocol but do not describe the actual analysis. Bella and Paulson [16] presented the formal verification of cardholder registration in Isabelle, in which the formal protocol model is declared as a set of traces. However, the methods for verification of e-commerce protocols are not quite as mature as those used for authentication protocols. The new types of threats, such as payment card transaction, caused by new requirements of on-line transaction will be a significant challenge to the analysis of e-commerce protocols. To our knowledge, existing tools cannot cover these problems in many cases.

The SET protocol is intended for the processing of payment card transactions over the Internet. However, the formal techniques used for the verification of authentication protocols cannot be directly applied to payment card transaction of SET, mainly because of the difference between traditional authentication and particular properties of SET. The SET protocol includes some standard authentication requirement. For example, in Cardholder Registration, cardholder (*C*) firstly must register with a Certificate Authority (*CA*). After receiving

the initial response, *C* wants to verify the certificate and ensure that he was communicating with the right *CA*. Also, some particular properties are included in the payment process. For example, when the cardholder verifies the certificate of *CA*, he wants to make sure that it is the proper *CA* who issues the certificate. Thus *C* has to validate it by traversing a series of certificate authorities in a multilevel trust chain. The validation process may stop at a level that has been previously validated. In the case where the validation fails the cardholder could cancel the transaction.

In this paper, we show how the ENDL can be used for verifying the Merchant Registration in the SET protocol. The dynamic and non-monotonic properties of ENDL allow us to convert the transaction into a sequence of actions and deal with the special processes of SET such as the verification by a multilevel trust chain. Also, we show how the verification can be implemented in high efficiency by using mechanically model checking.

The rest of this paper is organized as follows. In Section 2, the process of Merchant Registration is cursorily described. Section 3 presents the notations, metrology and terminology. Section 4 gives the complete verification to the Merchant Registration. Section 5 gives the modeling of the Merchant Registration in Prolog. Finally, Section 6 concludes this paper.

2. Merchant Registration in SET Protocol

The SET protocol [1][2][3] has been developed and standardized jointly by *Visa* and *MasterCard*. SET uses cryptography to provide confidentiality, interoperability and payment integrity, and to authenticate the identity of the cardholder and the merchant to each other. In general, the payment system participants include Cardholder (*C*), Issuer, Merchant (*M*), Acquirer and Payment Gateway (*P*). Cardholder uses the credit card, issued by issuer, to purchase a product or service provided by the merchant. Payment Gateway processes the payment instruction from the cardholder and the payment request from the merchant. A hierarchy of trust, namely **PKI** (Public key infrastructure) tree, is introduced and used for certificate issuance. To verify the validity of a received certificate, you must follow this tree to a known trusted party, *CARoot* who provide the user with the digital certificate for signature and encryption. All SET participants know the public signature key and public key-exchange key of CARoot. But the verification may not always traverse the geopolitical certificate authority in this way.

SET consists of five phrases. This paper presents the verification of Merchant Registration. In this phase, the user can register the signature certificate, which will be used to confirm that a transaction is from a legal merchant of a payment brand. Merchant Registration includes five fundamental steps.

- *Initiate request.* Merchant *M* sends initiate request to *CA*.
- *Initiate response.* After receiving the request, *CA* selects the appropriate registration form and digitally signs it by generating the message digest of the form and encrypting it with *CA* private signature key. Finally, *CA* sends registration form and *CA* certificates to merchant.
- *Merchant certificate request.* The merchant verifies the *CA* certificate and signature and creates two pairs of keys for later use. Then *M* fills out the registration form with information such as the merchant's name, address, and merchant ID. The merchant takes this registration information and combines it with the public keys in a registration message that is digitally signed by *M*. Next merchant generates a random symmetric encryption key. This key, along with the account data *AcctData(M)* is then encrypted with the *CA* public key-exchange key. The encrypted certificate request message is transmitted to *CA*.
- *Certificate Authority processes request and creates certificates.* The *CA* verifies the merchant signature. If the signature is valid, the message is processed; otherwise the message is rejected and an appropriate response message is returned to the merchant. Next the *CA* must verify the information from the registration request using known merchant information. If the information in the registration request is valid, the *CA* creates and digitally signs the merchant certificates. The validity of these certificates will be determined by the policy of *CA*. The certificates are then encrypted using a new randomly generated symmetric key, which in turn is encrypted using the merchant public key-exchange key. Finally, the response is sent to *M*.
- *Merchant Certificates.* When *M* receives the response from *CA*, it decrypts the digital envelope to obtain the symmetric encryption key, which is used to decrypt the registration response containing the merchant certificates. If the certificates are verified, the merchant stores the certificates on its computer for use in future transaction.

3. Overview of ENDL

Some notations of ENDL [4][5] are recalled here by outline such as encryption, decryption, and hashing, which are largely used in SET protocol.

In general, uppercase *X, Y, CA* (Certificate Authorities) and *CARoot* denote particular principals; m_1, m_2, ..., and m_n denote specific messages; *T* denotes specific timestamp that can be used to both authenticate the validity of message and assert that the message is generated by current session; *Cert* denotes the certificate that needs to be verified; *k* denotes encryption or decryption keys; *Generate* and *Send* denote specific actions (Encryption and digital signature etc. are some mapping operation on message, but not action.).

Function words are the abstract description of operations on the message. They consist of the encryption, signature, message digest and such like.

E(m, k): The message *m* is encrypted by the symmetric encryption key *k*.

S(m, k): The message *m* is encrypted by the public key *k*, namely *Kpb(X), Kpv(X), Spb(X), and Spv(X)* listed below.

H(m): The message digest of message *m* encoded by one-way hashing algorithm *H(x)*. One-way Hash function has the property that, given the output, it is difficult to determine the input.

Kpb(X): The public key-exchange key of *X*.

Kpv(X): The private key-exchange key of *X*.

Spb(X): The public signature key of *X*.

Spv(X): The private signature key of *X*.

$< m_1, m_2, \ldots\ldots, m_n >$: The combination of messages m_1, $m_2, \ldots\ldots$, and m_n.

When the message digest of a message is encrypted using the senders' private key and is appended to the original message, the result is known as digital signature of the message (see abbreviation below).

Action: Applied to describe that the principal try to execute some appointed task. There are three types of actions listed below:

Generate(X, m): *X* generates the message *m*.

Send(X, Y, m): *X* sends the message *m* to *Y* after *X* successfully generated the message *m*.

Verify(X, Cert, <CA1, CA2 ,....., CARoot>): *X* verified certificate *Cert* by traversing the trust chain, CA_1, CA_2, ... , to the root *CARoot* (Below, we assume $CA = \{CA_1, CA_2, \ldots\ldots, CARoot\}$).

Applying the conventional logic operator can derive further action. Suppose α and β are basic sequence of actions, then $\alpha \circ \beta$ denotes the conjunction of α and β. Thus, integrating with the known actions can generate the new actions.

Predicate: Applied to express the knowledge state and belief relation of principals:

Know(X, m): *X* knows message *m*. It is possible *X* generate the message *m* by itself or receives *m* from *Y*.

Auth(X, Y, m): *X* authenticates message *m* is sent by *Y*, and *m* has not been modified. If *X* can authenticate the message *m* is valid, then return true; otherwise return false.

IsVerified(X, CA, Cert): if *X* can verify the *Cert* is valid, then it returns true, otherwise returns false.

Equal(m, m'): if message *m* equals to message *m'*, then it returns true, otherwise returns false.

Assertion:

$$P \vdash_{\alpha} Q$$

P and *Q* present the set of formulae; α denotes the sequence of actions. This assertion means if the premise *P* is true then α can be executed, and the conclusion *Q* will be true if α can be performed successfully.

Abbreviation:

Sign(X, m)= $<m, S(<ID_X, H(m)>, Spv(X))>$: This represents plain text *m* and *X*'s identifier ID_X are affixed to *X*'s digital signature.

$Sign(X, m)_T = <m, S(<ID_X, T, H(m)>, Spv(X))>$: Inserting the timestamp *T* into *Sign(X, m)*;

$S_0(X, m) = S(<ID_X, H(m)>, Spv(X))$: This represents identifier ID_X is attached to *X*'s digital signature before *X* encrypted the message digest of *m* in private signature key *Spv(X)*.

$S_0(X, m)_T = S(<ID_X, T, H(m)>, Spv(X))$: Inserting the timestamp *T* into *S0(X, m)*

CertK(X) = *Sign(CA, <X, Kpb(X)>)*: The key-exchange certificate of *X*;

CertS(X) = *Sign(CA, <X, Spb(X)>)*: The signature certificate of *X*.

4. The Verification of Merchant Registration

In our former work, we have shown how to use our framework to validate the secure protocols, including the cardholder registration in the SET protocol [4][5]. Currently, some related works [8][16] are being developed to analyse this protocol, but the large e-commerce protocols, such as SET, are very complex and have never been formalized before. Meanwhile, the specification of SET protocol itself is incomplete [16]. For instance, a merchant verifies that the Chall-EE (EE's challenge to CA's signature freshness), a fresh random, received matches the one sent in the **Me-AqCInitReq** (Merchant or Acquirer Certificate Initialization Request messages). EE denotes the End Entity, including *M*, *C* or *P*. Actually, it is dangerous since an intruder could intercept Chall-EE and replay it [4]. Some potential vulnerability in cardholder registration has been reported in [4]. Thus, it is not surprising that some potentially dangerous flaws can be detected during the verification of Merchant Registration. Also the feasible method to settle the problem is presented and wants to draw the attention of the protocol designer.

Based on the description of Merchant Registration in Section 2, the verification is divided into five phases accordingly. Some hidden messages and complex operations such as secret value and hashing algorithm etc. are abstracted away for brevity.

4.1 Initiate Request

Merchants must register with a *CA* before they can process the SET transactions. In order to send SET message to the *CA*, the merchant must have a copy of the

CA public key-exchange key, which is contained in the *CA* key-exchange certificate. Meanwhile *M* also needs a copy of the registration form from the financial institution, which is used to identify the Acquirer to *CA*. The registration starts when *M* sends initiate request *InitReq* to *CA* and wishes to get a copy of the *CA* key-exchange certificate and a suitable registration form. This process can be denoted by abbreviation, *Send(M, CA, InitReq)*. Nothing needs to be verified since no message is deemed necessary to be protected during this step.

4.2 Initiate Response

After receiving *InitReq*, the *CA* identifies the merchant's financial institution and selects the suitable registration form, which is digitally signed using the *CA* private signature key. Next the *CA* sends the registration form along with *CA* certificates to merchant.

α = *Generate(CA, RegForm)* ∘ *Send(CA, M, <RegForm, S(H(RegForm), Spv(CA))>)* ∘ *Send(CA, M, <CertS(CA), CertK(CA)>)*

Actually, the intruder can intercept the communication between the *CA* and *M* and replay this message.

(1) *M* → *Z (CA)*: *InitReq*
(2) *Z(M)* → *CA*: *InitReq'*
(3) *CA→Z(M)*: *<CertS(CA), CertK(CA), RegForm', S(H(RegForm'), Spv(CA))>*
(2') *Z(M)* → *CA*: *InitReq"*
(3') *CA→Z(M)*: *<CertS(CA), CertK(CA),RegForm", S(H(RegForm"), Spv(CA))>*
(4) *Z(CA)→M*: *<CertS(CA), CertK(CA), RegForm', S(H(RegForm'), Spv(CA))>*

Here, the intruder *Z* intercepts the initial request from *M* to *CA* and replaces it with a new initial request *initReq'* and sends it to *CA* as the message (2). *CA* replies with message (3). *Z* impersonates *M* to produces a new message (2') and sends it to *CA*. The *CA* answers *M* with corresponding message (3'), and then *Z* intercepts it. Finally, *Z* impersonates *CA* to send an outdated message (4) that is intercepted by *Z* from message (3). This attack is continuously used until *M* wants to bring about authentication between *M* and *CA* again.

The reason is that the SET documents cannot give the detailed descriptions and sometimes are misleading. For example, it is unconvincing that the MeAqInitReq message can be securely sent to *CA* since no clear description indicates that the encryption or security policy has been applied. However, we cannot exclude the possibility that the Chall-EE is intercepted or tamped by the intruder. Thus, we think it is unfeasible, at least dangerous, to believe the *CA*'s signature freshness. The SET designer should describe this more clearly. This paper gives an optional method, in which the timestamp, Chal-EE and identifier ID_{CA} are included in the response message before it is encrypted.

(3) *CA→Z(M)*: *<CertS(CA), CertK(CA),RegForm', S(<*ID_{CA}*, Chall-EE, H(RegForm')>, Spv(CA))>*

This can prevent the above attack from working because the intruder will be unable to replay the message as before [18][19].

On the other hand, *CA* may alter the registration form *RegForm* for the sake of network block. If *CA* does not let *M* know what has been altered about *RegForm* or let *M* know the change but the content of new initiate response does not keep the same as the old one, namely ¬*Equal(m, m')*, *M* should stop the current transaction and validate the initiate response again. We do not need to concern about the *CA* certificates since trusted authority issues them.

4.3 Merchant Certificate Request

Once *M* has a copy of the *CA* key-exchange certificate *CertK(CA)*, the merchant can register to accept SET payment instruction and process SET transactions. *M* also generates two pair public/private key pairs for use with SET: key-exchange and signature. To register, the merchant fills out the form with registration information such as the merchant's name and ID. *M* takes this registration information and combines it with the public keys in a registration message. The merchant digitally signs the registration message. Next *M* encrypts the message using a randomly generated symmetric key. This key, along with the merchant's account data *AcctData(M)* is then encrypted using *Kpb(CA)*. Finally the merchant sends encrypted certificate request *CertReq* to *CA*, which contains the public keys, *Kpb(M)* and *Spb(M)*.

α = *Generate(M, <Spb(M), Spv(M),Kpb(M), Kpv(M)>)* ∘ *Generate(M, CertReq)* ∘ *Generate(M,* k_1*)* ∘ *Send(M, CA, E(<CertReq, S(H(CertReq),Spv(M))>,* k_1*)* ∘ *Send(M, CA, S(<*k_1*, AcctData(M)>, Kpb(CA)))*

When the *CA* receives the merchant's certificate request, it decrypts the digital envelop with *CA* private key-exchange key to obtain the symmetric key k_1, which is used to decrypt the request.

During the process that the merchant uses *Kpb(CA)* to encrypt *<*k_1*, AcctData(M)>*, *M* does not affix timestamp *T* to this message. The reason that the challenge Chall-EE cannot avoid the replay attacks has been presented above so it is not repeated. We are concerned with the direct exposure of symmetric key k_1 due to negligence or a design flaw in the system, i.e., an intruder can break into the computer and get the key. Thus, the intruder *Z* can repeat the similar attack like the case in initiate response since *Kpb(CA)* is known to all participants.

Msg 4.3.1 *M* → *Z (CA)*: *S(<*k_1*, AcctData(M)>, Kpb(CA))*
Msg 4.3.2 *M* → *Z (CA)*: *E(<CertReq, S(H (CertReq), Spv(M))>,* k_1*)*
Msg 4.3.3 *Z(M)* → *CA*: *S(<*k_1*', AcctData(M)>,*

Kpb(CA))

Msg 4.3.4 *Z(M)* → *CA*: *E(<CertReq', S(H(CertReq), Spv(M))>, k_1')*

Actually, the merchant *M* may alert symmetric key k_1 or account data *AcctData(M)* since *M* can wonder these messages have been masqueraded or the communication of the network is blocked. Meanwhile, whether the principals possess memory to message is also a crucial security factor, but the detailed description is outside this paper. Thus, we only show which reference rule the result derives from, but not give the details. Several possibilities are listed below:

(1) *M* changes *AcctData(M)* but not alter k_1,
(2) *M* changes k_1 but not alter *AcctData(M)*,
(3) *M* changes *AcctData(M)* and k_1,

In case (1), *M* creates a new *AcctData(M)*. If *M* does not let *CA* know what has been altered or let *CA* know the change but the content of *AcctData(M)* does not keep the same as the original one, *CA* should fail to authenticate *AcctData(M)*; In case (2), *M* creates a new k_1. If *C* does not let *CA* know what has been changed about k_1, *CA* should fail to authenticate <k_1, *AcctData(M)*>; in case (3), *C* alters *AcctData(M)* and k_1. If *M* does not let *CA* know what have been changed about k_1 and *AcctData(M)* or let *CA* know the new *AcctData(M)* but it does not keep the same as the original one, *CA* should fail to authenticate <k_1, *AcctData(M)*> and *RegFormReq*. Once one of them happens, *CA* must stop the process and not send the request to *M*. But SET does not provide the clear description on the similar security policy.

4.4 Certificate Authority processes request and creates certificates

In this stage, *CA* processes merchant's request. *CA* decrypts the digital envelope to get the symmetric encryption key, which is used to decrypt the registration request. *CA* then verifies merchant signature. If the verification is successful, the message processing continues, otherwise it is stopped and an appropriate response is returned to *M*. Next the *CA* validates the information from the registration request with known merchant information. If the information is verified, the *CA* creates the merchant certificates.

α = *Generate(CA, CertRes)* • *Send(CA, M, <CertS(M), CertK(M)>)* • *Send(CA, M, CertRes, S(H(CertRes), Spv(CA)))* • *Send(CA, M, CertS(CA))*

When *M* receives the certificate request, the merchant needs to verify the *CertS(CA)*, *CertS(M)* and *CertK(M)* by using a special trust chain, PKI tree, which popularly appears in the certificate verification of SET. We only show the verification of *CertS(CA)* since the other two processes are similar.

(1) *Send(CA, M, CertS(CA))* [action]
(2)*Know(M, CertS(CA))* (1)[R-1]
(3)*Verify(M, CertS(CA), CA)* [action]
(4)*IsVerified(M, CA, CertS(CA))* [discriminant]
(5)*Auth(M, CA, <Spb(CA), Spv(CA)>)* (2)(3)(4) [7-1]

The verification process may stop at a level that has been previously validated. But if *M* cannot verify the *CA* certificate is valid during traversing the trust chain, {CA_1, CA_2, ..., *CARoot*}, the transaction is rejected.

After confirming the validity of *CertS(CA)*, *CertS(M)* and *CertK(M)*, *M* can verify *CA* signature. For the ambiguity of Chall-EE mentioned above, so it is possible that the intruder *Z* can handle the next attack.

(1) *CA*→ *Z(M)*: *CertS(CA)*, *CertS(M)*, *CertK(M)*, *<CertRes, S(H(CertRes), Spv(CA))>*
(1') *Z(CA)* → *M*: *CertS(CA)*, *CertS(M)*, *CertK(M)*, *<CertRes', S(H(CertRes'), Spv(CA)')>*

Spv(CA)' denotes the replays of old keys or substitution of bogus key. Obviously, it will cause some serious damages to financial transaction since the merchant cannot verify the certificate response. Eventually, an error message is returned to *CA*.

In practical transaction, *CA* may alter *CertRes* for wrong format or content etc. If *CA* does not let *M* know what has been changed about *CertRes* or let *M* know the change but the content of new certificate response does not keep the same as the old one, namely *¬Equal(CertRes,Old_CertRes)*, *M* should fail to authenticate *CertRes*, otherwise this will cause some potentially dangerous security.

4.5 Merchant Certificate

Based on the verification of certificates, the merchant stores the certificates, *CertS(M)* and *CertK(M)*, and information from response on the merchant's computer for use in future e-commerce transactions.

The above analyses report some subtle flaws of SET. Also, some potential concerns in practical transaction are also presented. This can benefit to the future revision of SET protocol.

5. Modeling Merchant Registration

This Section presents the design of an automatic inference framework. The automation can make the verification easy. Modeling the verification of Merchant Registration in Prolog requires the distinct specifications of SET. The definition of SET, however, is incomplete. To resolve this problem, we decided to adopt the business description of SET [1] as the standard specification. That means the modeling should conform to the flow of transactions and the information contained in this flow.

5.1 Designing the Inference Engine

First of all, we need to establish the inference engine, which is the kernel in the inference framework. An inference engine knows how to actively use the knowledge in the base. Four basic functions of the framework are listed below:

- Adopt the external file as the infrastructure of knowledge base;
- Input the facts and new knowledge by the interaction between the user and the program;
- Access the existing knowledge base;
- Output the result to the database.

Figure 5.1.1 listed below present the algorithm flow outline. Firstly, we must check if all the rules have been tried out. If so, the inference should be halted. If not, it will continue to check if the conclusion of the current rule has been stored in the Database. If the conclusion can be found, we will skip this rule and start to check the next one; otherwise we have to match all the conditions of this rule one by one. If and only if the return value is true, we can create the explanation and store the result into Database, otherwise we will skip it and check the next rule.

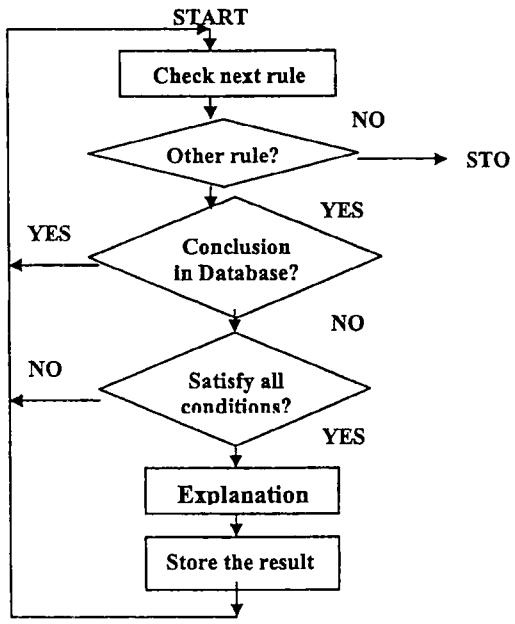

Figure 5.1.1 The algorithm flow

5.2 Handling the Knowledge base and Facts

Two kinds of approaches are applied to handle the knowledge base here. One is inputting a new knowledge and the other is accessing an existing knowledge base, which can be denoted by '*a*' and '*b*' respectively.

process ('a') if *acquisition*
process ('b') if *write ("Name of the knowledge base"),*
Readln (Name),
Consult (Name).

If the system fails to find the name of the knowledge base inputted by user, it will return an error message to the user. To acquire the knowledge, it is acted as rule consisted by the condition and conclusion, which are handled by two functors, *condition_read* and *conclusion_read*, respectively. Also, we number the rules according to the sequence.

A predefined functor *facts_reading* is used to collect the facts. Every time, when the fact is inputted, we check if it is a terminal symbol '*'. If so, the process is halted. If not, the functor *assertz* is used to store it into the database.

5.3 Recognition

The infrastructure of the framework is presented above. Next we employ an instance to show how to use this framework to mechanically verify it. The detailed illustrations of whole program, however, are outside this paper, thus we focus on the rules handling, which actually is one of the most important components.

The process of merchant certificate request is intercepted to be the instance. *Auth(CA, M, CertReq)* is the proposition that we want to authenticate. *Know(CA, Kpv(CA))* and the messages, *E(<CertReq, S(H(CertReq), Spv(M))>, k_1)* and *S(<k_1, AcctData(M) >, Kpb(CA))* etc. sent from *M* to *CA* are acted as the known facts and stored into the database. When the system reads the terminal symbol "*" the input of facts will be stopped. Next we build the knowledge base and store the rules with the serial number by sequence.

rule(1, ["Know(CA, S(<k_1, AcctData(M)>,Kpb(CA))", "Know(CA, Kpv(CA))"), "Know(CA, k_1)", "Know(CA, AcctData(M))"]
rule(2, ["Know(CA, E(<CertReq, S(H(CertReq), Spv(M))>, k_1))", "Know(CA, k_1)"], "Know(CA, <CertReq, S(H(CertReq), Spv(M))>)")
rule(3, ["Know(CA, <CertReq, S(H(CertReq), Spv(M))>)"], "Know(CA, CertReq)", "Know(CA, S(H(CertReq), Spv(M)))")
rule(4, ["Know(CA, CertReq)", "Know(CA, S(<ID_M, T, H(CertReq)>, Spv(M)))", "Know(CA, |Clock –T| < Δ t_1 + Δ t_2)"], "Auth(CA, M, CertReq)")

where Clock is the local time, Δ t_1 is an interval representing the normal discrepancy between the server's clock and the local clock, and Δt_2 is an interval representing the expected network delay time[4].

After establishing the knowledge base and inputting all the facts, we can input:

?- Auth(CA, M, CertReq).

According to the inference engine, the program then

tries to find the matched rule from the knowledge base. Finally, the system cannot verify *Auth(CA, M, CertReq)* and return a message to the user. The result is consistent with what we describe in Section 4.

Although the instance is only a section of Merchant Registration, it is convenient to extend the knowledge base and input new facts for verifying the whole phrase. Thus, our framework is effective and promising in mechanically verifying the secure protocols.

6. Conclusion

Because the security protocols in e-commerce are very important and complicated, so a small negligence in design stage can cause the hidden trouble of security. We have used ENDL to provide the formal analysis of Merchant Registration in SET protocol. To our knowledge, this is the first attempt to analyze this phase. We have reported some of the potential vulnerabilities and some concerns in the practical circumstance. Also, we have suggested the optional measures to fix these problems and want to draw the attention of the protocol designer.

In particular, we have described the modeling of Merchant Registration for the purpose of mechanically model checking, by which the verification can be implemented in high efficiency. We used one instance drawn from Merchant Registration to present how to use this framework to mechanically verify it. From the observation, our method is useful and promising.

Reference:

[1]SET Secure Electronic Transaction Specification, Book 1: *Business Description, Version 1.0*, May 31, 1997.

[2]SET Secure Electronic Transaction Specification, Book 2: *Programmer's Guide, Version 1.0*, May 31, 1997.

[3]SET Secure Electronic Transaction Specification, Book 3: *Formal Protocol Definition, 1.0*, May 31, 1997.

[4]Chen Q.F and Zhang C.Q, Using ENDL to verify Cardholder Registration in SET Protocol, *Proceeding of the International Conference on E-Business 2002 Beijing (ICEB2002)* to be held in Beijing, China, May 23-26, 2002.

[5]Bai Shuo, Chen Q.F, The Verification Logic for Secure Electronic protocols, Journal of Software (China), 11(2):213-221, 2000.

[6]GIE CB. C-set architecture de sécurité June 1996.

[7]Glassman S., Manasse M., Abadi M., Gauthier P., Sobalvarro P., The Millicent protocol for inexpensive electronic commerce, *Fourth International World wide Web Conference, Boston, MA, USA*, Dec 1995.

[8]Bolignano D., Formal verification of cryptographic protocols, *Proceedings of the third ACM Conference on Computer and Communication Security*, 1996.

[9]Burrows M., Abadi M., Needham R., A logic for Authentication. *ACM Transactions on Computer Systems*, 8(1):18-36, February 1990.

[10]Gong L., Needham R., and Yahalom R., Reasoning about belief in cryptographic protocols, *Proceeding of the Symposium on Security and Privacy*, pages 234-248, Oakland, CA, May 1990.

[11]Brackin S., A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols, *Proceeding of the 1996 IEEE Computer Security Foundations Workshop IX*, (1996) 62-76, IEEE Computer Society Press.

[12]Kailar R., Reasoning about Accountablity in Protocols for Electronic Commerce, *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, (1995) 236-250, IEEE Computer Society Press.

[13]Kessler V., Wedel C., AUTLOG – an advanced logic of authentication, *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, Los Alamitors, IEEE Computer Society Press, pages 90-99, 1994.

[14]Bella G, Paulson L.C, Kerberos version IV: Inductive analysis of the secrecy goals, *Proceeding of the 5 th European Symporism. On Research*, volume 1485, 361-375, Springer-Verlag, 1998.

[15]Paulson L.C., Inductive analysis of the internet protocol TLS, *ACM Transaction on Information and System*, 2(3):332-351,1999.

[16]Bella G, Massacci F., Paulson L.C, Tramontano P., Formal Verification of Cardholder Registration in SET, *Proceedings of the 6th European Symposium on Research in Computer Security (ESORICS'00)*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[17]Needham R., Schroeder M.D, Using Encryption for Authentication in Large Networks of Computers, *Communications of the ACM*, 21(12): 993-999, Dec 1978.

[18]Denning D., Sacco G, Timestamp in Key Distribution Protocols,*Communications of ACM*, 24(8), 533-536, August 1981.

[19]Lowe G, Some New Attacks upon Security Protocols, Proceedings of the 9th IEEE Computer Security Foundations Workshop, 162-169. IEEE Computer Society, June 1996.

[20]Meadows C., Syverson P., A formal specification of requirements for payment transactions in the SET protocol, *Proceedings of Financial Cryptography 98*, volume 1465 of *Lecture Notes in Comp. Sci.* Springer-Verlag, 1998.