The Florida State University
Department of Computer Science

Analysis Of SET Protocol in CPAL-ES

By

Shanmuga Raja Ramaswamy

Major Professor: Dr. Alec F. Yasinsac

Project submitted to the Department of Computer Science

in partial fulfillment of the requirements for the

MASTER OF SCIENCE DEGREE (M.S.)

The members of the committee approve the Master's Project of **Shanmuga Raja Ramaswamy** defended on

_____

Dr. Alec F. Yasinsac

Major Professor

_____

Dr. Ladislav J. Kohout

Committee Member

_____

Dr. Ernest L. McDuffie

Committee Member

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. Alec F. Yasinsac, for his mentorship in the project work. His timely suggestions, ideas, guidance and encouragement made the completion of this project possible. Also, I would like to thank the faculty, staff & many people in the Department Of Computer Science who helped me in my graduate studies. I would also like to thank Philip Kovoor and Prative Chend for their help, support and encouragement.

This work will be dedicated to my symbol of inspiration and source of moral strength: Mom and Dad.

1.INTRODUCTION

Internet has been one of the most dominant technologies in the field of computers in the last decade. The domain of internet usage ranges from sharing information, enabling communication between people, sharing resources, remote education, distributed computation, online shopping, etc. The advent of Internet led to the beginning of a whole new set of technologies. One such technology that redefined the manner in which business was conducted is Electronic Commerce (E-Commerce). [WBEC]

E-Commerce or E-Business is the method of conducting business and transactions through networks and computers. E-Commerce involves buying and selling of goods, business-to-business exchange of data and trading with the participating entities in remote locations. Indeed, this was a very convenient way of doing business and soon business through E-Commerce was a norm. But this convenience also brought along with it some serious security concerns. [RS01]

All the transactions carried out through E-Commerce are monetary and it involves the flow of money in some form from one remote place to another through networks. The most common way in which money is transferred in E-Commerce is using a credit card. Since the networks are inherently insecure it is possible for the malicious intruder to break into a network and gain access to secure data (Ex. reading credit card number). This requires securing communication channels by encrypting data, identifying and authorizing participating entities, limiting access to resources and by implementing security protocols to provide the needed security. [SEC01]

Transport Layer Security (TLS) [TLS], Secure Sockets Layer (SSL) [SSL] and Secure Electronic Transaction (SET) [SETBD] are some of the popular security protocols. Transport Layer Security (TLS) is a protocol that ensures privacy between communicating applications and their users on the Internet. Secure Socket Layer (SSL) is a protocol developed by Netscape for transmitting private documents or data via the Internet. SET (Secure Electronic Transaction) is a system for ensuring the security of financial transactions on the Internet [SETBD].

The security protocols are a collection of algorithms that facilitate the secure communication of data. Immaterial of the environment these protocols are used, the users of these protocols depend highly on the correctness of these protocols. Any minor flaw in these protocols could compromise the security of the data which these security protocols are intended to protect. [SP01] , [SS99]

Due to the critical nature of data, documents or applications these security protocols protect, it is doubly important to prove the correctness of these protocols. Formal methods have been used in a wide variety of environments to verify whether complex systems accomplish their intended functionality. One area where formal methods received wide attention is in the area of security protocol verification[SS99]. Cryptographic Protocol Analysis Language – Evaluation System (CPALES) is a framework developed by Dr.Alec Yasinsac that uses Weakest Precondition Reasoning in the evaluation of protocols and has been shown to facilitate the detection of flaws and inconsistencies in the security protocols (Ex.TLS). In this project we analyze the Secure Electronic Protocol (SET) using the CPAL-ES. [YAS96], [YAS00], [YW93]

The paper will be organized as follows. Chapter 2 will give an overview of the CPAL-ES environment with an example. Also the work of Justin Childs in verifying the TLS protocol using CPAL-ES will be discussed. Chapter 3 will give an overview of the SET protocol, which includes its history, architectural design and detailed design. Chapter 4 will discuss the analysis of SET protocol using the CPAL-ES environment. Chapter 5 will give the conclusion and the knowledge gained due to this work.

2. BACKGROUND

The analysis of SET was done in a framework known as the Cryptographic Protocol Analysis Language - Evaluation System (CPAL-ES). [YW99]

2.1 CPAL-ES

CPAL-ES is an automated system that allows the analyst to formally define the Cryptographic Protocol in a way that adds structure and simplicity to the protocol evaluation process. Security protocols are expressed in the CPAL language, which includes the assumptions, principal's actions and goals of the protocols. The system returns a Verification Condition (VC) for the given protocol. This verification condition is then simplified in an attempt to make the predicate easier to understand. The success of the security protocols depends on the truth or falsity of the VC.

The evaluation of protocols in CPAL-ES is a three-step process:

1.Encode the protocol actions in CPAL

2.Translate the specification into a Verification Condition and

3.Prove the Verification Condition. Using weakest precondition reasoning to evaluate the protocols, the analyzer tries to find flaws or inconsistencies in the protocol. At the end of the analysis, even if no flaws are found, a more thorough analysis of the protocols are made, which increases the credibility of these protocols. [AYMR01]

2.2 Syntax of CPAL:

Assignment Statement:

Messages are non-destructively assigned in CPAL [YAS96]. The source value is copied into the memory location of the destination identifier. The statement *A: X: =Y;* means that the value addressed by name Y is stored in the value of the address referenced by name X in A's address space. [YAS96]

Generation of nonce's:

A principal at any point in the protocol conversion process can generate a nonce by assigning the value "new" to the variable. Ex. *A: Na: = new;*

Concatenation of values:

Concatenated values produced during the cryptographic protocol runs are represented using a structure called a compound value. *Ex. A: X: = <A, B, C>;* In CPAL, provisions are there to retrieve concatenated messages into individual message components. "DOT" operator is used for representing individual components within the compound value. Ex. <A, B, C>.3 will mean C. Similarly, X.3 also means C, which represents the third value in the above-concatenated statement.

Encryption/Decryption operators:

In CPAL an encrypted value is enclosed in square brackets. It is then prefixed by the operator "e" and suffixed with the key with which the value is encrypted. Ex.[x]k.

The decryption operator is similar to the encryption operator except that it is replaced by “d” instead of “e”. Ex. d[e[x]k]k (decrypting the message x which was encrypted using key k).

If Statement:

Traditional psuedocode languages do not carry decision operators along with them. CPAL provides an “if” structure and the “then” and “else” structure as alternatives to be executed. By providing these operators CPAL facilitates the representation of change in the flow of control of the protocol steps.

Ex. *C: if (RRPID_RES == RRPID) then*

{assert (RRPID == RRPID_RES);}

else {assert(error);}

The secure send, receive, assert and assume operators are discussed in the next section. [YAS96]

2.3 Specification of a protocol in CPAL:

This is the step in the evaluation process in which the analyst manually converts the Cryptographic Protocol (CP) in consideration into the Cryptographic Protocol Analysis Language (CPAL). CPAL provides sufficient expressiveness to enable protocol analysts to specify complex CPs and to facilitate automated analysis of CPAL coded protocols through formal methods. CPAL is similar to the Standard Notation (SN) pseudocode previously used for specifying CPs. But, CPAL provides great enhancements to SN to help the analyst to model the data in a protocol.

The predominant difference between CPAL and SN is the identification of each protocol action (sending, receiving, encryption, decryption etc.) by preceding them with their principals. In CPAL, data is organized into specific address spaces that each participating entity can see or have access to. A dot notation is used to identify the address space where the value represented by the identifier resides, Ex. value A.k resides in principal A's address space. SN suffers with this shortcoming because it uses global address space and so the origin of the value cannot be traced easily.

In CPAL, the analyst expressing the protocol also includes explicit receipt of messages and also encryption and decryption of messages. This allows the analyst to easily understand whether a value was encrypted or merely forwarded as an encrypted value. The action of sending an encrypted value from one principal to another principal is shown in both SN and CPAL.

SN:

A → B {msg} k

CPAL:

A: => B (e [msg] k);

B: ← (msg');

B: msg: = d [msg'] k;

In the case of SN, only the send and encryption operations are explicit. The receipt, decryption and name binding are implicit. In the case of CPAL send, receive, encrypt,

decrypt and name binding are explicit which helps in the better understanding of the protocols.

Protocol goals and assumptions cannot be expressed in SN. The goals and assumptions relate to the comparative value of data elements before, during and after protocol session. Often they are provided separately in some narrative form. In CPAL, it is possible to encode the goals and assumptions of the principals into the specification, expressed as predicates that the specifier ‘assumes’ or ‘asserts’ to be true. Assumptions are encoded at the beginning of the protocol specification and assertions can be specified at any point in the protocol. A detailed description of CPAL is given in [YAS96].

2.4 The Verification Condition Generator:

The purpose of encoding a protocol is to allow the creation of its formal definition. In CPAL-ES, this definition is known as Verification Condition (VC). Assigning each CPAL statement with a weakest precondition definition produces the VC. The weakest precondition technique has three components: the state prior to the execution of the program statement, the program statement itself and a goal that is to be true after the statement executes. Preconditions are formed from protocol assumptions, assertions and actions. The final VC takes the form of a logical predicate that can be simplified to TRUE if a given protocol correctly executes. [YAS96]

Weakest preconditions are a relatively simple and effective way of formally defining the protocol statements. Most of the CPs are as short as the simple computer programs. Sometimes, CPs can be quite large ex. SSL, TLS, SET. TLS protocol has been analyzed by Justin Childs using the CPAL-ES environment [JUS00], [JUS01].

2.5 Verifying Protocols Using CPAL-ES:

An example is presented to illustrate CPAL-ES. The CPAL-ES evaluation of Woo and Lam protocol is provided. Woo and Lam protocol offers a two-way authentication and is a three party protocol that requires five transmissions to complete authentication.

CPAL-ES of this protocol provides three components during this evaluation

1. Protocol specification listing.
2. The verification condition derived from the protocol statements before simplification is performed.
3. The simplified verification condition for the protocol.

The listing below is taken directly from the CPAL-ES system [YAS96] .

```
X: assume ((S.Kbs == B.Kbs));
X: assume ((S.Kas == A.Kas));
X: => A (X.B);
A: <-(A.B);
X: => B (X.B);
B: <-(B.B);
A: => B (A.A);
B: <-(B.A);
B: => A (B.Nb);
```

```
A: <-(A.Nb);
A: => B (e [<A.A, A.B, A.Nb>] A.Kas);
B: <-(B.tickA);
B: => S (e [<B.A, B.B, B.Nb, B.tickA>] B.Kbs);
S: <-(S.msg4);
S: (S.A, S.B, S.Nb, S.tickA): = d [S.msg4] S.Kbs;
S: S.A_B_Nb: = d [S.tickA] S.Kas;
S: assert ((S.A_B_Nb == <S.A, S.B, S.Nb>));
S: => B (e [<S.A, S.B, S.Nb>] S.Kbs);
B: <-(B.msg5);
B: B.A_B_Nb: = d [B.msg5] B.Kbs;
B: assert ((B.A_B_Nb == <B.A, B.B, B.Nb>));

 *** End of Protocol ***

(TRUE and (B.Nb ==
d[e[d[d[e[<A.A,e[B.Nb]S.kas>]S.kbs]S.kbs.2]S.kas]S.kbs]S.kbs))

****** Simplified predicate follows.

TRUE

****** NO MORE PREDICATE
```

Figure 1 CPAL version of Woo and Lam protocol

The simplification of the initial verification condition removes all references to variables, leaving the final verification condition to be TRUE.

2.6 Analysis of the TLS family of protocols:

The Transport layer Security standard (TLS) is a refinement to the Secure Socket Layer(SSL) developed by the Internet Engineering Task Force. The goal of the TLS protocol is to provide privacy and data integrity between two communicating applications.

TLS protocol, like SSL, has provided a suite of protocols that could be agreed upon by two communicating principals. This allowed different key systems, levels of security and levels of authentication to be agreed upon by the two principals. In fact, TLS has provided twenty-eight different cipher suite definitions. In TLS, a cipher suite defines the key exchange method, a signature method, the authentication, the encryption algorithm to be used and the hashing function used. This makes the specifications complex in comparison to the older protocols.

Needham and Schroeder initiated cryptographic protocol analysis [NS78] in 1978. For the next 20 years the analysis focussed on simple, serial protocols in a structured and well-understood manner. During recent years, the advent of many new complex protocols like SET, IKE, TLS and SSL made the early analysis methods difficult and in some cases impossible. These complex protocols offer sub protocols that are mutually agreed upon by participants in a protocol execution. Interactions between the sub-protocols may allow the protocol to be subverted. Unfortunately, analyzing sub-protocol interactions can lead to rapid increases in the cost of analysis as the

number of possible interactions grows rapidly with the addition of sub-protocols [JUS01].

Since the security protocols have been sequential in nature there were very few tools to examine the sub-protocol interaction. Analysis tools had no reason to create ways to model branching operations. The Cryptographic Protocol Analysis Language Evaluation System (CPAL-ES) is based on a formal programming method, so a branch operator was included from the outset. This inclusion allowed the tool to represent and analyze protocols that include sub-protocols [JUS01].

One of the main problems with the security protocols is that they are difficult to analyze and design securely with their large size. Justin Childs used CPAL-ES to analyze sub-protocol interactions and illustrate the technique used to examine TLS [JUS 00]. CPAL-ES becomes even more useful when interaction among sub-protocols makes it even more difficult to keep track of effects of one subprotocol upon another. The results of the extensive analysis and the specification methods used to arrive at these results can b e found in [JUS00], [JUS01].

TLS was one of first complex modern protocol to be analyzed and verified using CPAL-ES environment that gave the confidence to undertake the job of verifying another modern complex protocol namely SET. A comprehensive analysis of the TLS protocol using the CPALES can be found in [JUS00].

3.Secure Electronic Transaction – SET

3.1 History

MasterCard and VISA developed SET in collaboration from leading technology companies, which includes Microsoft, IBM, Netscape, SAIC, GTE, RSA, Terisa Systems and VeriSign. On February 1st, 1996 these companies announced the single technical standard for safeguarding the payment card purchases made over open networks. This standard is called the SET Secure Electronic Transaction specification. SET specification includes digital certificates, which is a way of verifying the actual identity of the parties participating in the transaction. By using these sophisticated cryptographic techniques, SET protocol, aims to make cyberspace a safer place for conducting business and thereby increase consumer confidence in E-Commerce. [SETBD]

SET was developed to address these major requirements in the online shopping industry:

1 Provide confidentiality of information -- accomplished by the use of message encryption
2 Ensure the integrity of all transmitted data – accomplished by the use of digital signatures
3 Authenticate a cardholder meaning that he is the legitimate user of the branded payment card – accomplished by the use of digital signatures and cardholder certificates

4 Authenticate a merchant to accept payment card transactions and assure his relationship with an acquiring financial institution – accomplished by the use of digital signatures and merchant certificates

5 Protect all legitimate parties involved in the transaction using the best security practices

6 Facilitate interoperability among software and network providers – accomplished by the use of specific protocols and message formats. [SETBD]

3.2 Scope of the SET protocol:

The following are within the scope of the SET protocol

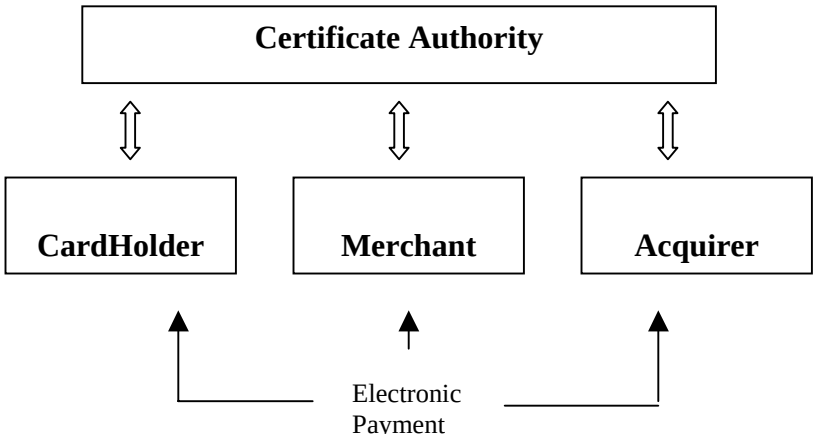


Fig 2. Participants in a SET protocol & their Interactions

1. *Application of cryptographic algorithms*

   Data Integrity, Confidentiality of Information, Cardholder account authentication, merchant authentication and interoperability are some of

the features of the SET specification. These features are ensured through the application of cryptographic algorithms

2. *Certificate messages and object formats*

   Certificates strengthen authentication of the participating entities. Certificate Authority could supply certificates that offer a high assurance of personal identity.

3. *Purchase messages and object formats*

   It checks the status of the processing of an order after the purchase the response has been received. It also indicates the status of authorization, capture and credit processing.

4. *Authorization messages and object formats*

   Authorization messages are sent between the participating entities and the payment gateway. It consists of a Request-Response pair.

5. *Capture messages and object formats*

   Digitally signed Capture Request containing information about the transaction is encrypted using a newly generated symmetric key and is sent to the Payment Gateway which then approves the Request in its Response message.

6. *Message protocol between participants*

   Participants in the SET compliant application message transfer should adhere to the protocol defined by the SET specification.

The following are outside the scope of the SET protocol

1. *Payments beyond the domain of payment cards*

   SET provides security for financial transactions carried through SET compliant merchants, cards and participating entities. Security for other forms of financial transactions is not supported by SET.

2. *Screen formats of order entry forms as defined by each merchant.*

   Screen formats including the content, presentation and layout of order entry forms are dependent on each merchant.

3. *Protection for participating system*

   Security of data on cardholder, merchant, and payment gateway systems including protection from viruses, Trojan horse programs, and hackers [SETBD]

3.3 Payment System Participants:

In SET the electronic processing begins with the cardholder. The set of participants in a SET protocol are represented in Fig 2.

Cardholder: A cardholder uses a payment card that has been issued by the Issuer. SET ensures that the cardholder's interactions with the merchant and the payment card information remain confidential.

Issuer: It is a financial institution. It establishes an account for the cardholder. It also issues the payment card. The Issuer also guarantees payment for authorized transactions using the payment card

Merchant: The merchant offers goods or services in exchange for a payment. In order to accept payment cards the merchant must have a relationship with an Acquirer

Acquirer: A financial institution that establishes an account with a merchant and process payment card authorizations and payments.

Payment Gateway: It is a device operated by an Acquirer or a designated third party that processes merchant payment messages, including payment instructions from cardholders.

Brand: Financial Institutions came up with payment card brands in order to protect and advertise the brand. It creates an atmosphere conducive to establishing and enforcing rules for use and acceptance of their respective payment cards. It also provides a network to interconnect the financial institutions.

Third Parties: Issuers and Acquirers sometime assign processing of payment card transactions to these third-party processors. [SETBD]

3.4 Use of Cryptography in SET

SET relies heavily on cryptography to ensure message integrity, confidentiality and validity. Messages are initially encrypted using a randomly generated symmetric

encryption key. The next step is to encrypt this key using the message receiver's public key. This combination of messages is referred to as the "digital envelope" and this is sent along with the encrypted message to the receiver (Fig 5). Upon receiving the digital envelope, the recipient decrypts it using its private key to obtain the randomly generated symmetric key and then uses the symmetric key to unlock the original message (Fig 6). Some of the basic concepts in the field of cryptography are explained below.

Symmetric Key Cryptography: It uses the same key to encrypt and decrypt the messages. The sender and the receiver of the message must share the same key. This requires trust between the parties who share the messages and share the key. Example of this kind of cryptography is the Data Encryption Standard (DES), used by financial institutions to share the personal identification numbers (PINs). This type of cryptography is also known as *Secret-key Cryptography*. The main drawback of this concept is that it is impractical for exchanging messages with a large group of previously unknown individuals over a public network. Fig 3 represents Symmetric Key Cryptography.

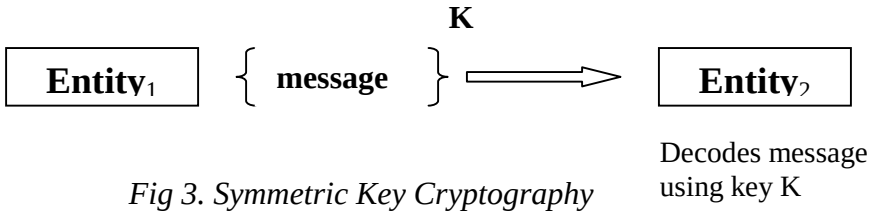


Fig 3. Symmetric Key Cryptography

Public-Key Cryptography: In this approach, each participant creates two unique keys. One is called the *"public key"* which is published to all and the other is called the *"private key"* which the participant keeps it secret from others. Also, one key is used for encrypting the data while the other key is used for decrypting the data. Moreover, the two

keys are mathematically related and so the data encrypted with either key can only be decrypted using the other key. The user distributes the public key. Because of the mathematical relationship between the two keys, the user and others who receive the public key can be assured that the data encrypted by the public key and sent to the user can only be decrypted by the user when he uses the private key. An example of the public-key cryptography is the well-known RSA algorithm (named after the inventors Rivest, Shamir and Adleman). This kind of cryptography is well suited in situations where the user has to share his key to many unknown participants. Also, it is 10 to 10,000 more computation intensive than Symmetric Key Cryptography. Fig 4 illustrates Asymmetric Key Cryptography.

Digital Signatures: A digital signature provides a way to associate the message with the sender. It helps in ensuring the authenticity and integrity of the message. When combined with *message digests*, encrypting messages using private keys allows users to digitally sign the messages. A message digest is a unique value generated for that particular message. Passing the message through a one-way cryptographic function generates a message digest. This message digest is then encrypted using the sender's private key and is appended to the original message resulting in the digital signature of the message. The recipient of the digital signature can be sure that the message really came from the sender because changing even one character in the message changes the message digest in an unpredictable way.

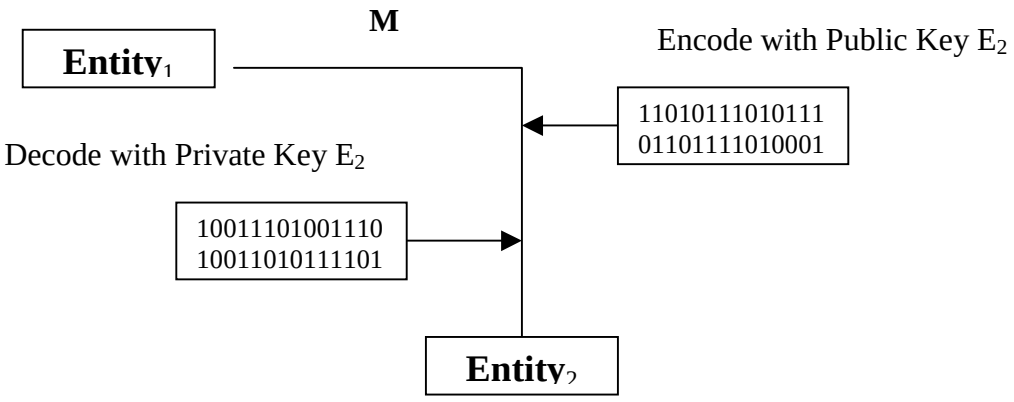


Fig 4. Asymmetric Key Cryptography

Digital Certificates: Before two-entities start using public key cryptography, each has to make sure that the other entity is authenticated. It is done by the use of a trusted third-party to authenticate that the public-key belongs to the intended person. The third-party in this case is called the *Certificate Authority (CA).* Initially the participant who wishes to be authenticated will have to prove his identity to the CA. Once the participant proves his identity, the CA creates a message containing the participant's name and its public key. The CA digitally signs this message known as the certificate.

The summary of cryptographic activities (encryption/decryption) carried out during the message transfer is explained below. Assume the transaction is between participants A and B. Participant A wish to sign some sensitive data and send it as an encrypted message to participant B.

Encryption: (A's side)

Step I: 'A' runs the message through a one-way cryptographic function to generate the message digest. The value of the message digest is unique. This

message digest is then encrypted with A's private signature key to produce the d*igital signature.*

Step II: A creates a random symmetric key. The message, digital signature and a copy of A's certificate (which contains his public key) are encrypted using the newly created random symmetric key.

Step III: A encrypts its symmetric key using B's public key, which it should have obtained prior to the beginning of the transaction. The encrypted key known as the digital envelope, along with the encrypted message will be sent to B.[SETBD]

At the end of the encryption process, A sends the following message components to B: Symmetrically encrypted message, signature and certificate and the digital envelope. This is illustrated in *Figure 5* in the next page.

Decryption: (B's side)

Step I: On receiving the message from A, B will decrypt the digital envelope using its private key to retrieve the random symmetric key.

Step II: Using the retrieved random symmetric key, B will then decrypt the encrypted message to retrieve the message, digital signature and the certificate and runs the message through the same one-way cryptographic function to generate the message digest for the message to be compared later.

Step III: A's digital signature is decrypted using its public key obtained from its certificate. Thus, B retrieves the original message digest. B compares this message digest with the message digest calculated in Step II. If they are equal,

then it means that the message is not tampered and its integrity and validity are proved. Moreover, through the use of digital signatures and certificates B also gets to know that the message indeed came from A. In case of message digest mismatch, B will come to know that somewhere the message was tampered. B can now notify A that it is ending the transaction. The entire decryption process is illustrated in *Figure 6*.

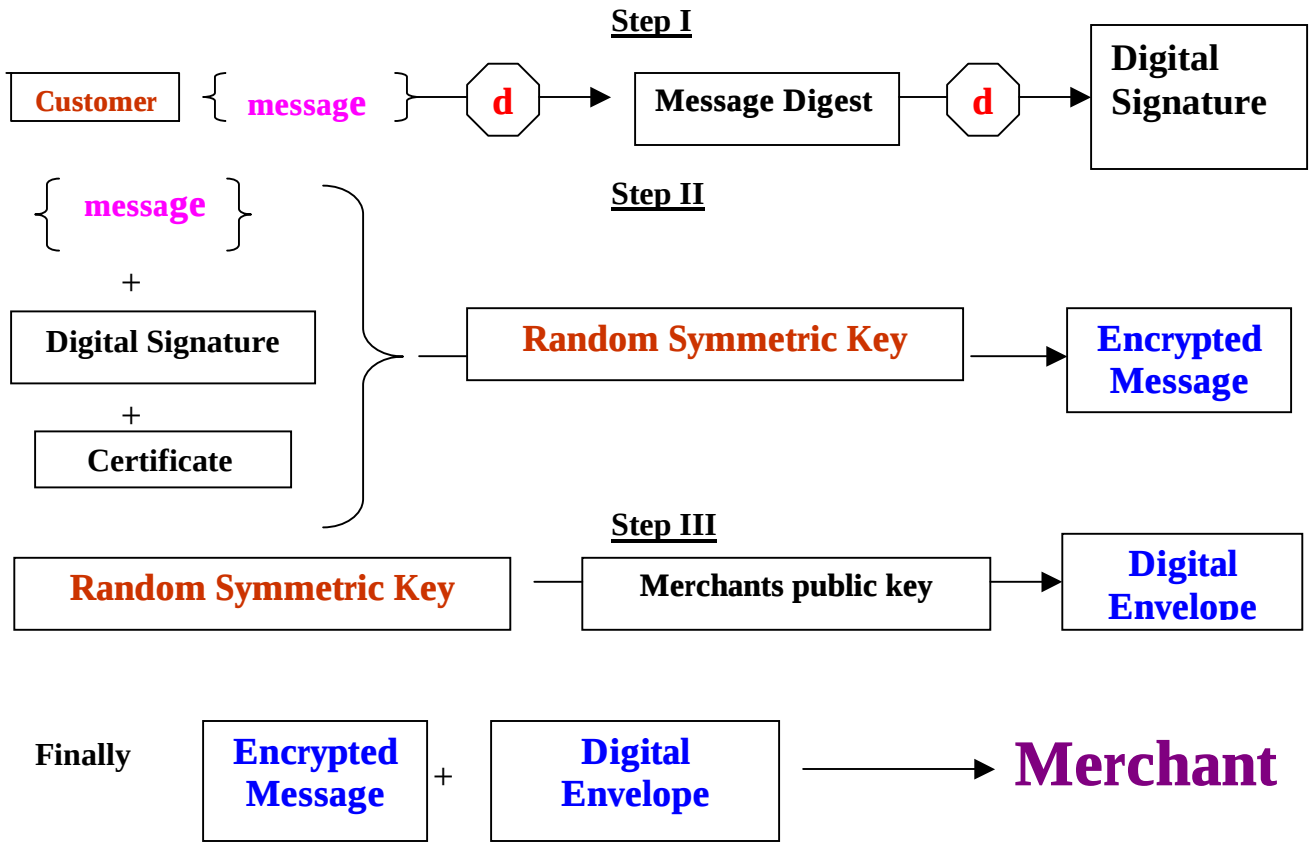


Figure 5. Encryption process in A's

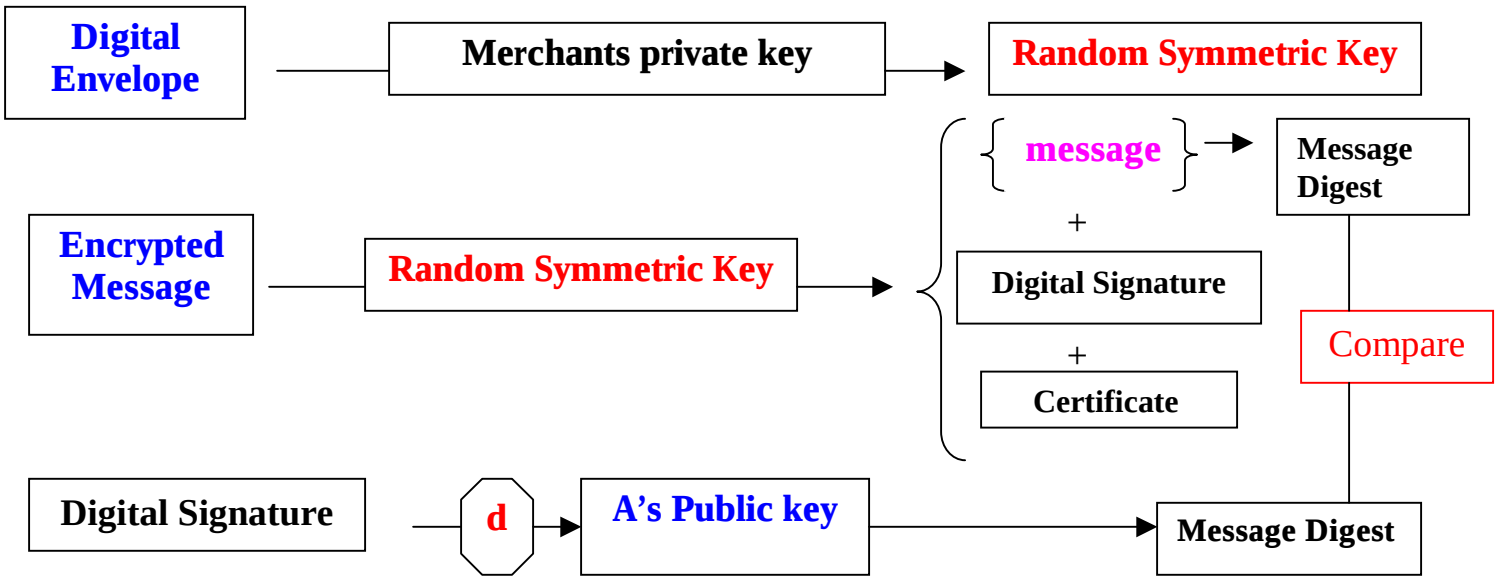


Figure 6.Decryption process in B's

3.5 Certificate Issuance

Each participating entity in the SET protocol is given a certificate that function as electronic representation of the participating entity. Certificates are given to cardholders, merchants, payment gateway, acquirers and issuers.

Cardholder Certificate:

As stated earlier, cardholder certificates function as electronic representation of the payment card. A financial institution digitally signs the cardholder certificates. This makes it impossible for a third party to alter it. Encoding the account information and a secret value through a one-way hashing function generates the cardholder certificate. This certificate is passed to merchants with purchase

requests and payment instructions. The merchant can be assured that the financial institution responsible for issuing the card has validated the account number of the cardholder.

Merchant Certificate:

This certificate is an electronic representation that the merchant has a relationship with a financial institution allowing it to accept the payment card brand. Since the financial institution digitally signs them a third party cannot alter them. These certificates are just an assurance that the merchants hold a valid agreement with an Acquirer. A merchant has at least one pair of certificates to participate in the SET environment. Basically, it has a pair of certificates for each payment card brand it accepts.

Payment Gateway Certificate:

Acquirers who process authorization and capture messages obtain payment gateway certificate. Payment gateway certificates are issued to the Acquirer by the payment brand.

Acquirer Certificate:

In order to accept and process certificate requests from merchants over public and private networks, an acquirer must have a certificate. Acquirers receive their certificates from their payment card brand.

Issuer Certificate:

Similar to the Acquirer, the Issuer possesses a certificate to accept and process certificate requests from cardholders over public and private networks. Also, they

receive the certificate from the payment card brand. If the Acquirer or the Issuer chose to have the payment card brand process the certificate requests, they will not require certificates because they are not processing SET messages. [SETBD]

3.6 Payment Processing in SET

The payment processing in SET involves the transfer of messages between the participating entities. The following messages are transferred in any SET payment transaction: Cardholder Registration, Merchant Registration, Purchase Request, payment Authorization and Payment Capture. At any point during the SET payment flow the following enquiry messages can be transferred, although they are optional: Certificate status inquiry, Purchase inquiry, Authorization reversal, Capture Reversal, Credit Reversal and Error Message. The messages that are transferred during SET payment flow are described in brief, in the following paragraphs. [SETBD].

Cardholder Registration:

Cardholders must register with a Certificate Authority (CA) before they can send SET messages to merchants. The entire registration process goes through the following steps.

1. The cardholder computer initiates the registration process by sending the **INITIATE REQUEST** message to CA. The CA sends a response to the request by sending the **INITIATE RESPONSE** to the cardholder.
2. The cardholder receives the response from the CA and requests the registration form by sending the **REGISTRATION FORM REQUEST** to the CA. The CA processes the request and sends the registration form through the **REGISTRATION FORM** message.
3. The cardholder then sends the **CARDHOLDER CERTIFICATE REQUEST** message to the CA. The CA upon receiving the request message will process that and eventually respond by sending the cardholder's certificate by sending the **CARDHOLDER CERTIFICTAE**
4. The above sequence of sending and receiving of messages ends when the cardholder receives his certificate. At any point during the transfer of messages if something is amiss, either one of the participating entities will send the **ERROR** message and the transaction will be aborted.

The transfer of many messages characterizes SET protocol. Each message has a specific format and giving the format of these messages is tedious. They are given in the SET Specification: Formal Protocol Definition [SETFPD]. The overall flow of these messages will be explained in detail.

Merchant registration:

Like cardholders, the merchants must register with a Certificate Authority (CA) before they can receive SET payment messages from cardholders or process SET

transactions through a payment gateway. The registration process starts when the merchant software requests a copy of the CA's key exchange certificate and the appropriate registration form. The entire merchant registration flow of messages is summarized below.

1. Initially, the merchant computer requests the registration form from the CA by sending the **INITIATE REQUEST.** The CA on receiving the request from the merchant processes the request and sends the registration form through the **REGISTRATION FORM** message.
2. On receiving the registration from message the merchant requests the certificate by sending the **MERCHANT CERTIFICATE REQUEST** message to the CA. The CA processes the request and generates the certificates and passes to the merchant through the **MERCHANT CERTIFICATE** messages.

Purchase Request:

The SET protocol is invoked after the cardholder has completed browsing, selection and ordering. Before the flow begins the cardholder will be presented with a completed order form and approved its contents. As is the norm, the cardholder initiates the transfer of messages by sending the request. The purchase request flow of messages is summarized below.

1. The cardholder computer sends **INITIATE REQUEST** to the merchant computer. The merchant computer on receiving the request sends its certificates through the **INITIATE RESPONSE** message.

2. The cardholder after receiving the response from the merchant will then send a **PURCASE REQUEST** message to the merchant computer. The merchant then processes the request and then sends the **PURCHASE RESPONSE** message.

Payment Authorization:

During the processing of an order from the cardholder, the merchant will have to authorize the transaction. The merchant software generates and digitally signs an authorization request, which includes the amount to be authorized. The payment authorization is a one step process. Initially the merchant computer sends the **AUTHORIZATION REQUEST** message to the payment gateway. The payment gateway processes the authorization request and sends the **AUTHORIZATION RESPONSE** to the merchant. The merchant then processes the response message to observe whether the payment has been authorized.

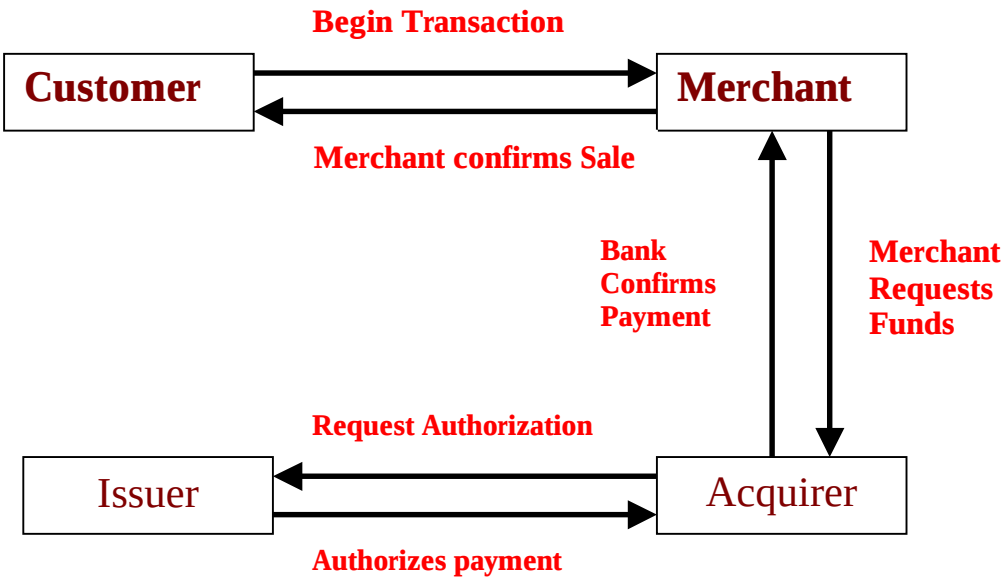


Fig 7 Merchant – Bank Transaction

Payment Capture:

After completing the processing of an order from the cardholder the merchant requests the payment. There will be often be a significant time lapse between the message requesting authorization and the message requesting payment. The flow contains only a single capture request but the merchant software is permitted to batch multiple requests into a single message. The merchant software generates and digitally signs a **CAPTURE REQUEST** and sends it to the payment gateway. The payment gateway processes the request and then sends the **CAPTURE RESPONSE** to the merchant.[SETBD]

Described above is a high level view of the exchange of messages between different participating entities in a transaction adhering to the SET specification. The message exchange is represented in Fig.7. These messages will be converted to CPAL format and then passed on to the CPALES for obtaining the Verification Condition (VC). The VC will then be analyzed to find the presence of flaws in the protocol. The conversion of these messages to CPAL and the resulting analysis will be the discussion in the next chapter.

4.CPAL Specification of the SET protocol

Our objective of this project is to gain a thorough understanding of a complex security protocol and to verify the correctness of the protocol, which directly led us to choosing SET as our main protocol of study. While implementing SET protocol in CPAL many assumptions were made regarding the representation of the protocol to keep the specification manageable and to eliminate details that did not have any effect upon the security of the protocol, given our assumptions.

4.1 Assumptions

The intended use of SET protocol is in Electronic Shopping. It is assumed that browsing and shopping, Merchant and Item Selection, Negotiation, Payment selection and Delivery of goods is not supported by SET protocol. SET supports three phases' in a transaction a) payment authorization and transport b) confirmation and enquiry and c) merchant reimbursement. To keep the project manageable, our project concentrates only on payment authorization, transport and confirmation. The reason we leave enquiry and merchant reimbursement is because they are special cases that do not occur in a normal SET transaction.

Only transactions that take place in Internet are considered. MOTO (Mail Order / Telephone Order) transactions are not considered.

4.2 CPAL specification for SET protocol

The transfer of messages in a transaction adhering to the SET protocol specification is sequential. The first step in the verification of the SET protocol is to

identify the flow of the messages between different entities and to convert them into CPAL specification. The messages that are exchanged in a SET transaction can be broadly classified into three major categories.

The three major categories of SET messages along with the list of messages in each category with their abbreviation in CPAL is listed below. The CPAL representation of these messages can be found in Appendix A.

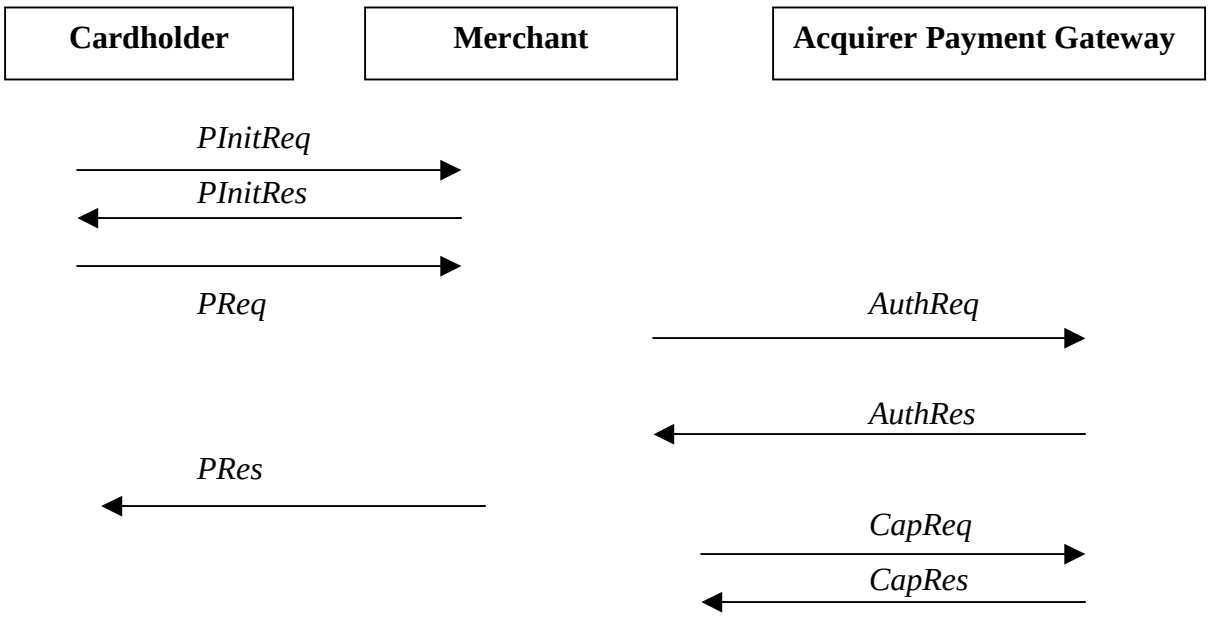
1. *Certificate Management messages*
    - Cardholder Certificate Initiation Request *(CardCInitReq)*
    - Cardholder Certificate Initiation Response *(CardCInitRes)*
    - Merchant-Acquirer Certificate Initiation *(Me-AqCInitReq)*
    - Merchant-Acquirer Initiation Response *(Me-AqCInitRes)*
    - Certificate Request Message from End-Entities (Cardholder, Merchant or Payment Gateway) *(CertReq)*
    - Certificate Response Message from Certificate Authority *(CertRes)*

2. *Cardholder-Merchant messages*
    - Purchase Initialization Request *(PinitReq)*
    - Purchase Initialization Response *(PinitRes)*
    - Purchase Request *(PReq)* & Payment Response *(PRes)*

3. *Merchant-Payment Gateway messages*

   - Authorization Request *(AuthReq)* & Authorization Response *(AuthRes)*
   - Capture Request *(CapReq)* & Capture Response *(CapRes)*
   - Credit Request *(CredReq)* & Credit Response *(CredRes)*

In this chapter, for terseness, only Cardholder-Merchant messages will be discussed. The format of these messages and the method in which these messages were converted to CPAL and verified using CPALES will be explained. The entire CPAL specification of the SET protocol is given in Appendix A. The overall transfer of messages between Cardholder/Merchant and Merchant/Payment Gateway is illustrated in Figure 7.



Legend: *PinitReq/PInitRes* → Payment Initialization Request/Response

PReq/PRes → Payment Request/ Response

AuthReq/AuthRes → Authorization Request/Authorization Response

CapReq/CapRes → Capture Request/Response

Fig 7. Cardholder/Merchant and Merchant/Payment Gateway Message Transfer

4.3 Cardholder-Merchant messages

Two pairs of messages are transferred between the cardholder and the merchant during this phase. They are the Payment Initialization Request/Response messages and the Purchase Order Request/ Response messages. Another pair of messages called the Inquiry Request/Response may be transferred but they are optional.

The process of generating the Payment Initialization request message goes through the following steps in the SET protocol.[SETPG]

1. Generate RRPID for matching the message and the matching response message.
2. Populate language of the cardholder's choice.
3. Generate LID_C, Local Identification for Cardholder
4. If Merchant has already supplied a LID_M in the SET initiation process then copy it into the message.
5. Generate a fresh Chall_C
6. Populate BIN (first 6 digits of the cardholder's account number)
7. Save RRPID, LID_C, LID_M (if available) and Chall_C
8. Invoke Compose Message Wrapper to send the message to Merchant.

These steps are defined in the SET Protocol Specification: Programmers Guide[SETPG]. The PInitReq message will now have the following components in it.

PInitReq {RRPID, Language, LID_C, LID_M, Chall_C, BrandID, BIN}

All the messages transferred under the SET protocol are wrapped using a MesageWrapper. The MessageWrrapper contains the Version of the SET protocol, the revision, the date and the Software Identification number, in addition to the message to be wrapped. So the MessageWrapper is applied to PinitReq and the Message wrapped PinitReq has the following format.

MWPInitReq {Version, Revision, Date, PinitReq, SWIdent}

The CPAL version of the message wrapped Payment Initialization Request message is given in *Figure 4.*

```
1. -- Action        : Initiate Transaction
2. -- Message       : Purchase Initialization Request (PInitReq)
3. -- Initiated by  : Customer
4. C: RRPID: = new;
5. C: Chall_C: = new;
6. C: PInitReq: = <RRPID, Language, LID_C, LID_M, Chall_C, BrandID,BIN>
7. C: TransRec: = <RRPID, LID_C, LID_M, Chall_C>;
8. C: MWPinitReq: = <Version, Revison, Date, PInitReq, XID>;
9. C: => M (MWPInitReq);
10.M: <- (MWPInitReq);
11.M: (Version, Revision, PInitReq, XID): = MWPInitReq;
12.M: (RRPID, Language, LID_C, LID_M, Chall_C, BrandID, BIN): =
              PinitReq;
```

Figure 8. CPAL specification of the PinitReq message

The first three lines are comment lines allowed by the CPAL editor to help in the better representation of the CPAL version of cryptographic protocols. Line numbers 4

and 5 represents the generation of nonces for Request Response pair and Chall_C. In line 6, the message components are concatenated and stored in a single message called PinitReq. As required by the SET specification, message components RRPID, LID_C, LID_M and Chall_C are stored in the *TransRec* of C's address space in Line 7. Since PinitReq is a SET message, it is wrapped using the MessageWrapper in line 8 and stored in *MWPInitReq.* This message is then sent from the Customer address space to the merchant address space using the secure send (=>) operator in line 9. Line 10 represents the receipt of the message *MWPInitReq* by the Merchant. Lines 11 and 12 represent the breaking of the concatenated message into its individual message components in the Merchant side. Thus, a single message called Payment Initialization request is transferred from the Customer to Merchant in accordance to the SET protocol specification and the corresponding CPAL specification the message is shown above.

In a similar manner the entire SET protocol is converted into a single CPAL specification and the resulting CPAL specification is then fed to the Cryptographic Protocol Analysis Language-Evaluation System (CPALES) to produce the Verification Condition.

After generating syntax-error free CPAL specification, the formal semantics for the protocol is automatically generated by CPAL-ES. The derivation is accomplished in two steps:

1. Each statement is applied to an initial predicate with value TRUE and beginning from the last protocol statement the control is progressed in reverse to the first statement.
2. Assumptions are applied wherever possible. The resulting predicate is scanned for conditions allowing replacement. [YAS96]

4.4 Assertions

CPAL extends the functionality of the pseudocode by allowing the participating principals to encode protocol goals and assumptions directly into the specification that the protocol analyst assumes or asserts to be true.

Now we consider how a combination of assume and assert statements are used to represent the assumptions and goals specified by the SET protocol. We consider the transfer of Cardholder Certificate Initiation Request Message for illustration.(Appendix A, 1st message). The SET protocol desires to assume that the Request/Response pair ID (RRPID) for the cardholder may be the same before and after a message wrap. The protocol analyst converting the SET protocol to CPAL specification can denote the assume statement as

```
C: assume (C.RRPID_MWREQ == C.RRPID_REQ);
```

The above statement means that in the Cardholder's address space (**C**) it is assumed that the Request/Response pair ID (RRPID) before message wrap (i.e. `C.RRPID_REQ)` and the RRPID after message wrap (i.e. `C.RRPID_MWREQ)` are the same.

The next step in the SET protocol involves having to prove that the message wrapped RRPID in the address space of Cardholder Certificate Authority is the same as the RRPID that it received from the Cardholder' CardCIntiReq message (Cardholder Certificate Initiation Request). The goal of proving this equality of the Request/Response Pair ID's is represented in the SET specification as

```
CCA: assert (RRPID_MW == RRPID_C);
```

The above CPAL statement asserts the goal that in Cardholder Certificate Authority's (CCA) address space, the message wrapped RRPID and the RRPID received from the cardholder are the same. Only after the assertion of this goal, the CCA separates the individual message components sent by the Cardholder. If the goal of RRPID equality is not met the CCA sends an error message back to the Cardholder and the transaction is halted.

The transfer of messages in SET protocol is a cycle of many request/response pairs. So for the Cardholder Certificate Initiation Request (CardCInitReq) message sent from Cardholder to CCA there will be a Cardholder Certificate Initiation Response (CardCInitRes) message sent from the CCA to the Cardholder. Among many message components that are part of this message one of them will be RRPID_RES (Request/Response Pair ID _ Response). The goal of the SET protocol is that the initial

RRPID in the Cardholder's address space before the transfer of CardCInitReq and RRPID_RES from the CardCInitRes message from the CCA should be equal. This goal is asserted in the CPAL specification as

```
C: if (RRPID_RES == RRPID) then
   {assert(RRPID_RES == RRPID);}
```

At the Cardholder's address space, the equality for RRPID_RES and RRPID are checked and if they are equal we can assert that the messages CardCInitReq and CardCInitRes were not tampered while they were sent from their respective source to the destination.

By using a combination of assume and assert statements in the CPAL specification we were able to effectively represents the assumptions and goals of the SET protocol and also assert that the goals of the protocol for each message cycles.

4.5 Proving the Verification Condition

This is the last step in the CPAL-ES process of verifying cryptographic protocols. The predicate resulting from the previous step is analyzed for repeated logical conditions that can be reduced through routine simplification. More often, the analysis will be left with more challenging predicate to prove, as was the case with the SET protocol. The analysis of these complex predicates revealed the assumptions that are necessary in order

to complete the proof. In the case of SET four assume statements were needed to prove the final verification condition to be true.

5. Conclusion

Initially the process of verifying cryptographic protocols using the powerful tool called CPAL-ES was understood by proving relatively smaller and simpler cryptographic protocols. SET protocol specification was then analyzed and understood. Initial steps in the understanding of the protocol was the grouping of messages that are transferred between different participating entities in the SET protocol based transaction and identifying and understanding each individual message components. Later on, the daunting task of converting the entire SET protocol specification into a single CPAL document was accomplished. The CPAL specification was then analyzed using the CPAL-ES environment, which gave the Verification Condition. Further analysis of the Verification Condition resulted in the simplified predicate to be TRUE in the end meaning that the SET protocol is verified. Since the SET protocol specification is inherently large, we have made several assumptions during the conversion of the SET specification into a CPAL document.

By converting the SET protocol into a CPAL specification and verifying against the CPAL-ES environment, we are able to assert the correctness of the SET protocol, given our assumptions (4.1). Asserting the correctness of the SET protocol increases the credibility of the protocol.

Bibliography:

- **[YAS96]** Alec F Yasinsac, “A Formal Semantics for Evaluating Cryptographic Protocols”, University of Virginia, January 1996.

- **[YW99]** Alec F Yasinsac and William A Wulf, “ A Framework for A Cryptographic Protocol Evaluation Workbench”, Proceedings of the Fourth IEEE International High Assurance Systems Engineering Symposium (HASE99), Washington D.C., Nov. 1999

- **[YW96]** Alec F Yasinsac and William A Wulf, ”Using Weakest Precondition to Evaluate Cryptographic Protocols”, Cambridge International Workshop on Cryptographic Protocols, March 1996

- **[SETPG]** “SET Secure Electronic Transaction Specification: A Programmer’s Guide”, **http://www.setco.org/**

- **[SETBD]** “SET Secure Electronic Transaction Specification: Business Description”, **http://www.setco.org/**

- **[JUS00]** Justin Childs,” Evaluating the TLS Family of Protocols with Weakest Precondition Reasoning", Dept. Of Computer Science, Florida State University, June2000.

- **[JUS01]** Justin Childs and Alec Yasinsac, "Using Weakest Preconditions to Evaluate the Transport Layer Security Protocol", The Sixth IEEE International Symposium on High Assurance Systems Engineering, Boca Raton, FL, Oct 24-26, 2001

- **[YR01]** Alec F Yasinsac and Michael P Runy, "The Weakest Precondition Protocol Analysis Environment", TR 010502 Dept. Of Computer Science, Florida State University, June2001

- **[WEBC]** Web Based Education Commission . The Power of the Internet for Learning: Moving from Promise to Practice. **http://www.ed.gov/pubs/edpubs.html**

- **[RS]** The Future of E Commerce. INTERNET-DRAFT. **http://www.iboost.com/manage/business/trends/20024.htm**

- **[TLS]** The TLS Protocol Version 1.0 RFC 2246

- **[SSL]** The SSL Protocol Version 3.0, Transport Layer Security Working Group, Netscape Communications

- **[SP01]** Security On The Internet, The Froehlich/Kent Encyclopedia of Telecommunications vol. 15. Marcel Dekker, New York, 1997, pp. 231-255. **http://www.cert.org/encyc_article/tocencyc.html**

- **[YAS00]** Alec Yasinsac, "Dynamic Analysis Of The Security Protocols", Proceedings of the New Security Paradigms Workshop 2000, pp 77-87

- **[YW93]** Alec Yasinsac and William A Wulf, "Evaluating Cryptographic Protocols", Univ. Of Virginia, Technical Report #CS-93-53, August 1993

- **[SS99]** Strand Spaces: Proving Security Protocols, Appears in Journal Of Computer Security, 7 (1999)

Appendix A: CPAL specification of the SET protocol

```
-- Certificate Management in SET
-- Action         : Generate Certificate Initiation Request
-- Message        : Cardholder Certificate Initiate Request
-- Initiated by   : Cardholder

C: assume (C.RRPID_MWREQ == C.RRPID_REQ);
C: RRPID := new;
C: LID_E := new;
C: Chall_EE := new;
C: CardCInitReq := <RRPID,LID_EE, Chall_EE,BrandID>;
C: MWCardCInitReq :=
<Version,Revision,Date,RRPID,SWIdent,CardCInitReq,XID>;
C: => CCA(MWCardCInitReq);
CCA: <- (MWCardCInitReq);
CCA: (Version, Revision, Date, RRPID_MW, SWIdent, CardCInitReq, XID) :=
MWCardCInitReq;
CCA: (RRPID_C, LID_EE, Chall_EE, BrandID) := CardCInitReq;
CCA: TransRec := <RRPID, LID_EE, Chall_EE, BrandID>;
CCA: assert (RRPID_MW == RRPID_C);

-- Action         : Generate Certificate Initiation Response
-- Message        : Cardholder Certificate Initiate Response
-- Initiated by   : Cardholder

CCA: LID_CA := new;
CCA: CardCInitResData := <TransRec, LID_CA,CAEThumb>;
CCA: CardCInitRes := S (CCA,CardCInitResData);
CCA: MWCardCInitRes :=
<Version,Revision,Date,RRPID_C,SWIdent,CardCInitRes,XID>;
CCA: => C (MWCardCInitRes);
C: <- (MWCardCInitRes);
C: (Version,Revision,Date,RRPID_RES,SWIdent,CardCInitRes,XID) :=
MWCardCInitRes;
C: (RRPID_C,LID_EE,Chall_EE,LID_CA,CAEThumb) := CardCInitRes;
C: assert(RRPID_MWREQ == RRPID_REQ);
CCA: TransRec := <RRPID_REQ,RequsetType,LID_EE,Chall_EE,LID_CA>;
CCA: Chall_CA := new;
CCA: RegTemplate := <RegFormID,BrandLogoURL,CardLogoURL,RegFieldSeq>;
CCA: ReferralData:= <Reason,ReferralURLseq>;
CCA: RegFormData := <RegTemplate,PolicyText>;
CCA: RegFormOrReferral := <RegFormData,ReferralData>;
CCA: RegFormResTBS :=
<RRPID,LID_EE,Chall_EE,LID_CA,Chall_CA,RequestType,RegFormORReferral>;
CCA: RegFormRes := S(CA,RegFormResTBS);
CCA: MWRegFormRes :=
<Version,Revision,Date,RRPID_RES,SWIdent,RegFormRes,XID>;
CCA: => C(MWRegFormRes);
C: <- (MWRegFormRes);
C: (Version,Revision,Date,RRPID_RES,SWIdent, RegFormRes,XID) :=
MWRegFormRes;
C:
(RRPID_R,LID_EE,Chall_EE,LID_CA,Chall_CA,RequestType,RegFormORReferral)
:= RegFormRes;
C: if(RRPID_RES == RRPID) then
```

```
     {assert(RRPID_RES == RRPID);}

-- Action  : Merchant-Acquirer Certificate Initiation Processing
-- Message : Me-AqCInitReq
-- Initiated By : Merchant-Acquirer

M: RRPID := new;
M: LID_EE := new;
M: Chall_EE := new;
M: MerchantAcqId := <MerchantBIN,MerchantID>;
M: AcqId := <AcqBIN,AcqBusinessID>;
M: IDData := <MerchantAcqId,AcqId>;
M: Me_AqCInitReq :=
<RRPID,LID_EE,Chall_EE,RequestType,IDData,BrandID,Language>;
M: MWMe_AqCInitReq :=
<Version,Revision,Date,RRPID,SWIdent,Me_AqCInitReq,XID>;
M: => CCA(MWMe_AqCInitReq);
CCA: <- (MWMe_AqCInitReq);
CCA: (Version,Revision,Date,RRPID_MW,SWIdent,Me_AqCInitReq,XID) :=
MWMe_AqCInitReq;
CCA: (RRPID_MCA,LID_EE,Chall_EE,RequestType,IDData,BrandID,Language) :=
Me_AqCInitReq;
CCA: assert(RRPID_MW == RRPID_MCA);
CCA: TransRec := <RRPID,LID_EE,Chall_EE,BrandID,IDData,Language>;

-- Action : Response for the Me_AqCertificate Initiation message
--          from Merchant or Acquirer
-- Message: Me_AqCInitRes
-- Initiated by : Certificate Authority (CA)

CCA: Chall_CA := new;
CCA: Me_AqCInitResData :=
<RRPID,LID_EE,Chall_EE,Chall_CA,ReqType,RegFormOrReferral>;
CCA: Me_AqCInitRes := S(CA,Me_AqCInitResData);
CCA: MWMe_AqCInitRes :=
<Version,Revision,Date,RRPID,SWIdent,Me_AqCInitRes,XID>;
CCA: => M(MWMe_AqCInitRes);
M: <- (MWMe_AqCInitRes);
M: (Version,Revision,Date,RRPID_MW,SWIdent,Me_AqCInitRes,XID) :=
MWMe_AqCInitRes;
M: (RRPID_MCARes,LID_EE,Chall_EE,Chall_CA,ReqType,RegFormOrReferral) :=
Me_AqCInitRes;
M: if(RRPID_MCARes == RRPID) then {
     assert(RRPID_MCARes == RRPID);
    }

-- Action : Certificate request message from End Entities
--          (Cardholder,Merchant or Payment Gateway)
-- Message: CertReq
-- Initiated By : End Entities

EE: RRPID := new;
EE: PANData := <PAN,CardExpiry,CardSecret,EXNonce>;
EE: AcctData:= <AcctIdentification,EXNonce>;
EE: AcctInfo := <PANData,AcctData>;
```

```
EE: CertReqData :=
<RRPID,LID_EE,Chall_EE,LID_CA,Chall_CA,ReqType,ReqDate,IDData,RegFormID
>;
EE: MWCertReqData := <Version,Revision,Date,CertReqData,XID>;
EE: => CCA(MWCertReqData);
CCA: <- (MWCertReqData);
CCA: (Version,Revision,Date,CertReqData,XID) := MWCertReqData;
CCA:
(RRPID,LID_EE,Chall_EE,LID_CA,Chall_CA,ReqType,ReqDate,IDData,RegFormID
) := CertReqData;

-- Action : Certificate Response Message from Cert Authority
-- Message: CertRes
-- Initiated By: Certificate Authority

CCA: CaMsg := <CardLogoURL,BrandLogoURL,Currrency,CardHolderMsg>;
CCA: CertStatus := <CertStatusCode,EEMessage,CaMsg>;
CCA: CertResData := <RRPID,LID_EE,Chall_EE,LID_CA,CertStatus>;
CCA: CertRes := <S(CA,CertResData),Enc(CAKeyData,CA,CertResData)>;
CCA: MWCertRes := <Version,Revision,Date,RRPID,SWIdent,CertRes,XID>;
CCA: => EE(MWCertRes);
EE: <- (MWCertRes);
EE: (Version,Revision,Date,RRPID_CRes,SWIdent,CertRes,XID) :=
MWCertRes;
EE: (RRPID_CRData,LID_EE,Chall_EE,LID_CA,CertStatus) := CertResData;
EE: if(RRPID == RRPID_CRes) then {
       assert(RRPID == RRPID_CRes);
     }


-- Action        : Initiate Transaction
-- Message       : Purchase Initialization Request (PInitReq)
-- Initiated by  : Customer

C: Chall_C := new;
C: PInitReq := <RRPID,Language,LID_C,LID_M,Chall_C, BrandID, BIN>;
C: TransRec := <RRPID,LID_C,LID_M,Chall_C>;
C: MWPinitReq := <Version, Revison, Date, PInitReq, XID>;
C: => M(MWPInitReq);
M: <- (MWPInitReq);
M: (Version,Revision, PInitReq, XID) := MWPInitReq;
M: (RRPID,Language,LID_C,LID_M,Chall_C,BrandID,BIN) := PInitReq;

-- Action        : Response to PInitReq
-- Message       : Purchase Initialization Response (PInitRes)
-- Initiated by  : Merchant

M: TransRec := <RRPID,Language,LID_C,LID_M,Chall_C,BrandID,BIN>;
-- Need to give an error message when LID_M is not matched
M: XID := new;
M: Chall_M := new;
M: PInitResData := <TransRec,XID,Chall_M,BrandCRLIdentifier,PEThumb>;
-- BrandCRLIdentifier and Cert_PE are to be processed in the
Certification
-- Generation Process
M: SPInitResData := S(M,PInitResData);
-- S(M,PInitResData) --> SignedData(M,PInitResData) in PKCS #7 format
```

```
M: MWSPInitResData := <Version,Revision,Data,SPInitResData,XID>;
M: => C(MWSPInitResData);
C: <- (MWSPInitResData);

-- Action       : Check the received Response for PInitReq
-- Performed By : Customer

C: assume(C.LID_C == C.TLID_C);
C: assume(C.LID_M == C.TLID_M);
C: assume(C.RRPID == C.TRRPID);
C: (Version,Revision,SPInitResData,XID) := MWSPInitResData;
-- Need to perform Receive SignedData
C: (M,PInitResData) := SPInitResData;
C: (TransRecRes,XID,Chall_M,BrandCRLIdentifier,PEThumb) :=
PInitResData;
C: (RRPID,Language,LID_C,LID_M,Chall_C,BrandID,BIN) := TransRecRes;
C: (TRRPID,TLID_C,TLID_M,TChall_C) := TransRec;
C:if (LID_C == TLID_C) then {
     if (LID_M == TLID_M) then {
        if (RRPID == TRRPID) then {
          assert(LID_C == TLID_C);
          assert(LID_M == TLID_M);
          assert(RRPID == TRRPID);
        }else
          {assert(error);}
     }
   }

-- Action        : Purchase Message
-- Message       : Purchase Request (PReq)
-- Initiated by  : Customer

C: OIData := <TransID,RRPID,Chall_C,HOD,Chall_M,BrandID,BIN>;
C: RRPID := new;
C: ODSalt := new;
C: HODInput := <OD,PAmt,ODSalt>;
C: HOD := DD(HODInput);
C: Inputs := <HOD,PAmt>;
C: PIHead := <TransID,Inputs,MerchantID,InstallRecurInd,SWIdent>;
C: PANData := <PAN,CardExp,PANSecret,EXNonce>;
C: PANToken := <PAN,CardExp,EXNonce>;
C: PIData := <PIHead,PANData>;
C: OIDualSigned := L(PIData,OIData);
C: PReqDualSigned := <PIDualSigned,OIDualSigned>;
C: => M(PReqDualSigned);
M: <- (PReqDualSigned);

-- Action       : Check the Purchase request from the customer
-- Performed By : Merchant

M: (PIDualSigned,OIDualSigned) := PReqDualSigned;


-- Action       : Payment response
-- Message      : PRes
-- Performed by : Merchant
```

```
M: Results := <AcqCardMsg,AuthStatus,CapStatus,CredStatusReq>;
M: PResPayload := <CompletionCode,Results>;
M: PResData :=
<TransID,RRPID,Chall_C,BrandCRLidentifier,PResPayloadSeq>;
M: PRes := S(M,PResData);
M: => C(PRes);
C: <- (PRes);
C: (TransID,RRPID_PRes,Chall_C,BrandCRLidentifier,PResPayloadSeq) :=
PResData;
C: if(RRPID == RRPID_PRes) then {
     assert (RRPID == RRPID_PRes);
     }

-- Merchant-Payment Gateway Message Transfer
-- Action        : Authorization Request
-- Message       : AuthReq
-- Initiated by  : Merchant

M: MerchData := <MerchCatCode,MerchGroup>;
M: AuthReqPayload := <SubAuthInd,AuthReqAmt,ReqCardtype,MerchData>;
M: MerTermsID := <MerchantID,TerminalID,AgentNum>;
M: RRPID := new;
M: RRTags := <RRPID,MerTermsID,Date>;
M: TransIDs := <LID_C,LID_M,XID,PReqDate>;
M: AuthTags := <RRTags,TransIDs,AuthRetNum>;
M: AuthReqItem := <AuthTags,AuthReqPayload>;
M: AuthReqData := <AuthReqItem,CaptureNow,SaleDetail>;
M: OIData := <TransID,RRPID,Chall_C,HOD,Chall_M,BrandID,BIN>;
M: PIHead := <TransID,Inputs,MerchantID,InstallRecurInd,SWIdent>;
M: PANData := <PAN,CardExp,PANSecret,EXNonce>;
M: PANToken := <PAN,CardExp,EXNonce>;
M: PIData := <PIHead,PANData>;
M: PI := <PIData,OIData>;
M: AuthReq := Enc(M,P,AuthReqData,PI);
M: MWAutheReq := <Version,Revision,Date,RRPID,SWIdent,AuthReq,XID>;
M: => PG(MWAuthReq);
PG: <- (MWAuthReq);

-- Action        : Authorization Response
-- Message       : AuthRes
-- Initiated by  : Payment Gateway
PG: (Version,Revision,Date,RRPID,AuthReq,AuthReq,XID) := MWAuthReq;
PG: (AuthReqData,PIData,OIData) := AuthReq;
PG: (PIHead,PANData) := PIData;
PG: (TransID_PI,Inputs,MerchantID,InstallRecurInd,SWIdent) := PIHead;
PG: (TransID_OI,RRPID_OI,Chall_C,HOD,Chall_M,BrandID,BIN) := OIData;
PG: if (TransID_OI == TransID_PI) then
       { assert(TransID_OI == TransID_PI); }
PG: if(RRPID == RRPID_OI) then
     { assert(RRPID == RRPID_OI);}
PG: ResData := <AuthVal,CardType>;
PG: AuthHeader := <AuthAmt,AuthCode,ResData,BatchStatus,CurrConv>;
PG: PANToken := <PAN,CardExp,EXNonce>;
PG: RRTags := <RRPID,MerTermsID,Date>;
PG: AuthTags := <RRTags,TransIDs,AuthRetNum>;
PG: CheckDg := <HOIData,HOIHead>;
PG: TransIDs := <LID_C,LID_M,XID,PReqDate>;
```

```
PG: AuthResData := <AuthTags,AuthResPayload>;
PG: AuthRes := Enc(PG,M,AuthResData,PANToken);
PG: MWAuthRes := <Version,Revision,Date,RRPID,SWIdent,AuthRes,XID>;
PG: => M(MWAuthRes);
M: <- (MWAuthRes);
M: (Version,Revision,Date,RRPID,SWIdent,AuthRes,XID) := MWAuthRes;
M: (AuthResData,PANToken) := AuthRes;
M: (AuthTags,AuthResPayload) := AuthResData;
M: (RRTags,TransIDs,AuthRetNum) := AuthTags;
M: (RRPID_Res,MerTermsID,Date) := RRTags;
M: if(RRPID_Res == RRPID) then
     { assert(RRPID_Res == RRPID);}

-- Action        : Capture Request
-- Message       : CapReq
-- Initiated by  : Merchant

M: CapPayload := <CapDate,CapreqAmt,AuthReqAmt,AuthResPayload>;
M: TransIDs := <LID_C,LID_M,XID,PReqDate>;
M: CapItem := <TransID,AuthRRPID,CapPayload>;
M: RRPID := new;
M: RRTags := <RRPID,MerTermID,Date>;
M: PANToken := <PAN,CardExp,EXNonce>;
M: CapReqData := <RRTags,CapItemSeq>;
M: CapReq := Enc(M,PG,CapReqData,PANToken);
M: MWCapReq := <Version,Revision,Date,RRPID,SWIdent,CapReq,XID>;
M: => PG(MWCapReq);
PG: <- (MWCapReq);
PG: (Version,Revision,Date,RRPID,SWIdent,CapReq,XID) := MWCapReq;
PG: (CapReqData,PANToken) := CapReq;
PG: (RRTags,CapItemSeq) := CapReqData;
PG: (RRPID_CReq,MerTermID,Date) := RRTags;
PG: (TransID,AuthRRPID,CapPayload) := CapItem;
PG: (LID_CReq,LID_M,XID_CReq,PReqDate) := TransID;
PG: if (RRPID_CReq == RRPID) then {
       if (XID == XID_CReq) then {
          if (LID_C == LID_CReq) then {
             assert (RRPID_CReq == RRPID);
             assert (XID == XID_CReq);
             assert (LID_C == LID_CReq);
          }
       }
     }

-- Action        : Capture Response
-- Message       : CapRes
-- Initiated by  : PaymentGateway

PG: CapResPayload := <CapCode,CapAmt,BatchID,BatchSeqNum>;
PG: TransID := <LID_C,LID_M,XID,PReqDate>;
PG: RRPID := new;
PG: RRTags := <RRPID,MerTermID,Date>;
PG: CapResItem := <TransID,AuthRRPID,CapResPayload>;
PG: CapResData := <RRTags,CapResItemSeq>;
PG: CapRes := Enc(PG,M,CapResData);
PG: MWCapRes := <Version,Revision,Date,RRPID,SWIdent,CapRes,XID>;
PG: => M(MWCapRes);
```

```
M: <- (MWCapres);
M: (Version,Revision,Date,RRPID,SWIdent,CapRes,XID) := MWCapRes;
M: (RRTags,CapResItemSeq) := CapResData;
M: (RRPID_CRes,MerTermID,Date) := RRTags;
M: (TransID,AuthRRPID,CapResPayload) := CapResItem;
M: if (RRPID_CRes == RRPID) then {
     if (LID_MCRes == LID_M) then {
        assert (RRPID_CRes == RRPID);
        assert (LID_MCRes == LID_M);
        }
     }

-- Action        : Credit request
-- Message       : CredReq
-- Initiated by  : Merchant

M: CredReqPayload := <CredCode,CredAmt,BatchID,BatchSeq>;
M: PANToken := <PAN,CardExp,EXNonce>;
M: CredReq := Enc(M,PG,CredReqPayload,PANToken);
M: MWCredReq := <Version,Revision,Date,RRPID,SWIdent,CredReq,XID>;
M: => PG(MWCredReq);
PG: <- (MWCredReq);
PG: (Version,Revision,Date,RRPID_CrReq,SWIdent,CapRes,XID) :=
MWCredReq;
PG: if(RRPID_CrReq == RRPID) then {
       assert(RRPID_CrReq == RRPID);
     }

-- Action        : Credit Response
-- Message       : CredRes
-- Initiated by  : Payment Gateway

PG: CredResData := <CredCode,CredAmt,BatchID,BatchSeq>;
PG: CredRes := Enc(M,PG,CredResData);
PG: MWCredRes := <Version,Revision,Date,RRPID,SWIdent,CredRes,XID>;
PG: => M(MWCredRes);
M: <- (MWCredRes);
M: (Version,Revision,Date,RRPID_CrRes,SWIdent,CredRes,XID) :=
MWCredRes;
M: (CredCode,CredAmt,BatchID,BatchSeq) := CredResData;
M: if(RRPID_CrRes == RRPID) then {
     assert(RRPID_CrRes == RRPID) ;
     }
```