A Formal Specification of Requirements for Payment Transactions in the SET Protocol

Catherine Meadows and Paul Syverson

Center for High Assurance Computer Systems
Naval Research Laboratory
Washington DC 20375, USA
{meadows,syverson}@itd.nrl.navy.mil

Abstract. Payment transactions in the SET (Secure Electronic Transaction) protocol are described. Requirements for SET are discussed and formally represented in a version of NPATRL (the NRL Protocol Analyzer Temporal Requirements Language). NPATRL is language for expressing generic requirements, heretofore applied to key distribution or key agreement protocols. Transaction vectors and other new constructs added to NPATRL for reasoning about SET payment transactions are described along with properties of their representation.

1 Introduction

The SET Protocol [5] is a protocol sponsored by major credit card companies and others that is intended to provide a standard for safe, secure credit card transactions over the Internet. ('SET' stands for 'Secure Electronic Transaction'.) As such, it is intended to supply an electronic version of the paper system that exists today. However, there are a number of risks connected with use of the Internet that do not arise in the paper world, or at least are not considered as severe. These arise from the difficulty of identifying participants in transactions and the difficulty of ensuring the private information sent over the Internet remains so. SET is intended to reduce these risks by introducing cryptographic means to protect sensitive information such as credit card numbers and to provide authentication of parties involved in a credit card transaction.

SET is a complex protocol. This has caused a certain amount of concern, since it is well known that even simple cryptographic protocols can have subtle flaws that can go undetected for a long time. This realization that correct protocols are difficult to write has led to an increasing amount of work in the application of formal methods to the analysis of cryptographic protocols, with some notable successes. However, most of this work has concentrated on key distribution protocols, which are intended to distribute keys among parties for secure communication. A key distribution protocol is required to safeguard the secrecy of the key and provide authentication of the key, that is, provide a secure binding between the key and the parties that are intended to use the key to communicate. But protocols such as SET have much more complex requirements than do key distribution protocols. A key distribution protocol provides

authentication and secrecy for an atomic object: the key. SET is intended to provide secrecy for the credit card number, which may be considered an atomic object, but the entity authenticated is the credit card transaction itself, which evolves over time. The issue is complicated further by the fact that not all components of the transaction may be known to all parties, even after they have been established. For example, the credit card number, which is an integral part of the transaction, may never be revealed to the merchant.

In this paper, we show how a requirements language developed for the NRL Protocol Analyzer, a formal tool for the analysis of cryptographic protocols, can be used for specifying the security requirements of a complex protocol such as SET. This is part of ongoing work on use of the NRL Protocol Analyzer for the specification and analysis of the SET protocol.

We have found that the flexibility of our simple temporal language has allowed us to specify requirements for the authentication of an evolving transaction without any major modification to the language itself. This is because the language is used to define correctness simply in terms of what events must precede others. The content of the events is left to the specification writer, and in this case we have let them be the components of the transaction.

The remainder of the paper is organized as followed. In Section 2 we describe the SET protocol. In Section 3 we describe the NRL Protocol Analyzer and its associated requirements language. In Section 4 we describe our specification of the SET requirements. In Section 5 we compare our work with others. Section 6 concludes the paper.

2 The SET Protocol

A payment transaction in the SET protocol involves three parties: a customer, a merchant, and an application payment gateway. The customer presents a purchase request to the merchant, which includes credit card information and a proposed purchase amount. The purchase request is identified with a transaction ID. The merchant then passes the request along to the gateway, together with a request that a certain amount (not necessarily equal to the purchase amount) be authorized. The gateway then checks the customer's credit, authorizes a certain amount, and passes this information back to the merchant. The merchant passes this information back to the customer. Either at the same time as the authorization request, or later, the merchant presents a capture request to the gateway for the same transaction, requesting that a certain amount of money be captured. The gateway approves a certain amount which may or may not be equal to the amount requested. The merchant then passes this information back to the customer.

The authentication structure of the SET protocol is complex. Messages between merchant and gateway are always authenticated using digital signatures, as are messages from the merchant to the customer. Digital signatures for authentication for messages from customer to merchant are optional, although they may be made mandatory by a particular application. However, it is in the au-

thentication of forwarded messages that the structure really becomes interesting. The message from customer to merchant includes information that is needed by the gateway but may be hidden from the merchant, such as credit card number (PAN, i.e., Primary Account Number) and expiration date. Also included, when customer digital signatures are used, is a data item called the PANSecret, which is known only by the customer and gateway, but not the merchant. This is not available when customer digital signatures are not used, since it is generated as part of the certificate registration process. This information is protected by the use of a *dual signature*. Two hash functions are computed, one over the the data to be kept hidden from the merchant, and the other over the data to be revealed to it, which includes a hash over the purchase amount and order description that the customer and merchant agreed to offline. The hidden data is encrypted using the gateway's public key. The customer then computes a digital signature (if customer signatures are used) over the two hashes. The signature, the two hashes, and the encrypted and unencrypted information are sent to the merchant. The merchant verifies the signature and forwards the information, including the signature if any, to the gateway. When the gateway receives the message, it verifies the signature, if any, and also verifies the PAN and PANSecret. Whether or not signatures are used, it also verifies the PAN and and the customer's portion of the PANSecret (if any) with the credit card issuer, although this may be done offline.

Authentication of gateway to customer via the merchant is much simpler: there is none. Any information from the gateway that the merchant passes on to the customer is authenticated only by the merchant's signature.

There are also a number of options available. A customer has the option of sending an initialization message prior to its purchase request, which allows it to obtain more up-to-date certificates from the merchant, and allows the merchant to send back a random challenge which it can use to verify the freshness of the customer's subsequent purchase request. When an initialization message is sent, the transaction ID is jointly created by customer and merchant. When no initialization message is sent, the customer may create the transaction ID, or it may be jointly created by the customer and merchant. The gateway also has the option, depending upon the policy followed, of sending the customer's PAN to the merchant (the PANSecret, however, is never sent). There are also protocols for inquiring about the status of an order, cancelling an order, etc.

For the purposes of this paper, we will make some assumptions to simplify our discussion. We will assume that authorization and capture are requested in the same message and granted in the same message. We will ignore the optional protocols for inquiring about the status of an order, cancelling an order, and so on. We will also assume that the customer alone generates the transaction ID when no initialization message exists. This will allow us to assume that the customer knows the entire transaction ID when it sends the purchase request, and so simplify our analysis. However, we will consider the customer options of signing or not signing the purchase request message, and sending or not sending the initialization messages, as well as the gateway option of sending or not send-

ing the PAN to the merchant, since these are directly relevant to the security of the central payment protocol. But we will assume that the gateway will not send a customer's PAN to the merchant if it allows unsigned purchase requests on that PAN. This is to foil the obvious attack in which the gateway sends the PAN to a dishonest merchant, and that merchant uses the PAN to impersonate the customer in an unsigned purchase request.

3 The NRL Protocol Analyzer and its Requirements Language

3.1 The NRL Protocol Analyzer Model

In the NRL Protocol Analyzer (NPA), protocols are modeled in terms of communicating state machines. Each state machine represents an "honest" participant of the protocol, that is one that obeys the rules of the protocol. The state machines send messages across a network that is controlled by a hostile intruder that can read all traffic, modify traffic, create messages, and normally perform all operations that are available to a legitimate user of the system. The intruder is not itself modeled as a communicating state machine but is identified with the network—likewise all "dishonest" nodes that are assumed to be in cooperation with, and thus identified with, the intruder. In particular, any word that would be available to a dishonest node, such as master keys belonging to that node, or random numbers generated by that node, are assumed to be known by the intruder.

A state in the NRL Protocol Analyzer model consists of three things. The first is the set of local state variable values of the honest nodes. The second is the set of words known by the intruder. These consist of all messages that have been passed by legitimate parties, words created by the intruder's performing operations on messages and words, and words that were initially known by the intruder. The third is the sequence of state transitions that have occurred, where each state transition involves some combination of the sending and receiving of messages (the intruder's operating on a set of words counts as its sending messages to itself) and the assignment of values to local state variables.

NPA works by having the user specify an insecure state. The Analyzer works backwards from that state and attempts to show that every path to it begins in an unreachable state. If it succeeds, then it has proved that the path is unreachable, given the assumptions of the Analyzer model. If on the other hand it finds a path that begins in an initial state, it may have found an attack on the protocol. The Analyzer includes inductive techniques for proving that infinite classes of states are unreachable. These can be used to prove lemmas about unreachability of infinite classes of states that can be used to assist the Analyzer in its search.

The NRL Protocol Analyzer makes very simple assumptions about the strengths of the crypto-algorithms involved. Cryptographic algorithms are modeled as operations which may obey certain algebraic properties, such as the fact that encryption and decryption with the same key cancel each other out. However, more

subtle properties of cryptographic algorithms are usually not modeled, and notions relying on probability theory or complexity theory, such as polynomial indistinguishability, are completely beyond it. Thus, the assurance it gives is based on fairly strong assumptions about the cryptographic algorithms used. However, since many protocol failures have been shown to arise even when the cryptographic algorithms used behave perfectly, it remains a valuable tool for reasoning about security at the protocol level.

3.2 NPATRL (The NRL Protocol Analyzer Temporal Requirements Language)

Our language, NPATRL (pronounced N-patrol) contains a denumerable collection of constant singular terms, typically represented by letters from the beginning of the alphabet. We also have a denumerable collection of variable terms, typically represented by letters from the end of the alphabet. We also have, for each $n \geq 1$, n-ary function letters taking terms of either type as arguments and allowing us to build up functional terms in the usual recursive fashion. (We will always indicate whether a term is constant or variable if there is any potential for confusion.) We have a denumerable collection of n-ary action symbols for each arity $n \geq 1$. These will be written as words in typewriter script (e.g., `accept`). The first argument of an action symbol is reserved for a term representing the agent of the action in question. An atomic formula consists of an n-ary action symbol, e.g., '`act`' followed by an n-tuple of terms. We have the usual logical connectives: $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$, and also one temporal operator: $\diamondsuit$, which stands for "happened previously". Complex formulae are built up from atomic formulae in the usual recursive fashion. (Note that this is only a formal language, not a logic; hence there are no axioms or inference rules.) We also include quantifiers. Earlier version of NPATRL did not use them, but we have found that the introduction of projections which could be defined over a number of possible vectors necessitated their use for the SET requirements specification, and so we now include them in the language.

In general, an action symbol will be of the following form. It will have four arguments, the first representing the agent of the action in question, the second representing the other principals involved in the action, the third representing the words involved in the action, and the fourth representing the local round number of the agent of the action, where a round number local to a principal identifies all actions pertaining to a single session as far as that principal is concerned. Action symbols can describe such events as a principal sending a message, the learning of a word by the intruder, or a principal's making a change to one or more of its local state variables. An action symbol may map to more than one event, and for a given event, there may be more than one action symbol mapping to it. Requirements are stated in terms of conditions on traces of action symbols. For example, we may require that an event indicated by an action symbol can only take place if some event indicated by another action symbol has taken place previously.

For example, suppose that we wish to require that a party A accept a key as good for a session with another party B only if that key was sent by a key server. This would be done as follows:

$$\texttt{accept}(\mathit{principal}(\texttt{A},[\mathit{honest}]),\mathit{principal}(\texttt{B},[\texttt{X}]),[\texttt{KEY}],\texttt{N?})$$
$$\rightarrow \Diamond\texttt{send}(\mathit{server}(\texttt{S}),(\mathit{principal}(\texttt{A},[\mathit{honest}]),\mathit{principal}(\texttt{B},[\texttt{X}])),[\texttt{KEY}],\texttt{N?})$$

Note that the $N?$ is a 'wild-card' symbol. Thus, when $N?$ is used in more than one place, it does not necessarily meant that the two round numbers are the same. Instead, it means that we do not care what the round numbers are.

There are several types of action symbols that we use: these are `receive`, `accept`, `send`, `request`, `learn`, and `compromise`. A `receive` event is one in which a party receives a message. An `accept` event is one in which a party accepts a message as genuine. A `send` event is one in which a party sets in motion a chain of events in which a message will be sent to another party. A `learn` event in one in which the intruder learns a word. A `compromise` event is an event in which a secret such as a session key is compromised and made available to the intruder.

The interpretation of `receive`, `learn`, and `compromise` events are straightforward. However, the interpretation of `accept` and `send` events is deliberately left up to the protocol specifier. This is because what the protocol specifier is trying to determine is whether or not what he or she thinks of as adequate `send` and `accept` events are actually the ones that are necessary to make the protocol function properly. Thus, for example, the `send` event

$$\texttt{send}(\mathit{server}(\texttt{S}),(\mathit{principal}(\texttt{A},[\mathit{honest}]),\mathit{principal}(\texttt{B},[\texttt{X}])),[\texttt{KEY}],\texttt{N?})$$

could describe the server sending a key to A who will then forward it to B, the server sending a key to A and B simultaneously, or even the server sending the key to some fourth party who will then forward it to A and B. The point is that, whatever interpretation is used, in each case the server has set into motion a chain of events by which A and B are supposed to receive the key.

Of course, since the Analyzer reasons about the unreachability of insecure states, it does not use the requirements language directly. Instead, it is necessary to define insecure states in terms of negations of the requirements. The Analyzer can then be used to reason about the unreachability of these insecure states. Details of how this is done are given in [8]. A briefer, earlier description of the language and its application is given in [6], and an analysis of protocols for repeated authentication using NPATRL and NPA is given in [7]. NPATRL is a fairly flexible language. This is reflected by the fact that all of the constructs set out below are application details specified in NPATRL. We have, however, found it necessary to make one addition to NPATRL itself. Previous applications have not required the use of quantifiers, but they appear to be unavoidable in this context. Nonetheless, the requirements that make use of them seem to be normally amenable to NPA analysis, and we will not here discuss any details of adding quantifiers to the language.

4 SET Requirements Specification

Having described the basic ideas behind payment transactions in SET and the language NPATRL, we now present a formal specification of requirements for SET in NPATRL. The chief problem faced in developing formal requirements for SET is the complex nature of the construct that is being verified by the three parties. We thus begin with a presentation and discussion of the constructs that we develop to allow the representation of SET requirements in NPATRL, before proceeding to the requirements themselves.

4.1 Constructs Used

Most work on developing formal requirements for cryptographic protocols has concentrated on key distribution and agreement protocols, in which the construct being agreed upon, the key, is an atomic object that is visible to all parties. The transaction agreed upon in the SET protocol is much more elaborate. It contains a number of components that are added as the protocol progresses. Thus, the transaction agreed upon by merchant and customer in the first part of the protocol will not be the same as the one agreed upon by merchant, customer, and payment gateway at the end. Things are made even more difficult by the fact that some components of a transaction may be hidden from one of the parties. For example, both the customer and payment gateway have access to the customer's credit cared number (PAN), but this may be hidden from the merchant. Thus we need to be able to specify an evolving construct, some parts of which may be hidden from the parties involved.

Our solution is to model a transaction as a vector. We define a set of projection functions that give each party's view of a transaction at each point in the protocol. Note that this use of projection functions defined in terms of both the relevant party and that party's place in the protocol allow us to model both that party's ignorance of terms that have not yet been generated and its ignorance of terms that it is not supposed to know.

We define our projection functions and related notions as follows.

Definition: Let $\mathcal{V}$ be the set of n-vectors over $C \cup \{\perp\}$, where C is some alphabet, and let $\mathcal{W}$ be the set of n-vectors over C. If $V \in \mathcal{V}$, we define the *support* of V to be the set of all i between 1 and n such that $V[i]$ is not $\perp$. We say that V *agrees with* V' (written $V \sim V'$) if they agree on their common support, i.e., if for all $i \in \mathit{support}(V) \cap \mathit{support}(V')$, $V[i] = V'[i]$. A *projection*, proj, is a function mapping $\mathcal{W}$ to $\mathcal{V}$ such that there exists a set X such that $i \in X$ implies that $\mathrm{proj}(V)[i] = V[i]$ and $i \notin X$ implies that $\mathrm{proj}(V)[i] = \perp$. In such a case we say $X = \mathit{support}(\mathrm{proj})$.

We note that the agrees relation is not necessarily transitive. However, the following lemma does hold. We leave its proof as an exercise to the reader:

Lemma 1: Suppose that $\mathit{support}(\mathrm{proj}_1) \subseteq \mathit{support}(\mathrm{proj}_2)$, and that $\mathcal{W}$ is as in the above definition. Then, for all V, V', and V'' in $\mathcal{W}$, if $\mathrm{proj}_3(V'') \sim \mathrm{proj}_2(V')$, and $\mathrm{proj}_2(V') \sim \mathrm{proj}_1(V)$, then $\mathrm{proj}_3(V'') \sim \mathrm{proj}_1(V)$.

The following lemma also helps us to simplify our requirements in some cases:

Lemma 2: Suppose that $support(\text{proj}_1) \subseteq support(\text{proj}_2)$, and that $\mathcal{W}$ is as in the above definition. Then, for all V, V', and V'' in $\mathcal{W}$, $\text{proj}_1(V') \sim \text{proj}_2(V)$ if and only if $\text{proj}_1(V) = \text{proj}_1(V')$.

Lemma 2 will allow us to replace $\text{proj}_1(V')$ with $\text{proj}_1(V)$ in any requirement involving $\text{proj}_1(V')$ where $\text{proj}_1(V') \sim \text{proj}_2(V)$ and $support(\text{proj}_1) \subseteq support(\text{proj}_2)$.

We define the transaction vector as follows. Let V be a transaction vector. Its components are defined by:

- $V[1a]$ = Portion of the Transaction ID generated by customer
- $V[1b]$ = Portion of the Transaction ID generated by merchant
- $V[2]$ = customer
- $V[3]$ = hash of order data
- $V[4]$ = customer's PAN
- $V[5]$ = customer's PANSecret
- $V[6]$ = purchase amount
- $V[7]$ = merchant
- $V[8]$ = authorized purchase amount
- $V[9]$ = capture amount
- $V[10]$ = authorized capture amount
- $V[11]$ = *signed* or *unsigned*

Some comments:

1. We will assume that $V[1b]$ is zero if no initialization message is sent, and that $V[5]$ is zero if no PANSecret in used.
2. The hash in $V[3]$ is taken over the purchase amount and the order description that was agreed to by the customer and merchant offline. The order description may or may not be sent to the gateway, but the hash always is sent, and can be used by the gateway to verify that the customer and merchant agreed on the same order description.
3. The value *signed* or *unsigned* does not actually appear in the SET protocol. It refers to whether or not the customer's purchase request was signed with a digital signature or not. Since the type of security that may be guaranteed will be different in each case, it is important that this be accounted for.

The projections for honest principals are constructed as follows. We first give projections describing the knowledge that each principal has. This information will be used in 'accept' actions.

1. $cust\_req(V) = [1a, 1b, 2, 3, 4, 5, 6, 7, \perp, \perp, \perp, 11]$
   (information known by customer when it sends purchase request)
2. $merch\_req(V) = [1a, 1b, 2, 3, \perp, \perp, 6, 7, \perp, 9, \perp, 11]$
   (information known by merchant when it sends request to bank)
3. $apg\_resp(V) = [1a, 1b, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$
   (information known by bank when it accepts a merchant's authorization request)

4. $merch\_resp(V) = [1a, 1b, 2, 3, \perp, \perp, 6, 7, 8, 9, 10, 11]$
   (information known by merchant when it accepts a customer's purchase request)
   (in some cases $merch\_resp(V) = [1a, 1b, 2, 3, 4, \perp, 6, 7, 8, 9, 10, 11]$)
5. $cust\_accept(V) = [1a, 1b, 2, 3, 4, 5, 6, 7, 8, \perp, 10, 11]$
   (information known by customer when it accepts a transaction)

We next give projections describing the information that each honest principal sends, to be used in 'send' actions. Note that a single party's send projections are not monotonically increasing, as in the case of the accept actions, since once a data item is sent it may not be sent again. Note also that we do not define a projection for the merchant's response to the initialization request. That is because the merchant's response does not appear in our requirements.

1. $cust\_pinitsend(V) = [1a, \perp, 2, \perp, \perp, \perp, \perp, 7, \perp, \perp, \perp, \perp]$
   (information sent by customer in initialization request to merchant)
2. $cust\_reqsend(V) = [1a, 1b, 2, 3, 4, 5, 6, 7, \perp, \perp, \perp, 11]$
   (information sent by customer in purchase request to merchant)
3. $merch\_reqsend(V) = [1a, 1b, 2, 3, \perp, \perp, 6, 7, \perp, 9, \perp, 11]$
   (information sent by merchant in authorization request to bank)
4. $apg\_respsend(V) = [1a, 1b, 2, \perp, \perp, \perp, \perp, 7, 8, 9, 10, \perp]$
   (information sent by bank to merchant in response to authorization request)
   (in some cases $apg\_respsend(V) = [1a, 1b, 2, \perp, 4, \perp, \perp, 7, 8, 9, 10, \perp]$)
5. $merch\_respsend(V) = [1a, 1b, 2, \perp, \perp, \perp, \perp, 7, 8, \perp, 10, \perp]$
   (information sent by merchant to customer in response to purchase order)

A table summarizing the content of all these projections is given in Figure 1.

We now consider how to map transaction vectors to the NRL Protocol Analyzer model. In the case of honest participants, components of accept transaction vectors can be represented by values of local state variables. Thus, when a customer requests a purchase amount, that will be stored in a local state variable representing $cust\_req(\mathrm{V})[7]$, and so forth. Components of send transaction vectors can be represented by components of messages.

In the case of dishonest participants, projections can no longer represent beliefs, since beliefs of dishonest participants are not represented in the NPA model. However, dishonest participants do send messages, and we can determine what the appropriate values of the send projections should be from messages received by honest participants. For example, let X be the result of encrypting a customer P's PAN, PAN_P, with the gateway's public key. If an honest merchant receives the message X attributed to dishonest customer P, we can conclude that the value of $cust\_reqsend(\mathrm{V})[4] = PAN_P$, whether or not PAN_P is P's actual PAN.

4.2 SET Payment Protocol Requirements

We are now ready to begin with the actual requirements. In each case, we give an informal account, followed by the NPATRL specification.

Replay Requirements We have a general requirement that no projection of a transaction (for the same role) should be accepted twice. We must be careful, though, to exclude from this requirement any fields that the principal generated itself when accepting that transaction. Thus, we phrase our requirement as follows: If an honest principal accepts a projection $\mathrm{proj}_1(V)$, such that $\mathrm{proj}_2(V') \sim \mathrm{proj}_1(V)$ for some other projection proj_2, where proj_1 and proj_2 are from the set of projections defined in section 4.1, then no other principal in the same role previously accepted any $\mathrm{proj}_1(V'')$ such that $\mathrm{proj}_1(V'') \sim \mathrm{proj}_2(V')$.

$$\begin{array}{l}(\mathtt{accept}(\mathit{role}(\mathtt{P},[\mathit{honest}]),-,\mathrm{proj}_1(\mathtt{V}),\mathtt{N}?) \ \wedge \ (\mathrm{proj}_1(\mathtt{V}) \sim \mathrm{proj}_2(\mathtt{V}'))) \\ \quad \rightarrow ((\mathrm{proj}_1(V'') \sim \mathrm{proj}_2(V')) \\ \qquad \rightarrow \neg\Diamond\mathtt{accept}(\mathit{role}(\mathtt{Q},[\mathit{honest}]),-,\mathrm{proj}_1(\mathtt{V}''),\mathtt{N}?)))\end{array}$$

This is a fairly simple requirement. But, as in the rest of SET, nothing is ever quite this simple. This is so in the case of $\mathit{merch\_req}(V)$, if the optional customer initialization message is used. In that case, the merchant sends a random challenge in response. The presence of this challenge, together with the customer's digital signature, protects the merchant against replay of the purchase request message. If these features are not present, the merchant can be protected against replay by the gateway's checking against the forwarded customer request against its database, but the merchant itself apparently does not check for the freshness of its data directly. Our requirement thus becomes: if the merchant accepts $\mathit{merch\_resp}(V)$, and previously the customer sent $\mathit{cust\_pinitsend}(V')$, where $\mathit{merch\_resp}(V) \sim \mathit{cust\_pinitsend}(V')$ and $\mathit{merch\_resp}(V)[11] = \mathit{signed}$, then the merchant did not previously accept $\mathit{merch\_resp}(V'')$, where $\mathit{merch\_resp}(V'') = \mathit{merchresp}(V)$.

$$\begin{array}{l}(\mathtt{accept}(\mathit{merchant}(\mathtt{Q},[\mathit{honest}]),\mathit{customer}(\mathtt{P},[\mathtt{X}]),\mathit{merch\_resp}(\mathtt{V}),\mathtt{N}?) \wedge \\ \Diamond\mathtt{send}(\mathit{customer}(\mathtt{P},[\mathtt{X}]),\mathit{merchant}(\mathtt{Q},[\mathit{honest}]),\mathit{cust\_pinitsend}(\mathtt{V}'),\mathtt{N}?) \wedge \\ \mathit{merch\_resp}(V) \sim \mathit{cust\_pinitsend}(V') \ \wedge \ \mathit{merch\_resp}(V)[11] = \mathit{signed} \ ) \\ \quad \rightarrow (\mathit{merch\_resp}(V'') = \mathit{merch\_resp}(V) \\ \qquad \rightarrow \neg\Diamond\mathtt{accept}(\mathit{merchant}(\mathtt{Q},[\mathit{honest}]),\mathit{customer}(\mathtt{P},[\mathtt{X}]),\mathit{merch\_resp}(\mathtt{V}''),\mathtt{N}?))\end{array}$$

Faithful Protocol Execution This requirement states that honest principals will faithfully execute the protocol. Thus, if a send event occurs, then the corresponding accept event occurred previously. In other words, if an honest principal P, playing a given role, engages in the send event $\mathit{role\_eventsend}(V)$, then it should have previously engaged in a corresponding accept event $\mathit{role\_event}(V')$, where $\mathit{role\_event}(V') \sim \mathit{role\_eventsend}(V)$.

$$\begin{array}{l}\mathtt{send}(\mathit{role}(\mathtt{P},[\mathit{honest}]),-,\mathit{role\_eventsend}(\mathtt{V}),\mathtt{N}) \\ \quad \rightarrow \exists V'((\mathit{role\_eventsend}(V) \sim \mathit{role\_event}(V')) \wedge \\ \qquad \Diamond\mathtt{accept}(\mathit{role}(\mathtt{P},[\mathit{honest}]),-,\mathit{role\_event}(\mathtt{V}'),\mathtt{N}))\end{array}$$

Customer Requirements The customer's main requirement is that he be given a guarantee that he will receive the goods in return for his money, or if he has already received the goods, that he be given notification that the merchant agrees

that the bank has agreed to pay for the goods, that is, that the customer has obtained the goods legally. In other words, if the customer accepts $cust\_accept(V)$, then there should exist a V' such that $merch\_respsend(V') \sim cust\_accept(V)$ and the merchant sent $merch\_respsend(V')$. However, Lemma 2 and the fact that $merch\_respsend(V') \sim cust\_accept(V)$ allow to simplify this to the requirement that, if the customer accepts $cust\_accept(V)$, then the merchant should have sent $merch\_respsend(V)$. This requirement may be stated as follows:

$$\texttt{accept}(customer(\texttt{P}, [honest]), merchant(\texttt{Q}, [\texttt{X}]), cust\_accept(\texttt{V}), \texttt{N}?)$$
$$\rightarrow \Diamond\texttt{send}(merchant(\texttt{Q}, [\texttt{X}]), customer(\texttt{P}, [honest]), merch\_respsend(\texttt{V}), \texttt{N}?)$$

We also have a requirement that the customer and an honest merchant should have the same view of what is going on. The merchant's view of the transaction is not captured by $merch\_respsend(V)$, but rather by $merch\_resp(V')$, for some V'. Thus, we must add the new requirement that, if the customer accepts $cust\_accept(V)$, then there exists a V' such that $merch\_resp(V') \sim cust\_accept(V)$ and the merchant accepted $merch\_resp(V')$. This requirement may be stated as follows.

$$\texttt{accept}(customer(\texttt{P}, [honest]), merchant(\texttt{Q}, [honest]), cust\_accept(\texttt{V}), \texttt{N}?)$$
$$\rightarrow \exists V'((cust\_accept(V) \sim merch\_resp(V')) \wedge$$
$$\Diamond\texttt{accept}(merchant(\texttt{Q}, [honest]), customer(\texttt{P}, [honest]), merch\_resp(\texttt{V}'), \texttt{N}?)))$$

We now move to requirements concerning the gateway. We note that, in the case in which the merchant is dishonest, the customer has no way of knowing that the merchant even communicated with the gateway, so we can make no requirements in that case. However, in the case of an honest merchant, it is reasonable to require that the customer and the gateway share the same view of the transaction, and that the gateway actually communicated with the merchant.

We first consider the requirement that, if the customer accepts $cust\_accept(V)$, then there is a V' such that $apg\_respsend(V') \sim cust\_accept(V)$ and the gateway sent $apg\_respsend(V')$. As it turns out, we do not need to state this requirement explicitly; it results from Lemma 1 and the following:

1. Our previously stated requirement that there exists a V'' such that $merch\_resp(V'') \sim cust\_accept(V)$ and the merchant accepts $merch\_resp(V'')$;
2. A requirement to be given in the Merchant Requirements section saying that, if the merchant accepts $merch\_resp(V'')$, then there is a V' such that $merch\_resp(V'') \sim apg\_respsend(V')$, and the gateway sent $apg\_respsend(V')$, and;
3. The fact that $support(apg\_respsend) \subseteq support(merch\_resp)$.

We do not have such a convenient subset relation for the requirement that the customer and the gateway share the same view of the transaction. Thus, it is necessary to introduce an explicit requirement here: if the customer accepts $cust\_accept(V)$ from an honest merchant, then there is a V' such that $cust\_accept(V) \sim apg\_resp(V')$ and the gateway accepted $apg\_resp(V')$. This may be stated formally as follows.

$$\begin{aligned}&\mathtt{accept}(\mathit{customer}(\mathtt{P},[\mathit{honest}]),\mathit{merchant}(\mathtt{Q},[\mathit{honest}]),\mathit{cust\_accept}(\mathtt{V}),\mathtt{N}?)\\&\quad\rightarrow \exists V'((\mathit{cust\_accept}(V)\sim \mathit{apg\_resp}(V')\wedge\\&\qquad \Diamond\mathtt{accept}(\mathit{gateway}(\mathtt{R},[\mathit{honest}]),\mathit{customer}(\mathtt{P},[\mathit{honest}]),\mathit{apg\_resp}(\mathtt{V}'),\mathtt{N}?))\end{aligned}$$

Finally, there is a less obvious requirement that, if the customer receives notification that he will receive or has paid for goods, then these should be goods he already ordered. In other words, if the customer accepts $\mathit{cust\_accept}(V)$, then the customer should have sent some $\mathit{cust\_reqsend}(V')$, where $\mathit{cust\_reqsend}(V')\sim \mathit{cust\_accept}(V)$. By Lemma 2, this can be simplified to the requirement that, if the customer accepts $\mathit{cust\_accept}(V)$, then the customer sent $\mathit{cust\_reqsend}(V)$. This is stated as follows.

$$\begin{aligned}&\mathtt{accept}(\mathit{customer}(\mathtt{P},[\mathit{honest}]),\mathit{merchant}(\mathtt{Q},[\mathtt{X}]),\mathit{cust\_accept}(\mathtt{V}),\mathtt{N})\\&\quad\rightarrow \Diamond\mathtt{send}(\mathit{customer}(\mathtt{P},[\mathit{honest}]),\mathit{merchant}(\mathtt{Q},[\mathtt{X}]),\mathit{cust\_reqsend}(\mathtt{V}),\mathtt{N})\end{aligned}$$

Merchant Requirements The merchant's main requirement is that, if the merchant agrees to deliver the goods to the customer, then it should have received a guarantee from the gateway that it will receive its money. In other words, if the merchant accepts $\mathit{merch\_resp}(V)$, then the gateway should have sent $\mathit{apg\_respsend}(V')$ for some V' such that $\mathit{apg\_respsend}(V')\sim \mathit{merch\_resp}(V)$. Since $\mathit{support}(\mathit{apg\_respsend})\subseteq \mathit{support}(\mathit{merch\_resp})$, Lemma 2 allows us to replace this with the requirement that the gateway send $\mathit{apg\_respsend}(V)$. Likewise, if the merchant passes on a response to a customer, than it should have previously requested that response from the gateway. This will prevent the merchant from agreeing to supply goods that it never offered for sale. In other words, if the merchant accepts $\mathit{merch\_resp}(V)$, then it should have previously sent $\mathit{merch\_reqsend}(V)$. Finally, we require that the merchant and the gateway have the same picture of the transaction. This is captured by requiring that, if the merchant accepts $\mathit{merch\_resp}(V)$, then the gateway should have accepted $\mathit{apg\_respsend}(V')$ for some V' such that $\mathit{apg\_resp}(V')\sim \mathit{merch\_resp}(V)$. These three requirements may be stated as follows.

$$\begin{aligned}&\mathtt{accept}(\mathit{merchant}(\mathtt{Q},[\mathit{honest}]),\mathit{customer}(\mathtt{P},[X]),\mathit{merch\_resp}(\mathtt{V}),\mathtt{N}?)\\&\quad\rightarrow \Diamond\mathtt{send}(\mathit{gateway}(\mathtt{R},[\mathit{honest}]),\\&\qquad(\mathit{customer}(P,[X]),\mathit{merchant}(Q,[\mathit{honest}])),\mathit{apg\_respsend}(V),N?)\end{aligned}$$

$$\begin{aligned}&\mathtt{accept}(\mathit{merchant}(\mathtt{Q},[\mathit{honest}]),\mathit{customer}(\mathtt{P},[\mathtt{X}]),\mathit{merch\_resp}(\mathtt{V}),\mathtt{N})\\&\quad\rightarrow \Diamond\mathtt{send}(\mathit{merchant}(\mathtt{Q},[\mathit{honest}]),\mathit{gateway}(\mathtt{R},[\mathit{honest}]),\mathit{merch\_req}(\mathtt{V}),\mathtt{N})\end{aligned}$$

$$\begin{aligned}&\mathtt{accept}(\mathit{merchant}(\mathtt{Q},[\mathit{honest}]),\mathit{customer}(\mathtt{P},[X]),\mathit{merch\_resp}(\mathtt{V}),\mathtt{N}?)\\&\quad\rightarrow \exists V'((\mathit{apg\_resp}(V')\sim \mathit{merch\_resp}(V))\wedge\\&\qquad\Diamond\mathtt{accept}(\mathit{gateway}(\mathtt{R},[\mathit{honest}]),\\&\qquad\quad(\mathit{customer}(P,[X]),\mathit{merchant}(Q,[\mathit{honest}])),\mathit{apg\_resp}(V'),N?))\end{aligned}$$

We now turn to requirements concerning the merchant's interaction with the customer. We first want to show that the merchant accepts a transaction only if the customer has requested it, that is, the merchant accepts $\mathit{merch\_resp}(V)$ only

if there exists a V' with $merch\_resp(V) \sim cust\_reqsend(V')$ such that the customer sent $cust\_reqsend(V')$. We leave it as an exercise to the reader to show that this requirement is implied by Lemma 1 and the fact that $support(cust\_reqsend) \subseteq support(apg\_resp)$, together with the requirements expressed in this section and the Gateway Requirements section, and thus does not need to be stated explicitly.

We may also want to require that, when the merchant is aware that digital signatures are used by the customer ($merch\_req(V)[11] = signed$), and the merchant accepts $merch\_req(V)$, then the customer must have sent $cust\_reqsend(V')$, where $merch\_req(V) \sim cust\_reqsend(V')$. This may be stated as follows.

$$\begin{aligned}
&(\mathtt{accept}(merchant(\mathtt{Q}, [honest]), customer(\mathtt{P}, [\mathtt{X}]), merch_req(\mathtt{V}), \mathtt{N}) \\
&\quad \wedge (merch_req(V)[11] = signed)) \\
&\rightarrow \exists V'((cust_req(V') \sim merch_req(V)) \wedge \\
&\quad \Diamond \mathtt{send}(customer(\mathtt{P}, [honest]), merchant(\mathtt{Q}, [honest]), cust_req(\mathtt{V'}), \mathtt{N}))
\end{aligned}$$

There is also an implicit requirement that the merchant be able to prove that the customer initiated the transaction in the event of a later dispute. This is indeed the reason for allowing the gateway to send the PAN to the merchant. However, since the protocol for verifying the merchant's claim is not explicitly defined in SET 1.0, we do not include this as a formal requirement.

Gateway Requirements The gateway's job is to mediate between the customer and the merchant. To do this, it must be able to determine that both the customer and the merchant have actually sent their requests. Thus, the gateway will not accept $apg\_resp(V)$ unless the merchant and the customer have already sent $merch\_reqsend(V')$ and $cust\_reqsend(V'')$, respectively where $merch\_reqsend(V') \sim apg\_resp(V)$ and $cust\_reqsend(V'') \sim apg\_resp(V)$. Again, Lemma 2 and the fact that the supports of both merchant and customer projections are subsets of $support(apg\_resp)$ allows us to replace $merch\_reqsend(V')$ with $merch\_reqsend(V)$ and $cust\_reqsend(V'')$ with $cust\_reqsend(V)$. We do not make any requirement on the order in which the merchant and customer requests are sent. This raises the possibility of a 'psychic merchant' who anticipates a customer's request and sends his request to the gateway before receiving it from a customer. Although this is nonsensical, we don't consider it a security violation, so we don't attempt to guard against it.

The formal version of this requirement is as follows:

$$\begin{aligned}
&\mathtt{accept}(gateway(\mathtt{R}, [honest]), \\
&\qquad (merchant(Q, [honest]), customer(P, [honest])), apg_resp(V), N?) \\
&\rightarrow \Diamond(\mathtt{send}(customer(\mathtt{P}, [\mathtt{X}]), \\
&\qquad (customer(P, [X]), merchant(Q, [Y])), cust_req(V), N?) \wedge \\
&\quad \mathtt{send}(merchant(\mathtt{Q}, [\mathtt{X}]), \\
&\qquad (customer(P, [X]), merchant(Q, [Y])), merch_req(V), N?))
\end{aligned}$$

In the case that the customer is honest, we also make the requirement that her view of the transaction agrees with the view of the gateway, that is, that the

customer previously accepted $cust\_req(V')$, where $cust\_req(V') \sim apg\_resp(V)$. In this case, however, the fact that $support(cust\_req) \subseteq support(apg\_resp)$, together with the faithfulness requirements and the requirements given just above on the gateway's accepting requests, allows us to use Lemma 1 to derive this requirement from the others. We again leave the proof of this as an exercise to the reader.

Likewise, when the merchant is honest, we make the requirement that if the gateway accepts $apg\_resp(V)$, then the merchant previously accepted $merch\_req(V')$, where $merch\_req(V') \sim apg\_resp(V)$. The proof that this requirement is derivable from the others is also left to the reader.

Requirements for customer, merchant, and gateway are represented diagrammatically in Figure 2.

Secrecy Requirements The SET protocol makes use of the PAN and the PANSecret to provide authentication. Since the PAN is also used for authentication outside of the SET protocol, it is the responsibility of the protocol to protect the PAN. As we have seen, if a misjudgment is made and the PAN is delivered to a dishonest merchant, then it can be compromised. Thus we need to guarantee that, if the intruder learns the PAN (from runs of the protocol), then this was done as a result of the gateway's sending the PAN to a dishonest merchant. Similarly, the protocol should not reveal the PANSecret of an honest customer under any circumstances. These requirements are specified formally as follows.

$$\begin{aligned}&\texttt{learn}(intruder, -, pan(customer(\texttt{P}, [honest]), \texttt{N}?)\\ &\quad \rightarrow \exists V(V[4] = pan(customer(P, [honest]) \wedge\\ &\qquad \Diamond\texttt{send}(gateway(\texttt{R}, [honest]), merchant(\texttt{Q}, [dishonest]), apg\_respsend(\texttt{V}), \texttt{N}?))\end{aligned}$$

$$\neg(\texttt{learn}(intruder, -, pansecret(customer(\texttt{P}, [honest]), \texttt{N}?))$$

The SET protocol is also intended to protect the secrecy of the monetary values passes in the protocol: that is, the purchase amount, the amount authorized by the gateway, and the amount captured by the gateway. The secrecy protection is very weak: ratios of authorization amount to purchase amount and of capture amount to purchase amount are sent in the clear. But we still need to guarantee that these data are not revealed otherwise than by a direct attack on this weak method of encryption. This we can do by modeling the ratio function in the Protocol Analyzer specification, requiring that one element of the ratio is required in order to reveal the other, and then requiring that the intruder could not have learned these data unless the customer were involved in an interaction with a dishonest merchant. This will be done in two parts:

1. If the intruder learns the purchase amount of a transaction involving an honest customer, then that customer must have initiated the transaction with a dishonest merchant.
2. If the intruder learns the authorization or capture amounts of a transaction involving an honest customer, then that customer must have initiated a

transaction $cust\_reqsend(V)$, with a dishonest merchant, Q, and then the gateway must have sent $apg\_respsend(V')$ to Q, where $cust\_reqsend(V) \sim apg\_respsend(V')$.

The formal versions of these requirements are given below. We are not actually concerned with the intruder learning purchase, authorization, or capture dollar amounts; we are concerned with her learning the association of these with a given transaction. However, there are difficulties in the representation and analysis of that association itself. Therefore, as a quick solution, we assume that a purchase/authorization/capture amount chosen by a principal at any time T is unique. While perhaps not reflective of reality, the only practical limitation this has for us is the inability to represent attacks depending on the identity of two such amounts in different transactions. We may seek a more elegant solution in the future. The requirement for the intruder learning the capture amount is virtually identical to the requirement for learning the authorization amount, and so we omit it.

$$\begin{aligned}
&\texttt{learn}(intruder, -, purchamt(customer(\texttt{P}, [honest]), \texttt{T}), \texttt{N}?) \\
&\quad \rightarrow \exists V(\Diamond\, \texttt{send}(customer(\texttt{P}, [honest]), \\
&\qquad merchant(Q, [dishonest]), cust_reqsend(V), N?) \wedge \\
&\qquad V[6] = purchamt(customer(P, [honest]), T))
\end{aligned}$$

$$\begin{aligned}
&\texttt{learn}(intruder, -, authamt(customer(\texttt{P}, [honest]), gateway(\texttt{R}, [honest]), \texttt{T}), \texttt{N}?) \\
&\quad \rightarrow \exists V(\Diamond\, \texttt{send}(customer(\texttt{P}, [honest]), \\
&\qquad merchant(Q, [dishonest]), cust_reqsend(V), N?) \wedge \\
&\qquad \exists V'(cust_reqsend(V) \sim apg_respsend(V') \wedge \\
&\qquad \Diamond\, \texttt{send}(gateway(\texttt{R}, [honest]), \\
&\qquad\quad merchant(Q, [dishonest]), apg_respsend(V'), N?) \wedge \\
&\qquad V'[8] = authamt(customer(P, [honest]), gateway(R, [honest]), T)))
\end{aligned}$$

5 Comparison With Other Work

As we mentioned at the beginning of this paper, most work on developing requirements for cryptographic protocols has concentrated on secure agreement on atomic object such as keys. However, there has been some work closely related to ours on developing requirements for agreement on more complex transactions. In [2] Bolignano describes the following approach to specifying requirements for complex protocols such as SET. According to Bolignano's definition, a requirement is divided into two parts: a regular language L, and a filtering function ff_x on sequences of messages so that the requirement is satisfied on sequence M if and only if $ff_x(M)$ is in L. Bolignano shows how filtering functions can be used to express requirements on sequences of messages in a protocol in terms of conditions on individual components in the messages. Thus, like us, he can require that different components of different messages in a sequence must agree in order for a protocol to execute correctly. Although we have not attempted to verify this, we believe that Bolignano's approach could be used to specify

the requirements we have set out in this paper. Indeed, in [2] he reports that he is applying this technique to the analysis of the SET protocol. We believe that the advantage of our approach lies in the fact that the use of projections and the agreement relation allows us to simplify greatly the expression of the requirements, and thus makes them easier to work with. The complex part of requirements specification for protocols such as SET appears to lie mostly in the definition of the projections. Once this was done, the remaining portion of the requirements turned out to be not that much more complex than requirements for protocols for secure agreement on atomic objects such as keys. It might be interesting to see if a similar approach could be used to simplify the requirements in [2].

Another approach to formal analysis of complex payment protocols is given by Brackin [3]. He describes the analysis of two large protocols for electronic commerce developed by CyberCash. These protocols are similar to SET in their primary respects. Brackin uses an automated theorem prover based on HOL. He specifies the protocol in an extension of GNY [4], itself an extension of BAN [1]. Thus, protocol goals are specified in terms of the beliefs of the principals, e.g., that the gateway believes that the merchant has sent a *merch_req*. The analysis appears to be at a higher level of abstraction than either the present work or that of Bolignano. It thus potentially assumes away some significant features and possibly even vulnerabilities. It is also in some ways not as abstract as the present work. For example, agreement on a transaction is not represented at all; rather, agreement on individual fields within the transaction is analyzed. However, Brackin's analysis is able to highlight, e.g., some of the trust assumptions in the protocols and to provide assurance against some common high level protocol vulnerabilities.

6 Conclusions

We have presented a formal specification of requirements for the payment portion of the SET protocol in the language NPATRL. By introducing transaction vectors, projections thereon, and the vector agreement relation we have been able to present requirements that are completely formal and capture much detail yet are quite readable representations of intuitive goals. Understanding the goals of SET, even informally, has previously been difficult. Since we have been able to represent the SET requirements in NPATRL, they are now amenable to analysis using the NRL Protocol Analyzer to evaluate. This is the focus of future work.

Acknowledgements

David Goldschlag took part in most of the early meetings when this work was taking shape. We thank him for much helpful input given at that time. This work was supported by DARPA.

References

1. M. Burrows, M. Abadi, and R. Needham, *A Logic of Authentication*, SRC Research Report 39, Digital Systems Research Center, February 1989.
2. D. Bolignano, "Towards the Formal Verification of Electronic Commerce Protocols", *Proceedings of the* 10^{th} *IEEE Computer Security Foundations Workshop*, pp. 133–146, Rockport Massachusetts, IEEE CS Press, June 1997.
3. S. Brackin, "Automatic Formal Analyses of Two Large Commercial Protocols", *DIMACS Workshop on Design and Formal Verification of Security Protocols*, Rutgers New Jersey, September 1997. (Paper available at http://dimacs.rutgers.edu/Workshops/Security/program2/brackin.html)
4. L. Gong, R. Needham, and R. Yahalom, "Reasoning about Belief in Cryptographic Protocols", *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 234–248, IEEE Computer Society Press, Oakland California, May 1990.
5. SET Secure Electronic Transaction Specification, Version 1.0, May 1997. (Downloaded from http://www.visa.com/set/)
6. P. Syverson and C. Meadows, "A Logical Language for Specifying Cryptographic Protocol Requirements", *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 165–177, IEEE Computer Society Press, Oakland California, May 1993.
7. P. Syverson and C. Meadows, "Formal Requirements for Key Distribution Protocols", *Advances in Cryptology — EUROCRYPT '94*, LNCS vol. 950, A. De Santis, ed., pp. 320–331, Springer-Verlag, Perugia Italy, 1994.
8. P. Syverson and C.Meadows, "A Formal Language for Cryptographic Protocol Requirements", *Designs, Codes, and Cryptography*, vol. 7, nos. 1 and 2, pp. 27–59, January 1996.