Towards the Formal Verification of Electronic Commerce Protocols*

Dominique Bolignano, GIE Dyade, Dominique.Bolignano@dyade.fr

Abstract

We generalize the approach defined in [4] so as to be able to formally verify electronic payment protocols. The original approach is based on the use of general purpose formal methods. It is complementary with modal logic based-approaches as it allows for a description of protocols, hypotheses and authentication properties at a finer level of precision and with more freedom. The proposed generalization mainly requires being able to express and verify payment properties. Such properties are indeed much more elaborate than authentication ones, and require a significant generalization in the way properties are expressed. The modelling of the protocol and of the potential knowledge hold by intruders is on the other hand left unchanged. The approach is currently being applied to the C-SET and SET protocols, and has already lead to significant results.

1 Introduction

Consumer demand for secure access to electronic shopping and other services is becoming very high. Many electronic commerce protocols ([12, 6, 2, 9, 14], etc.) have been proposed recently to meet this demand. Such protocols mainly use encryption and decryption functions to achieve security requirements.

But the design of a cryptographic protocol is a difficult and error-prone task, and many popular and largely used cryptographic protocols have been shown to have flaws. For this reason the use of formal methods that allow for the verification of such protocols in a systematic and formal way has received increasing attention. Formal methods have been mainly applied to authentication protocols. Their industrial application has been quite marginal until now, but they have shown successful in finding problems or flaws in protocols (e.g. [5, 11, 4], etc.)

Electronic payment appears to be a very promising area of application of such systematic approaches. Formal approaches for the verification of cryptographic protocols are indeed quite mature at a time where many new protocols are designed and implemented. These new protocols put at the same time a higher requirement on security issues: while it is possible to obtain account information in other environments, there is a heightened concern about the ease of doing so with public network transactions; this concern reflects the potential for high volume fraud, automated fraud (such as using filters on all messages passing over a network to extract all payment card account numbers out of a data stream), and the potential for mischievous fraud that appears to be characteristic of some hackers.

Formal techniques for the verification of cryptographic protocol have been developed for the verification of authentication protocols but they cannot directly be applied to electronic payment protocols, mainly because of the difference between authentication and payment properties.

*Draft: to be presented at the "10th IEEE Computer Security Foundations Workshop"

Payment properties involve some standard authentication requirements. As an example of such standard requirement, let us consider the situation found in many payment protocols such as SET [12], C-SET [6], Globe-On-line [14], iKP [2], etc., where the merchant will typically ask to a representative of a bank, the payment gateway, for a payment authorization. The merchant, at the time when he receives payment authorization, wants typically to be sure that he has really been talking to the proper gateway, and that the payment authorization has just been issued by the gateway and is not a replay of an old message for example. This property is a typical requirement for traditional authentication protocols. But payment properties also involve more specific requirements that are not found in authentication protocols. In the previous example the merchant wants also to be sure that the payment authorization he receives really is for the transaction he requested for: not for another product, for another price, or for another customer, as in case of "misunderstanding" the customer could well try to cancel payment.

2 Formal Verification of Cryptographic Protocols

Techniques for the formal verification of cryptographic protocols have been designed and applied mainly for the verification of authentication protocols.

The formal verification of authentication protocols may be done in basically three ways. One possibility is to use a modal logic for authentication, that is, a logic that has some modalities for expressing properties in an appropriate way. One such modal logic is the Burrows, Abadi and Needham (BAN) logic [5]. The second possibility is to use general purpose formal methods. Such methods have been designed and used for various kinds of software applications: safety-critical embedded systems for trains, nuclear power plants; complex distributed systems; etc. The approach of Meadows [13] is a good representative of such approaches. The third possibility is to use model-checking techniques as in [11, 10].

More recently another approach has been presented in [4]. This is the approach we rely on in this paper. It uses general purpose formal methods but tries to exploit the specificity of the problem to achieve easier formalization and to reduce proofs. It has been mainly designed to achieve simplicity and conciseness in proofs when a precise modeling of the protocol is required. The idea is to have proofs that can be read and understood, and for example used as a basis for code inspection procedures or vulnerability analysis. The approach is based on a clear separation between the modeling of trustable principals and that of untrustable ones. A specific axiomatization for the knowledge of untrustable principals is proposed. This allows for easy and concise proofs in the style of modal logic proofs. The specification of the protocol is done in a precise manner, and the precise sequencing constraints are in particular taken into account. A discussion on the respective advantages of these different approaches can be found in [4] for example. In any case none of these approaches has been applied, to our knowledge, to electronic commerce protocols. We believe this would require significant extensions or revisions for most existing formal approaches as electronic commerce protocols guaranty more elaborate properties then authentication protocols do.

The objective of this paper is to show how such extension can be done using the work in [4] as a starting point. The revisited approach has first been applied on the C-SET protocol, the smartcard based payment protocol proposed by GIE CB, an organization sponsored and owned by major french banks, and is currently being applied on the SET (Secure Electronic Transaction) protocol, proposed recently by Visa and MasterCard. The presentation is organized as follows. In the two next sections we present the problem basics and introduce as an illustration

an hypothetical payment protocol, which although much more simple then SET and C-SET, presents many similarities with these two protocols. In Sections 5 we recall the main steps of the approach presented in [4]. In Sections 6 and 7 we show how the approach can be extended to express payment properties. In section 8 we describe the verification itself by means of some examples.

3 Problem Basics

Encryption is the transformation of data into a form unreadable by anyone without a secret decryption key. Decryption is the inverse function, which recreates the original data in its form prior to encryption A cryptographic key system is said to be symmetric if, and only if, the same key can be used for both encryption and decryption. A cryptographic key system is said to be asymmetric when different keys are used for encryption and decryption. In that case the encryption key is called a public key, while the decryption key is called a private key. The public key can be made available to anyone, while the private key can be used only by its legitimate owner. If K_a is the encryption key, then K_a^{-1} is the corresponding decryption key. An asymmetric cryptographic key system can also be used to provide a digital signature. A digital signature makes it possible to prove to anyone that some data has been issued by an entity. A digital signature is obtained either by adding some data redundancy to the data to be signed or by condensing the data into a small value that is characteristic of the data and then encrypting this value using a private key. Anyone that knows the corresponding value of the public key can then verify the signature. In that case, these keys are sometimes called, respectively, a signature key and a verification key. Some asymmetric algorithms can be used to provide data encryption or a digital signature (e.g., the RSA algorithm), while some others can be used only to provide digital signatures (e.g., DSA algorithm).

4 A Payment Protocol

In this section we provide, as an illustration, an hypothetical payment protocol which although quite different and much more simple then the SET or C-SET protocols, encompasses most of the specificities and difficulties encountered in the verification of electronic commerce protocols such as SET and C-SET.

The proposed protocol which corresponds to the purchase and authorization phases is invoked after the cardholder has completed browsing, selection and ordering. Before this flow begins, the cardholder will have been presented with a completed order form and approved its contents and terms. As a result of this phase which is supposed to be performed out of bound, the cardholder has a valid order description (OD) and a valid payment description (PD). The protocol is described in the figure below where $Transaction$ stands for the sextuple $(C, M, (Lid\_c, Lid\_m), Purchamt, hash(Od), hash(Pd))$ whose components will be commented in the sequel.

$(PInitReq)$	$C \rightarrow M : (C, M)$
$(PInitRes)$	$M \rightarrow C : Lid\_m$
$(PReq)$	$C \rightarrow M : ((Transaction)_{Kc^{-1}}, (Od)_{Km}, (Pd)_{Kg})$
$(AuthReq)$	$M \rightarrow G : ((Transaction)_{Kc^{-1}}, (Transaction)_{Km^{-1}}, (Pd)_{Kg})$
$(AuthRes)$	$G \rightarrow M : (Results, (hash(Transaction)))_{Kg^{-1}}$
$(PRes)$	$M \rightarrow C : (Results, (hash(Transaction)))_{Kg^{-1}}$

PInitReq: The message is sent by the cardholder, i.e. C, to the merchant, i.e. M, to request a fresh and unique local transaction identifier (i.e. Lid_m) from the merchant.

PInitRes: When the cardholder receives the requested identifier, he also assigns a fresh and unique local transaction identifier (i.e. Lid_c) to the transaction, so as to form a unique transaction identifier, i.e. (Lid_c, Lid_m), (in SET and C-SET some additional information is added and the resulting aggregate is called $Transid$).

PReq: The transaction data, i.e. $Transaction$, is then composed by the cardholder and includes in particular $Purchamt$, the purchase amount, C, the cardholder identity, and M the merchant identity. The transaction data is then signed using the cardholder private key K_c^{-1}, and is sent together with the order description and the payment description. The order description should only be known by the merchant and is thus encrypted using the merchant public key. The payment description should only be known by the payment gateway G (i.e. the representative of the bank) and is thus encrypted using the payment gateway public key. On reception of $PReq$, the merchant decrypts and validates Od, checks that the same Od was used for the composition of the $Transaction$ data aggregate, and furthermore validates M, Lid_m, and $Purchamt$. The merchant also checks that the signature of the message belongs to the cardholder whose identity appears in $Transaction$ (i.e. C).[1] The merchant finally stores the signed $Transaction$ data together with the decrypted Od.

AuthReq: The merchant then signs the same $Transaction$ data using his own private key, K_m^{-1}, and sends it together with $(Transaction)_{K_c^{-1}}$ and $(Pd)_{K_g}$ as received from C. When the Payment Gateway receives the authorization request, it verifies that the same data was signed by the merchant and the cardholder as specified by C and M components of $Transaction$.

AuthRes: The Payment Gateway then formats and sends an authorization request to the Issuer via a payment system. Upon receiving an authorization response from the Issuer, the Payment Gateway generates and digitally signs an authorization response message. When the cardholder system receives the purchase response message from the merchant, it verifies signature and conformance with the previous purchase request.

PRes: The same authorization message is then sent to the cardholder who also verifies signature and conformance with the previous purchase request.

5 Approach

5.1 Modeling Strategy

First the different principals involved in the process must be identified. Principals receive messages at one end and emit other messages at another end. Some principals will be considered to be "trustable" (i.e., work according to their role in the protocol) and some not. Communications media are typically considered to be non-trustable, because messages can usually be intercepted, replayed, removed, or created by intruders. We will consider that this is the case in the following

[1] The distribution of keys is not part of this protocol.

discussion. The same protocol can be studied in terms of many different hypotheses. Trustable principals will always be assumed to play their roles as stated by the protocol. For non-trustable principals, the situation is very different, because we do not know how many there are, how they work, or how they cooperate, and so the worst situation has to be assumed. So instead of modeling each non-trustable principal separately, and instead of describing how they work, the combination and cooperation of all such principals is modelled as a unique agent which is called it the "external world" or, more concisely, the intruder. The intruder is modelled as a principal that may know some data initially and that will store and try to decrypt all data passed to the malicious principals and thus all information circulating on the communications media. The intruder will also be able to encrypt data to create new messages that will be sent to mislead other principals. But the intruder will be able to decrypt and encrypt data only with keys he or she knows. This modeling will in particular allow us to determine at any time which data are potentially known to the intruder under the chosen "trustability" hypothesis. The selection of such hypotheses will be discussed and illustrated in section 6.

5.2 Exploitation of Knowledge

The knowledge of the intruder is formalized as a set of data components. The intruder will be able to deduce new data from available data by using four basic operations; namely encryption, decryption, and two other operations for handling clear-text data structures. These operations have been formalized in [4]. The exploitation of a given knowledge (i.e., a given set of data) to deduce new information (i.e., new data) consists of the application of one or more of the four basic operations, in any order and any number of times. A set of data s' (or a single data element) is said to be deducible from a set of data s if, and only if, there exists a sequence of applications of the four basic operations that can obtain s' starting from s. The predicate $known\_in$ is used to formalize this rule and the previous fact is noted $s'\ known\_in\ s$. A set of rules based on the predicate $known\_in$ is then derived from this formalization. Appendix shows some of these rules for a restricted set of encryption and decryption operators (other operators such as the hash operator are introduced in the same way). These rules make it possible to implement decision procedures (i.e., tools) to determine if a given data may be known by the intruder or not. But in the most general case such facts are to be proved in an interactive manner using formal provers as shown in [3]. An example of such a rule is $c_k\ known\_in\ s \wedge\ k^{-1}\ known\_in\ s \Rightarrow c\ known\_in\ s$ which states that if the a principal knows (i.e. can deduce) both the encrypted component c_k (i.e. c encrypted using the key k) and the inverse key k^{-1} of k, using the set of data s, then he knows (i.e., can deduce) the component c using the same set of data s.

5.3 Formalization of the Protocol

We then need to formally specify the protocol itself. This specification consists of describing the role of each trustable agent. The formal specification of the protocol consists of a set of atomic actions. The sending and reception of a message are not synchronous. Consequently the transmission of a message is considered as two atomic actions, one for sending and one for receipt. This corresponds exactly to the decomposition used in the SET as well as in the C-SET specification documents. Our modeling of the previous payment protocol will thus distinguish 12 different kinds of atomic actions, two for each message type. These actions will be identified using the labels $PInitReq_C$, $PInitReq_M$, $PInitRes_M$, $PInitRes_C$, $PReq_C$, $PReq_M$, $PRes_M$, $PRes_C$, $AuthReq_M$, $AuthReq_G$, $AuthRes_G$, $AuthRes_G$. Each of the 12 labels M_X stands for

the sending or reception of message M by principal X: $PInitReq_C$ stands for example for the sending of message $PinitReq$ by the principal C. [2]

As part of its role, the cardholder will, for example, initiate payment session by sending a $PInitReq$ message, i.e. by performing a $PInitReq_C$ action, and will then receive a $PInitRes$ response message, i.e. perform a $PInitRes_C$ action, and then will complete the session with a $PReq_C$ action followed by a $PRes_C$ one. Each atomic action also involves some specific processing (verification, generation, storing of values, computation, etc.) which is described as for any piece of code in an implementation language, by describing its precise test, computation and effects on the state variables. In order to facilitate verification and also to specify the protocol at the required level of abstraction, a logic-like language is used for this purpose (c.f. [4]). The first step is thus to identify and define the state variables that are to be used by each principal in order to "implement" the protocol. The only difference from a low-level description is that we may use more abstract types, such as sets or sequences, and that we do not need to consider storage or efficiency problems.

More precisely, the formalization is based on the chemical abstract machine paradigm [1]: a system is described as a set of atomic actions which may be applied repeatedly, in any order and whenever pre-condition holds. Let $\mathcal{S}$ stand for the domain of global states. In the case where the cardholder, the merchant and the gateway are all trustable, $\mathcal{S}$ will quite naturally be defined as $\mathcal{S}_C \times \mathcal{S}_M \times \mathcal{S}_G \times \mathcal{S}_I$ where each component describes the domain of local states for each trustable principal and for the intruder: C for the particular cardholder, M for the particular merchant, G for the particular payment gateway, and I for the intruder. In [4] the specification is defined as a single predicate r on $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ binding the state before application to the state after application. The domain $\mathcal{A}$, is the set of action labels. We use instead a relation on $\mathcal{S} \times (\mathcal{A} \times \mathcal{M}) \times \mathcal{S}$, where $\mathcal{M}$ is the domain of messages: $r(s,(l,m),s')$ if and only if the global state s is modified into s' upon firing of action labelled l for the sending or receipt of message m. The information on the content of the message which is added here will be exploited in section 6 in the expression of security properties. The domain $\mathcal{M}$ is defined more formally as the domain of data components, and $\mathcal{S}_I$ as the domain for sets of data components. Both are formally defined in [4].

Intuitively the state of the intruder is the set of data components that have been listened to on the communication line. Each sending of message m augments the knowledge of the intruder, i.e. $s'_i = s_i \cup m$ where s_i is the state of the intruder before firing of the action and s'_i is the state after. The receipt of a message m does not change the state of the intruder. According to the modelisation all messages received by a trustable principal are produced by the intruder (the communication media are part of the intruder). Thus, m $known\_in$ s_i, is a precondition to the reception of message m.

6 Expressing Properties

Electronic commerce protocol have to address standard requirements such as confidentiality of information; integrity of data; cardholder account authentication; and merchant authentication. These requirements and corresponding functions are not specific to electronic commerce protocols. For all such basic and standard properties the approach defined in [4] can be applied without any adaptation.

[2] C, M, G, stand respectively for cardholder, merchant and payment gateway.

6.1 Extending the approach

But electronic commerce protocols also involve some significant specificities that are responsible for additional difficulty in the expression of properties, when compared to more traditional authentication protocols. The first one comes from the fact that the principals involved in an electronic commerce transaction have rather different expectations than the one involved in authentication sessions. The merchant is for example not only interested in making sure that he has really been talking to the proper gateway at the time he completes a session. He also wants to make sure that the transaction cannot be refuted afterwards, due to some misunderstanding between the parties. The second source of difficulty comes from the fact that payment protocol involves three-party decisions. In the case of authentication protocols, interactions can usually be considered as separate two-party agreements, even when such agreements are interleaved: the requester and the resquestee involved in an authentication session will typical interact with some other entity such as a key server during a session, but each interaction is to fulfill a specific function, such as obtaining a correct and fresh key for the interaction with key server. In the case of electronic commerce protocols it is not sufficient for the gateway just to make sure that the information produced by the cardholder or the merchant has the desired confidentiality, or integrity properties. It also needs to be sure that both correspond to a coherent, non ambiguous and real three-party deal. In other words, a typically electronic commerce protocol should implement an abstract atomic transaction between three principals. In the sequel of this section we address these two specific problems and show how to extend the approach presented in [4] to handle electronic payment protocols.

The revised approach is presented as the combination of 5 modeling decisions. The first decision was already part of the approach proposed in [4]. The second one and the fourth one are quite direct generalization or extension of decisions that were already implicit in the original approach. The two other decisions are introduced here.

6.1.1 Adopting the view of a global observer

The first decision is indeed to adopt the view of a global observer which has a correct and complete view of the system. This decision is taken in many approaches and is a quite natural one: weaknesses or flaws of protocols are often reported using this same perspective.

6.1.2 Breaking down security properties into agent centric properties

The second decision is more of methodological nature and amounts to break down properties into more basic properties that express the belief of a particular principal at a particular point in the session. Confidentiality properties are not amenable to this kind of decomposition as they are of more global nature, but their expression is straightforward as will be shown in the next section. The proposed decomposition allows for a more systematic identification and classification of properties. A practical way of obtaining this decomposition is to consider each role in turn, and to identify in each case and at each possible point during a corresponding session the belief and expectation of the principal about what really happened. Particular points of interests are the beliefs at the end of a session. Typically such beliefs need only be identified in the case where the session proceeds normally from the point of view of the principal at hand, as whenever an incorrect message is received, the principal will generally refuse to continue with this particular session. In the case of authentication protocols only two different kinds of principals had to be considered, the initiator of the authentication session and the requested principal. The two corresponding

properties where called the master and the slave properties. In the case of payment protocols the beliefs or expectations of three different kinds of principal need typically to be formalized: the cardholder, the merchant and the payment gateway. The cardholder wants, for example, to make sure that his bank account is only debited for the purchases he did accept, and with the correct amount. The merchant wants to make sure that the authorization he receives from the gateway is really done for the purchase he agreed with the cardholder. The gateway wants to make sure that he is giving an authorization for a coherent purchase agreement. Such properties will be referred to as the cardholder, merchant or gateway properties. Most protocols will have to satisfy all such kinds of properties.

6.1.3 Verification under various trustability hypotheses

In the approach presented in [4], a property is verified under the assumption that some principals are trustable (i.e. behave according to their role in the protocol) and some are not, or more precisely that they cannot be considered to be trustable. The selection of trustability hypotheses is done so as to minimize the number of hypotheses. Let us consider in order to illustrate this, the case of an authentication protocol involving three different roles, a master (a principal requesting an authentication session), a slave, and a key server. Let us then consider the following master property: any principal A requesting an authentication session with B, wants to be sure at the end of its (master) session that he has really been talking to B. In order to verify this statement we will do the verification under the assumption that both A and B are trustable. The particular key server used during the session will also be considered to be trustable if this is required to satisfy the property. The proof of this property will guaranty that whenever B and the key server are trustable and A has performed a session intended to be with B, then A can be sure he has really been talking to B. But A knows on the other hand that this is only true if B is trustable. If this is not the case then the previous security property is not guaranteed: if for example B happens to be compromised, then B may well give his private key to a third principal which is then in a position to impersonate B. If there is a doubt about the importance or validity of one of the hypotheses it is possible to relax some of them and perform the same verification again. Thus the use of various trustability hypotheses is only meant to test the importance and effect of selected hypotheses, such as for example the importance of having a trustable key server.

In the case of payment protocols the same considerations apply but are not sufficient. It is for example not sufficient for the gateway to know that whenever the cardholder and merchant involved in a particular transaction are trustable then the gateway can be sure that both the cardholder and the customer have agreed on a non ambiguous and coherent transaction. It is also important for the gateway to be able to identify responsibilities and commitments in an unambiguous manner, in the case of dispute. Worst, it is not sufficient for the gateway to be able to identify responsibilities. It may also be important to demonstrate this to other parties (e.g. the court) which will not take the trustability of the gateway for granted. The gateway thus needs to keep and provide irrefutable proofs a posteriori.

In other words, an authentication protocol is typically only meant to provide some level of confidence to participants of an authentication session in the case where the two principals that try to authenticate themselves are *trustable.* Trustable principals should not be mislead by compromised principals that are supposed to be external to the particular session. An electronic commerce protocol is meant to provide *in addition* some kind of assurance even in the case of a deal involving corrupted principals.

In order to tackle these new problems we will use again the same technique (i.e. verification

under different trustability hypotheses) but the selection of hypotheses will be done in a different way. Indeed, even in the case where the trustability of principal A_1, .., A_n is required for the transaction to proceed normally, we will have to consider some *weaker* sets of hypotheses, and derive from these various hypothetical situations various beliefs that the protocol provide for each principal. In the case of the previous example, it is clear that a proper deal should involve at least three trustable principals: a client, a merchant, and a payment gateway. In the case of an authentication protocol this would be the main set of hypotheses to consider. Here this set of hypotheses is of course to be considered and will be referred to as the *main set of hypotheses*. But we will also need to carry the analysis under weaker sets of hypotheses, namely:

(1) the particular gateway and the particular merchant are the only trustable principals,

(2) the merchant is the only trustable principal,

(3) the gateway and the particular cardholder are the only trustable principals,

(4) the cardholder is the only trustable principal,

The first set is used to prove important merchant and gateway properties that are complementary and different from merchant and gateway properties expressed using the main set of hypotheses. An important gateway property that can be expressed under this weaker set of hypotheses is indeed the guaranty that the gateway has enough evidence, at the time when he receives the merchant request, that the merchant has taken an unquestionable commitment towards a new transaction with a valid cardholder (independently of cardholder trustability). In the protocol of section 4 this is intuitively achieved by the fact that the data structure that unambiguously describes the transaction is signed by the merchant. As another example, we might consider the merchant property that guarantees to the merchant on reception of message *AuthRes* that the gateway has taken an unquestionable commitment towards a new transaction with a valid cardholder (independently of cardholder trustability), and information passed by the client is coherent with that passed by the merchant. These two commitments are complementary with properties that can be expressed using the main set of hypotheses, for example the fact that whenever the commitments of the merchant and of the cardholder are guaranteed then they are are coherent (i.e. compatible): the situation where the trustable merchant believes that he fixed a given price and the trustable client believes that he agreed on another price is impossible, even in the presence of other corrupted principals.

The second weak set hypotheses is used to prove that the gateway will have enough evidence of the merchant commitment, in the case of dispute between the merchant and the bank: the gateway will in this case provide information received from the merchant, and it should be possible to prove the merchant commitment without making the assumption that the gateway is trustable: in the protocol of section 4 the data signed by the merchant provides this proof (the gateway or any trustable or non trustable principal can only get this signed data if the merchant has really made a commitment for the particular transaction at hand).

The third and the fourth sets of hypotheses are used to prove the commitment of the cardholder and are similar to the two first.

All such aspects are complementary. If the first one was missing, their would still be some room for the merchant to argue that the cardholder has not behaved according to its role in the protocol (is not a trustable principal). Although the merchant would not have any evidence of this, there would not be either any evidence (i.e. proof) to discard this possibility. The second and fourth

sets of hypotheses may be irrelevant for protocols which do not provide to the gateway the ability to prove its trustability in case of dispute. Even in the case where such ability is provided, this can only be done a posteriori with the help of another trustable principals (e.g. the court) which will have access to all data provided. During the session, both the client and the merchant have to rely on the gateway trustability: all protocols that we studied so far were falling in this category. Thus some particular combination of hypotheses may bring little information on principal belief. Such combinations were omitted here for the sake of conciseness: for example when the cardholder and the merchant are the only trustable principals, very little can be guaranteed to the merchant or the cardholder; the cardholder has no evidence that the merchant has agreed on the same transaction even if he receives a valid authorization response; the merchant does not know if the commitment of the cardholder is valid.

To sum up, the general approach we propose here is to consider each possible set of trustability involving A_1, .., A_n (instead of only considering the main one for which A_1, through A_n are trustable), and express the various gateway, cardholder or merchant properties that are to be satisfied in each case. In doing so we may of course have to skip situations for which nothing interesting can be guaranteed.

As a result of this splitting into many sets of trustability hypotheses, a new global state domain, $\mathcal{S}$, is to be used in each case. This construction is automatic and based on section 5.3 principles: $\mathcal{S} = \mathcal{S}_C \times \mathcal{S}_M \times \mathcal{S}_G \times \mathcal{S}_I$ for the main set of hypotheses, $\mathcal{S} = \mathcal{S}_M \times \mathcal{S}_G \times \mathcal{S}_I$ for the first set of hypotheses, etc. The same is true for the predicate r. If an unbounded number of trustable principals of the same type needs to be considered (e.g. a chain of trustable payment gateways), then functions are used to associate particular principals identities to principal local states in the global states (e.g. $\mathcal{S} = \mathcal{S}_M \times (Id \rightarrow \mathcal{S}_G) \times \mathcal{S}_I$) but the general approach is left unchanged.

6.1.4 Focussing on "correct" actions for a specific session

As a result of the specific modeling of the intruder knowledge only actions performed by trustable principals are explicitly considered: potential processing of data by compromised principals or intruders is indeed taken care of by the specific axiomatization of intruder knowledge proposed in [4] (see Appendix). Now the approach also provides the ability to express security properties on *partial views* of the global system, i.e. some actions are abstracted away, based on their type, or/and based on the values that they carry (see [4] for a justification of this feature). In other words the approach implicitly makes use of two kinds of abstraction functions. The first one abstracts away actions performed by non trustable principals, and the second one allows to focus on any particular session and on the correct actions expected for this session: unexpected actions or actions carrying non coherent values will be abstracted away, and the security property can be more concisely and more simply expressed on remaining actions, which are called the visible actions. This second abstraction function will be called the *filtering function* in the sequel, as it also performs some renaming of visible actions. The first abstraction is meant for proof simplification. The filtering function is meant to simplify the expression of security properties. But none of these two abstractions results in any loss of precision or in any kind of approximation: actions that are abstracted away are still modelled and taken care of in the proof.

6.1.5 Using a finite automaton and a filtering function to describe each property

In the following we extend the use of partial views by revisiting and formalizing the filtering function implicitly proposed in [4] and by using it within a more general framework.

For this we need to introduce some notations and definitions. The filtering function will thus be noted $ff_x(y)$ where x is used to parametrize[3] the function and y ranges in the domain of actions. The range of the function will be finite (for any given instance of x), and the particular value $\perp$ will be given a particular meaning: if $ff_x(y) = \perp$, then y is to be abstracted away; otherwise y is to be replaced by $ff_x(y)$. Thus if t is a finite trace of actions we will define $ff_x(t)$ recursively as: $ff_x([]) = []$ and $ff_x([y|t]) = ff_x(t)$ if $ff_x(y) = \perp$, and $ff_x([y|t]) = [ff_x(y)|ff_x(t)]$ otherwise.

According to the modelisation proposed in [4], the system behavior can be described as a set of finite traces[4] T. Each trace in T models one of the potential *global* and finite behavior of the system up to a given point in time. Finite as well as infinite behaviors are thus modelled using all their finite prefixes and the set T is closed by prefix inclusion (any prefix of a run is also a run). Each trace in T models a sequence of actions the global system may go through. We propose to describe each security property (other then confidentiality properties and similar ones, which can be described as simple invariant properties) using a filtering function $ff_x(y)$ and a regular language L: the property is satisfied if and only if:

$$\forall x. \forall t. t \in T \Rightarrow ff_x(t) \in L$$

Since T is prefix closed we can use prefix closed regular languages L. We will describe L using either a finite automaton A or a regular expression A and we will consider by convention that L is defined as the reflexive transitive prefix-closure of the language accepted by A.

As a simple example let us suppose we want to express that the same action is never played twice: we just need to use a as the regular expression describing the regular language L (i.e. $L = \{[], [a]\}$ by reflexive transitive prefix-closure) and $ff_x(y) = \perp$ if $x \neq y$ and $ff_x(y) = a$ if $x = y$ where a is just an arbitrary symbol. If there exists some trace t in T such that the same action, let us say[5] 3, is played twice, for example $t = [5, 4, 3, 6, 3]$, then there exists a value of x, namely 3, for which $ff_x(t) = [a, a] \notin L$.

For illustrating purposes we now consider a more elaborate example, namely the main gateway property under the first set of hypotheses. More precisely we will try to express the belief of the gateway at the time when it receives an authorization request. Let us thus consider a particular gateway G and an action, let us say α, on which G receives a new authorization request (G will receive many such requests and we consider here one of them). According to the protocol modelisation described in section 5.3, α is a triple $(state_\alpha, (label_\alpha, message_\alpha), state'_\alpha)$, where $label_\alpha = AuthReq_G$. The message will contain the identification of a particular merchant, let us say M, together with the identification of a particular client C. M and C are considered by G to be the initiators of the deal.

Here we are under the assumption that M and G are the only trustable principals. The main gateway property to consider under this set of hypotheses is that G can be sure at the time when he receives the authorization message that this message really originates from M and has not been tempered with: in other words there is another action, $\beta = (state_\beta, (label_\beta, message_\beta), state'_\beta)$,

[3] More formally, the filtering function ff has the form $\lambda x.\lambda y.body$, and ff_x has the form $\lambda y.body$.

[4] The use of traces of actions results in some data redundancy here as the same state appears twice: once as the state after of a given and once as the state before of the next action. This could easily be avoided but at the expense of some loss of conciseness in the definition of the filtering function.

The use of finite traces (as opposed to infinite trees for example) does not result in any loss in expressiveness here and is justified by the absence of invisible actions (i.e. τs), and by the fact that security properties of concern are all safety properties: denial of service properties that would be liveness properties are not considered here as well as in other formal approaches.

[5] For this first illustration we oversimplify the situation by considering that actions range in Nat domain.

such that $label_\beta = AuthReq_M$ and $message_\alpha = message_\beta$ which has preceded α, and neither α or β is repeated. In order to formalize this we just need to use the regular expression $AuthReq_M$ $AuthReq_G$ together with the following filtering function:

$$
\begin{array}{ll}
ff_{message} & ((st,(l,msg),st')) = \\
& AuthReq_M \ if \ l = AuthReq_M \wedge message = msg \ \wedge st'.mcht.gateway = G \\
& AuthReq_G \ if \ l = AuthReq_G \wedge message = msg \ \wedge st'.gtw.merchant = M \\
& \bot \ \text{otherwise}
\end{array}
$$

where st' is the global state after the execution of the action; $st'.mcht$ (resp. $st'.gtw$) is the part of this global state that corresponds to the local state of the particular merchant M (resp. the particular gateway G), and $st'.mcht.gateway$ (resp. $st'.gtw.merchant$) refers to the field $gateway$ (resp. $merchant$) in this local state that holds the identifier of the particular gateway (resp. the particular merchant), M (resp. G) *believes* he is talking to. If the message is tempered with, and an incorrect value is received by G, let us say $wrong\_msg$, then the problem will be pointed out with the particular parameter value $wrong\_msg$ as the trace t that corresponds to the problematic scenario will be such that $ff_{wrong\_msg}(t) \notin L$ (i.e. $ff_{wrong\_msg}(t) = AuthReq_G$). In a similar way if a message $messageforG'$ has been sent to another gateway, let us say G', and G believes it has been sent to him, then if trace t stands for this problematic scenario then $ff_{messageforG'}(t) = AuthReq_G \notin L$. Furthermore, if the same message can be replayed by the intruder and is accepted unnoticed by G then there exists x and t such that $ff_x(t) = AuthReq_G \ AuthReq_G \notin L$ (x is the content of the message and t the problematic scenario).

The property expressed so far is in fact too strong for the protocol of section 4 as evidenced by the following scenario: a wrong value is passed for $(Pd)_{Kg}$ on action $PReq_M$; M has no way of detecting this and uses this wrong value for $AuthReq_M$; the intruder then replaces the wrong value of $(Pd)_{Kg}$ by the correct one so that a correct authorization message is received and checked by G. For this particular scenario t, taking the authorization message effectively received by G for the value of x, leads to $ff_x(t) = AuthReq_G \notin L$. This scenario is not problematic for the transaction in itself. Thus, instead of revisiting the protocol by adding some kind of link in the message so as to prevent this kind of scenario we decide here to weaken the property: in the new version we will only require that the two first components of the message are not tempered, and not necessarily the whole message. The third component is indeed just forwarded by M and could be forwarded by any other mean: the checking of this component is under G responsibility. The new filtering function thus becomes:

$$
\begin{array}{l}
ff_{(c,lid\_c,lid\_m,amt,hod,hpd)}((st,(l,(sg1,sg2,enc),st')) = \\
\quad let \ Trnsc = (c,M,(lid\_c,lid\_m),amt,hod,hpd),\sigma_1 = Trnsc_{K_c^{-1}},\sigma_2 = Trnsc_{K_m^{-1}} \ in \\
\qquad AuthReq_M \ if \ l = AuthReq_M \wedge sg1 = \sigma 1 \wedge sg2 = \sigma 2 \ \wedge st'.mcht.gateway = G \\
\qquad AuthReq_G \ if \ l = AuthReq_G \wedge sg1 = \sigma 1 \wedge sg2 = \sigma 2 \ \wedge st'.gtw.merchant = M \\
\qquad \bot \ \text{otherwise}
\end{array}
$$

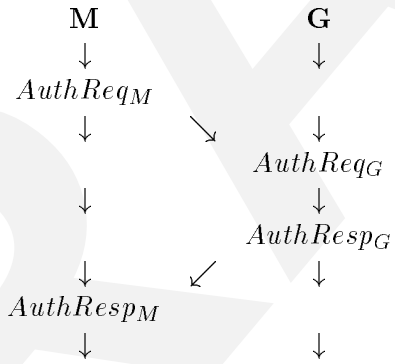
Intuitively a filtering function is to express the belief that a principal can have at a given point while playing his role about what really happened concerning actions related to the current session. Thus the parameter for the filtering function is chosen so as to provide enough information to characterize the last expected visible action (the one after which the belief is implicitly expressed) but also to characterize the other visible actions that should precede. In practise we would expect this parameter to range in a restricted domain of values: hod should for example be a valid hash

which we could express by using *od* as a component of the parameter of the filtering function and $hash(od)$ in the body of the function; similarly not all combinations of *hod* and *amt* are feasible. But there is usually no need for expressing such constraints as in the case where improper or incoherent value are used the filtering function will return [] which by construction belongs to L.

The gateway property expressed so far is only concerned with the $AuthReq_M$ as, in the case of the protocol of section 4 (but also in the case of both C-SET and SET), $AuthReq_M$ is the particular action during which the merchant takes its transaction commitment. But we could very easily if necessary express additional requirements on the actions $PInitReq_M$ and $PInitReq_M$ that the merchant necessarily performs in order to prepare the payment request. The corresponding filtering function and previous regular expression can be obtained by a slight modification of the previous ones.

More interestingly, let us consider the same gateway property under the general set of hypotheses, i.e. C, M, and G are trustable. We would typically want to express that the authorization request always follows a coherent purchase request $PurReq_C$ and thus uses $PurReq_C\ AuthReq_M\ AuthReq_G$ as a regular expression together with a new filtering function. In fact the new filtering function can easily be obtained from the previous one by associating $PurReq_C$ to actions whose label is $PurReq_C$, whenever these actions carry values that are coherent with the transaction at hand: all actions are coherent if the value they carry refer to the same $Transaction$ data (with the exception of $(Pd)_{Kg}$ for actions $PReq_M$ and $AuthReq_M$ as explained previously).

As another illustration let us consider the merchant property under the first set of hypotheses (M and G are trustable). We use $AuthReq_M\ AuthReq_G\ AuthRes_G\ AuthRes_M$ as a regular expression. This corresponds to the following partial order (which happens to be total here):



In this particular situation the coherent response from P must fall within a given time frame delimited by the two last events performed by M (i.e. $AuthReq_M$ and $AuthResp_M$). This particular situation was the specific case addressed in [4], and was called a *window property*. It was formalized and verified in an ad-hoc manner. This situation is only a very particular case for electronic commerce protocols, where more elaborate three-party flows of messages are involved.

As for any formal verification one has to be convinced that the formalized property is adequate. Another user might well prefer another formulation, and it is always possible to use redundant properties in case of doubt. The interesting aspect here is that a property can be expressed with simple and well-defined basic notions such as the ordering of events. The more surprising fact is that, for the large variety of protocols formalized with this approach, complex properties can be expressed using quite simple automata.

7 Verification

7.1 Proving confidentiality properties

Confidentiality properties are expressed and verified as simple invariants. The verification of such properties amounts to verify that the various invariants are preserved by each action of the protocol. In the case of protocol at hand it is essential that private keys remain confidential. The property is formalized easily by writing that $\neg(K^{-1}\ known\_in\ s_I)$ is an invariant of the protocol, where s_I stands for the knowledge owned by the intruder (i.e. the state of the intruder according to our formalization), and where K^{-1} is the private key under consideration. In other words, the invariant property expresses that the key cannot be deduced from the intruder knowledge. Once the invariant property is expressed, two standard distinct verifications need to be done in order to prove invariant preservation. First we need to make sure that the invariant property is initially true. If this initial property is found acceptable, it will be taken and registered as an assumption under which the protocol is supposed to work. Someone in charge of reviewing the verification process of the protocol will typically make sure that all such assumptions are acceptable. The same protocol can of course be analyzed under various sets of assumptions, so as to assess the importance of some problematic assumptions. The second kind of verification that needs to be done is to check that each possible action involved in the protocol preserves the invariant. Such verifications have been formalized and detailed in [4]. We only summarize the main steps. For each of the 12 protocol actions we need to prove that the invariant is preserved. More formally, each proof obligation has the form $\forall s, s', m. inv(s) \wedge r(s, (l_i, m), s') \Rightarrow inv(s')$ where inv is the logic formula that expresses the invariant, r is defined according to section 5.3, and l_i is the label of the i^{th} action. In the case of the particular property at hand the verification is straightforward and basically uses axiom (D1),(D2) and (C2) to (C4) of Appendix: more informally, the confidentiality of K^{-1} is trivially guaranteed because no key is ever sent in clear or in encrypted form; thus even if the intruder is able to decrypt all messages he will never succeed in obtaining K^{-1} as a result of such processing. More account on such proofs can be found in [4].

7.2 Proving more elaborate properties

In the section 6 we have discussed how to formalize more elaborate properties using both an automaton and a filtering function to describe expected constraints on behaviors. Automata turn out to be very simple in terms of the number of states, and it is thus a good strategy to discharge them using invariant techniques.

According to section 6, each property is expressed using a filtering function ff_x and a finite automaton. Let n be the number of states of the automaton. The user has to provide one *temporary* invariant inv_i for each automaton state $i \in 1..n$. Each of these invariants is defined on the domain of global states $\mathcal{S}$ introduced in section 5.3. These invariants may use an additional parameter which has the same structure and meaning as the parameter x used for ff_x. Proofs obligations are then generated automatically. They are discharged using the techniques illustrated in the previous section.

In the sequel we justify the proposed verification approach. By convention we will let 1 be the automaton initial state. All states 1 to n are final states by construction: such an automaton can be obtained directly and automatically from a regular expression or from any finite automaton by applying simple reduction steps; the existence of such a normal form is due to the fact that the accepted language is prefix closed [8]. The first step in the verification process is

to transform the automaton into a deterministic automaton. This step which is of course automatic will add exactly one state to the automaton[6]. This state which is the only non final state will be noted $n+1$ by construction. It is an absorbing state: all transitions from $n+1$ lead to $n+1$. By conventions the labels of the transition from i to j will be called l_{ij}[7]. States of the final automaton will be called meta-states in order to avoid confusion with the global states used for the protocol modelisation. The second step in the verification process is to incorporate the value of the current meta-state in the system modelisation itself in a way that does not affect the behavior of the system: this addition can be seen as the test performed by the global observer to check whether all behaviors conform to the property at hand. The test is rather simple: the meta-state $n+1$ should not be reachable, as this state precisely models the failure for the filtered global behavior to conform to the described automaton. The global states, which range in $\mathcal{S}$, are thus extended using an extra global variable *meta_state*. This variable is used to store the current meta-state. The value of the new *meta_state* variable depends on the global history. Let us consider a particular global history and it modelisation using the particular trace t. The sequence of visible actions that has been traversed is, according to section 6.1.6, $ff_x(t)$ for any particular value of x (x characterizes the particular session of interest). The value carried by *meta_state* should thus be the state of the automaton reached upon input of sequence $ff_x(t)$. More practically, the *meta_state* will be initialized with the initial state of the automaton (i.e. 1), and will change each time a visible action is fired (i.e. each time $ff_x(action) \neq \perp$) and the state change is done according to the specified automaton. Thus if the protocol specification described in section 5.3 is expressed using predicate r, the revised relation for the particular property at hand will be r' defined as $r'((s, meta\_state), (l, m), (s', meta\_state')) = r(s, (l, m), s') \wedge ((ff_x(s, (l, m), s') = \perp \wedge meta\_state = meta\_state') \vee \exists i, j.(ff_x(s, (l, m), s') = l_{ij} \wedge l_{ij} \neq \perp \wedge meta\_state = i \wedge meta\_state' = j))$. Within this framework, the property specified using the particular automaton and filtering function, amounts to verify the very simple invariant: $meta\_state \neq n+1$. This invariant needs typically to be reinforced into a stronger invariant in order to be proved inductively. For this we use the n temporary invariants proposed as a result of the previously mentioned user interaction. The new reinforced global invariant inv is defined on $\mathcal{S} \times Nat \times C$ where $\mathcal{S} \times Nat$ is the new global state and C is the domain of potential values for the parameter x of ff_x. It is defined as $inv(s, meta\_state, x) = meta\_state \in 1..n \wedge inv_{meta\_state}(s, x)$. In other words $\forall x.inv(s, 1, x)$ should be true for the initial values of s and $\forall s, s', x, ms, ms'.inv(s, ms, x) \wedge r'((s, ms), (l, m), (s', ms')) \Rightarrow inv(s, ms, x)$ should be satisfied. In fact this latter proof obligation can be transformed into the two following more intuitive proof obligations that are here implicitly universally quantified over their free variables:[8]

$$(1) \quad r(s, (l, m), s') \wedge ff_x(s, (l, m), s') = \perp \wedge inv_i(s, x) \wedge i \in 1..n \Rightarrow inv_i(s', x)$$
$$(2) \quad r(s, (l, m), s') \wedge ff_x(s, (l, m), s') = l_{ij} \wedge inv_i(s, x) \wedge i \in 1..n \Rightarrow j \neq n+1 \wedge inv_j(s', x)$$

The first one specifies that if no visible action is fired then the temporary invariant is preserved. The second one specifies that if a visible action is fired then it is an expected one (i.e. the transition

[6] The transformation into a deterministic automaton adds exactly one more state, in all cases but the trivial one situation in which the automaton is already determistic and the property is trivially satisfied: $L = \Sigma^*$ where Σ is the alphabet of labels.

[7] It is assumed in this document, for the sake of concisness in mathematical formulations that, all functions and indexed notations are extended to be defined on all possible values, i.e. are total: l_{ij} is for example defined for all indices and return a particular and non conflicting value, let us say $undef$, for all non relevant indices.

[8] This transformation can be automatized and formalized once for all using a higher order prover like Coq.

is conform to the initial automaton) and the next temporary invariant is satisfied. In fact if we decide that by convention that $inv_{n+1}(s,x)$ is defined as $false$, then $j \neq n+1 \wedge inv_j(s',x)$ in the second proof obligation can even be simplified into $inv_j(s',x)$. The two proofs obligations can be discharged using the techniques described in the previous section. The only specificity is due to the use of parametrized invariants and is illustrated below using a very simple example. Consider the following protocol with only one principal and only one action that stores in its only local variable *done* the set of all messages received so far, and accepts a new incoming message *msg* only if it is not already in set *done*: $msg \notin done$. We can easily express that the same message is never accepted twice: we just need to use a as the regular expression and a filtering function ff_x such that $ff_x(s,(l,m),s') = \perp \; if \; x \neq m$ and $ff_x(s,(l,m),s') = a \; if \; x = m$. The original automaton is a very simple two state automaton (i.e. $n = 2$). Before the particular message of interest x is first received we are in meta-state 1. The meta-state becomes 2 the first time a message *msg* whose value is x is received. We thus propose $inv_1((done,s_i),x) = true$[9] and $inv_2((done,s_i),x) = x \in done$. The first proof obligation will basically turn out proving after a few trivial steps that $\forall x, done, msg.x \in done \wedge x \neq msg \Rightarrow x \in (done \cup \{msg\})$. The second proof obligation will turn out proving that $\forall x, done, msg.x = msg \Rightarrow x \in done \cup \{msg\}$ and $\forall x, done, msg.x \in done \wedge x = msg \wedge \neg(msg \notin done) \Rightarrow false$. The first part corresponds to $i = 1$ and $j = 2$, whereas the second one corresponds to $i = 2$ and $j = 3$.

Let us now consider the revisited merchant property of the previous section under the first set of hypotheses. Meta-state 1 will refer to the state before firing of the first expected visible action (i.e. $PReq_M$), meta-state 2 to the state after firing of the first visible action and before firing of the second (i.e. $PReq_G$), meta-state 3 to the state after firing of the second and last expected visible action; meta-state 4 will stand for the absorbing state which is reached whenever a scenario does not conform to the security property. Now we propose the following *local* invariants:

$$
\begin{array}{l}
inv_1(s,(c,lc,lm,amt,hod,hpd)) = \neg(((c,M,(lc,lm),amt,hod,hpd))_{K_m^{-1}} known\_in\ s_i \\
inv_2(s,(c,lc,lm,amt,hod,hpd)) = lm \in s.mcht.used\_ids \\
inv_3(s,(c,lc,lm,amt,hod,hpd)) = lm \in s.mcht.used\_ids \wedge (c,M,(lc,lm)) \in s.gtw.auth\_ids
\end{array}
$$

where M and K_m^{-1} are constants (i.e. for the selected merchant), where *used_ids* is the local state variable used by the merchant to store the set of already used local identifiers (so as to avoid the reuse of a same local id more then once) and where *auth_ids* is the local state variable used by the gateway to store the set of already requested authorizations.

Intuitively, the first invariant corresponds to the situation where the first visible action $AuthReq_M$ (i.e. this action is characterized by $(c,lid\_c,lid\_m,amt,hod,hpd)$) has not been fired. Thus the corresponding message has not been generated and the intruder cannot compose the message by himself: $\neg(((c,M,(lc,lm),amt,hod,hpd))_{Km^{-1}} known\_in\ s_i$. The proof obligations for the preservation of this invariant can be discharged quite easily using the technique illustrated in section 7.1 for the proof of confidentiality. In doing so confidentiality properties need typically to be used as lemmas. Invariant inv_1 is also useful for proving that the visible action $AuthReq_G$ cannot be fired in the first place. The second invariant corresponds to the situation where the first visible action has been fired. This property is useful when proving that the visible action $AuthReq_M$ cannot be fired a second time. The third invariant is similar and guarantees also that $AuthReq_G$ cannot be fired a second time.

[9] $inv_1((done,s_i),x) = x \notin done$ would be a possible alternative provided that *done* is always initialized to $\emptyset$.

The proof of properties involving more visible actions or more trustable principals is very similar. It may additionally involve the use of constraints on the position of a particular principal with respect to its role: e.g. $pre\_PInitReq_m \leq s.mcht.at \leq pre\_AuthReq_M$ where $s.mcht.at$ is the abstract program counter of M as used in [4] and where $pre\_PInitReq_M$, $pre\_AuthReq_M$ are particular program points (i.e. labels) for the role of the merchant. This would be the case for the proof of the merchant property expressed in section 6. But the proof would be very similar. We would in particular use, in addition to the invariant properties used for the previous proof, the fact that before $AuthResp_G$ is fired the intruder does not know (in other words cannot compose) the signed $AuthResp$ message that M is expecting. Thus, if we had not included the merchant identifier M as part of the $Transaction$ data, then the merchant property verification would point out the following problematic scenario: a corrupted merchant M' intercepts the $AuthReq$ message sent by M and replaces $(Transaction)_{K_{m^{-1}}}$ by $(Transaction)_{K_{m'^{-1}}}$ and then sends the authorization request under his own identity. The payment gateways sends the authorization response back to M' and M' forwards it to M as if G had been accepting the transaction between C and M. In this situation M' wrongly believes that he has received an authorization from G and that he can deliver the object of the transaction to C who can contest the deal and claim that he did in fact perform the transaction with M'. For this scenario t there exists x such that $ff_x(t) = AuthReq_M AuthRes_M \notin L$ (i.e. the actions performed by G are abstracted away as they involve an incorrect value for $st'.gtw.merchant$). The problem is pointed out because non visible action $AuthRes_G$ violates the invariant, namely the part that says that the signed response from G cannot be known by the intruder.

The technique has shown to scale up very well. Furthermore, in practise the number of visible actions used for a particular expressions is limited: the number of different actions specified in the protocol (e.g. 12 for the protocol of section 4) is in practise a upper limit as properties can be expressed by focussing on one particular session at a time.

8 Conclusion

We have presented a new approach to the formal verification of electronic commerce protocols which extends the approach proposed originally in [4] for authentication protocols. The proposed approach has first been applied successfully on the C-SET protocol, and is currently being applied on the SET protocol, with complete formal proofs using Coq [7, 3]. The generality of the approach has been further validated by expressing security properties for other electronic commerce protocols (such as [2, 9, 14]). The main extensions are relative to the expression of security properties, and involves two new steps: (1) the use of a finite automaton and of a filtering function to describe each property, and (2) the verification under various weaker trustability hypotheses. The verification process was also revisited as a result of these extensions.

The proposed way of expressing properties amounts to consider the global behaviors of the system after some appropriate filtering of global behaviors and to express the constraint using a finite automaton. In practise it was found both intuitive and powerful. The combination of a filtering function and of an automaton can indeed be seen as the specification of a (symbolic) test performed by the global observer, and this fact was indeed exploited in discussions with protocol designers.

The approach has shown to scale up quite well in the case of large protocols such as C-SET or SET. The simplicity of proofs has been found to be a very important element to achieve useful

interaction between designers of the protocol and formal method experts[10]. Formal proofs can indeed be presented in an informal way by explaining for each automaton meta-state, why unexpected visible actions cannot be fired, and why the invariant is preserved by all other visible as well as non visible actions. The informal explanations are based on intruder knowledge as illustrated for confidentiality properties in section 7.1. Informal explanations can furthermore be used in optional informal reviews which can prove useful in demonstrating pertinence of formal verification to security experts. In the case of C-SET whose formal verification has been completed, the formal verification has allowed to identify significant improvements to the protocol in terms of security. The verification activity happens to reuse much from one verification to the other and from on protocol to the other. As a result many variants of a given protocol or sets of hypotheses can be analyzed with few additional effort.

9 Appendix

A1 $c\ known\_in\ s \wedge c'\ known\_in\ s \Rightarrow (c, c')\ known\_in\ s$

A2 $(c, c')\ known\_in\ s \Rightarrow c\ known\_in\ s \wedge c'\ known\_in\ s$

A3 $c\ known\_in\ s \wedge k\ known\_in\ s \Rightarrow c_k\ known\_in\ s$

A4 $c_k\ known\_in\ s \wedge k^{-1}\ known\_in\ s \Rightarrow c\ known\_in\ s$

A5 $s'\ known\_in\ s \wedge s''\ known\_in\ s \Rightarrow (s' \cup s'')\ known\_in\ s$

A6 $(s' \cup s'')\ known\_in\ s \Rightarrow s'\ known\_in\ s \wedge s''\ known\_in\ s$

A7 $s\ known\_in\ s$

A8 $\emptyset\ known\_in\ s$

B1 $(c, c')\ comp\_of\ s \Rightarrow c\ comp\_of\ s \wedge c'\ comp\_of\ s$

B2 $c_k\ comp\_of\ s \wedge k^{-1}\ comp\_of\ s \Rightarrow c\ comp\_of\ s$

B3 $s'\ comp\_of\ s \wedge s''\ comp\_of\ s \Rightarrow (s' \cup s'')\ comp\_of\ s$

B4 $(s' \cup s'')\ comp\_of\ s \Rightarrow s'\ comp\_of\ s \wedge s''\ comp\_of\ s$

B5 $s\ comp\_of\ s$

B6 $\emptyset\ comp\_of\ s$

C1 $\neg(c\ comp\_of\ s) \wedge \neg(k^{-1}\ comp\_of\ s) \wedge c \neq c'_k \Rightarrow \neg(c\ comp\_of\ s \cup c'_k)$

C2 $\neg(c\ comp\_of\ s \cup c') \wedge c \neq c'_k \Rightarrow \neg(c\ comp\_of\ s \cup c'_k)$

C3 $\neg(c\ comp\_of\ s) \wedge c \neq d \Rightarrow \neg(c\ comp\_of\ s \cup d)$

C4 $\neg(c\ comp\_of\ s \cup c_1 \cup c_2) \wedge c \neq (c_1, c_2) \Rightarrow \neg(c\ comp\_of\ s \cup (c_1, c_2))$

C5 $\neg(c\ comp\_of\ \emptyset)$

[10] This has been experimented in the case of C-SET.

C6 $\neg(s'\ comp\_of\ s) \vee \neg(s''\ comp\_of\ s) \Rightarrow \neg(s' \cup s''\ comp\_of\ s)$

C7 $c \notin s \wedge setofkeys(s) \Rightarrow \neg(c\ comp\_of\ s)$

C8 $setofkeys(s) \Rightarrow setofkeys(s \cup k)$

C9 $setofkeys(\emptyset)$

D1 $\neg(s'\ known\_in\ s) \Rightarrow \neg(s'\ comp\_of\ s)$

D2 $\neg(b\ comp\_of\ s) \Rightarrow \neg(b\ known\_in\ s)$

D3 $\neg(c_k\ comp\_of\ s) \wedge \neg(k\ comp\_of\ s) \Rightarrow \neg(c_k\ known\_in\ s)$

D4 $\neg(c_k\ comp\_of\ s) \wedge \neg(c\ known\_in\ s) \Rightarrow \neg(c_k\ known\_in\ s)$

D5 $\neg((c, c')\ comp\_of\ s) \wedge (\neg(c\ known\_in\ s) \vee \neg(c'\ known\_in\ s)) \Rightarrow \neg((c, c')\ known\_in\ s)$

D6 $\neg(s\ known\_in\ s'') \vee \neg(s'\ known\_in\ s'') \Rightarrow \neg(s \cup s'\ known\_in\ s'')$

References

[1] J.-P. Banâtre and D. Le Métayer. Gamma and the chemical reaction model: ten years after. In *Coordination programming: mechanisms, models and semantics*. World Scientific Publishing, IC Press, 1996.

[2] Mihir Bellare, Juan A. Garay, Amir Herzberg, Hugo Krawczyk, Michael Steiner, Gene Tsudik, and Michael Waidner. ikp – a family of secure electronic payment protocols. In *"Proc. First USENIX Workshop on Electronic Commerce"*, July 1995.

[3] D. Bolignano. Vérification formelle de protocoles cryptographiques à l'aide de Coq. In *Actes des journées GDR*, 1995.

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

[5] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8, 1990.

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

[7] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Murthy, C. Parent, C. Paulin-Mohring, and B. Werner. The coq proof assistant user guide. In *Rapport INRIA 154*, 1993.

[8] S. Eilenberg. *Automata, Languages, and Machines (Vol. A)*. Academic Press, 1974.

[9] Steve Glassman, Mark Manasse, Martin Abadi, Paul Gauthier, and Patrick Sobalvarro. The millicent protocol for inexpensive electronic commerce. In *"Fourth International World Wide Web Conference Dec 1995, Boston, MA, USA"*, December 1995.

[10] G.Leduc, O. Bonaventure, E. Koerner, L. Lonard, C. Pecheur, and D. Zanetti. Specification and verification of a ttp protocol for the conditional access to services. In *Proceedings of the 12th Workshop on the Application of Formal Methods to System Development (Univ Montreal)*, 1996.

[11] G. Lowe. An attack on the needham-schroeder public-key protocol. In *Information Processing Letters*, 1995.

[12] MasterCard and VISA. Secure electronic transactions specification (books 1, 2, 3). June 1996.

[13] C. Meadows. Applying formal methods to the analysis of a key management protocol. In *Journal of Computer Security*, 1992.

[14] Paul-André Pays. An intermediation and payment system technology. In *"Fifth International World Wide Web Conference May 6-10, 1996, Paris, France"*, May 1996.