Formal Verification of Cardholder Registration in SET

Giampaolo Bella[1], Fabio Massacci[2,3], Lawrence C. Paulson[1], and Piero Tramontano[3]

[1] Computer Laboratory, Univ. of Cambridge — Pembroke Street, Cambridge CB2 3QG, England
{gb221,lcp}@cl.cam.ac.uk

[2] Dip. di Ing. dell'Informazione, Univ. di Siena — via Roma 56, 53100 Siena, Italy
massacci@dii.unisi.it

[3] Dip. di Inform. e Sist., Univ. di Roma "La Sapienza" — via Salaria 113, 00198 Roma, Italy
{massacci,tramonta}@dis.uniroma1.it

Abstract. The first phase of the SET protocol, namely Cardholder Registration, has been modelled inductively. This phase is presented in outline and its formal model is described. A number of basic lemmas have been proved about the protocol using Isabelle/HOL, along with a theorem stating that a certification authority will certify a given key at most once. Many ambiguities, contradictions and omissions were noted while formalizing the protocol.

1 Introduction

The last ten years have seen the rapid development of formal methods for analyzing security protocols. At the same time, protocols have become much more complex. Early security protocols typically involved two or three agents and established a shared secret. Six pages were enough to describe the Needham-Schroeder protocol in 1978 [16]. But six hundred pages are not enough to describe the SET protocol [11,12,13]. Such a complex protocol is likely to contain errors, but verifying it formally is a huge challenge.

Meadows [14] notes a further problem: electronic commerce protocols are *systems* of protocols and their goals are difficult to express in terms of traditional protocol concepts such as authentication and secrecy. The SET protocol is split in many phases each of which can be seen as a protocol on its own and has quite different high-level goals.

Therefore, it is no surprise that we do not find many published works on its verification. After Kailar's [7] analysis of simple electronic commerce protocols, there have been attempts to model more realistic protocols such as Kerberos [3], TLS/SSL [19] and Cybercash coin-exchange [4,5]. However, to the best of our knowledge, the SET protocol has still been out of reach. Meadows and Syverson [15] have designed a language for describing SET specifications but have left the

F. Cuppens et al. (Eds.): ESORICS 2000, LNCS 1895, pp. 159–174, 2000.

actual analysis to future work. Kessler and Neumann [8] have designed a belief logic to analyse a single message of the payment phase of SET.

The present paper describes our work in the analysis of a complete phase of SET: Cardholder Registration.

2 The SET Protocol

The SET protocol [9,11,12,13] has been proposed and standardized by a consortium of credit card companies (VISA, Mastercard, American Express) and software corporations (Microsoft, Netscape, etc.). SET aims to protect sensitive cardholder information, to ensure payment integrity and to authenticate merchants and cardholders. It does not support non-repudiation.

The overall architecture of SET is based on a rooted hierarchy of *Certification Authorities* (CAs). The top level is a trusted *Root Certification Authority*, below which we find centralized CAs corresponding to credit card brands. One level down, there are geo-political subsidiaries and finally, two levels down, there are CAs (corresponding to banks) that actually interact with customers. The task of these CAs is to provide customers with digital certificates for signature and encryption. Customers must generate and safeguard their private keys.

Participants of the payment system are *Cardholders* (C) and *Merchants* (M). Their financial institutions are called *Issuers* and *Acquirers*, respectively; they act largely outside the protocol. *Payment Gateways* (PG) play the traditional role of clearing-houses: their task is to settle the payment requests made by merchants and cardholders when buying goods.

2.1 Overview of SET

The SET protocol consists of five phases. The first two phases are used by the agents participating in the protocol to register their keys and get the appropriate certificates. The remaining phases constitute the electronic transaction itself.

Cardholder Registration. This is the initial step for cardholders. The agent C sends to a certification authority CA the information on the credit card he wants to use. The CA replies with a registration form, which C completes and returns, together with the signing key that C wants to register. Then, CA checks that the credit card is valid (this step is outside the protocol) and releases the signature certificate for C who stores it for future use. All this information (such as credit card details) must be protected and this makes the protocol steps complicated. A couple of points are worth noting:

- C may register as many public keys as he wants to.
- C's identity is not stored in the certificate. It only contains the hash of the *primary account number* (PAN), loosely speaking the credit card number, and of a secret nonce (PANSecret). A merchant should not be able to verify a cardholder's identity from his certificate [11, pp. 7, 12 and 25].

– The certificates must assure the merchant (without his having to see the PAN) that there is a link between a cardholder, a card and a PAN that has been validated by the card issuer [11, pp. 8 and 25].

Merchant Registration. This phase performs the analogous function for merchants. In contrast with Cardholder Registration, the merchant M can not only register a public key for signature but also a public key for encryption. The process is shorter because there is no confidential information to be protected.

Purchase Request. We reach this phase if C has decided to buy something. C sends to M the order information and the payment instructions. M processes the order and starts the *Payment Authorization* phase by forwarding the payment instructions to the PG. This last step is needed because SET aims to keep the cardholder's PAN confidential; M cannot simply take this number, as done in telephone credit card transactions [17], and settle directly with the Issuer.

Payment Authorization. After receiving the payment instructions from the Merchant, the PG, in cooperation with Issuers and banks, checks that everything is fine. If so, it sends the payment authorization to M, who sends to C the confirmation and possibly the purchased goods. C acknowledges the result and M passes to the next stage.

Payment Capture. In this last phase, M sends to PG one or more payment requests and the corresponding capture tokens obtained during the previous steps. PG checks that everything is satisfactory and replies to M. The actual funds transfer from C to M is done outside the protocol.

To accomplish these tasks, SET uses numerous combinations of cryptographic functions. Even for the handling of certificates, SET makes many extensions to the PKCS standards by RSA-Security [20,21].

2.2 Cardholder Registration: A Closer Look

Our analysis concerns the Cardholder Registration phase of the protocol. Figure 1 [11, p. 36] provides a high-level view of this phase. We can distinguish three message pairs. The first pair starts the registration process; the second gives the cardholder an appropriate registration form; the last exchanges the completed registration form for the requested certificate. Let us describe them in a bit more detail.

Initiate Request. The cardholder C starts the protocol.

Initiate Response. When the CA receives the request, it transmits a signed answer and its certificates to the cardholder. The signature certificate is used to verify the signature affixed to the response. The encryption certificate provides the cardholder with the key necessary to protect the payment card account number (PAN) in the registration form request. The CA will identify the issuer of the card using the first six to eleven digits of the account number to select the appropriate registration form.

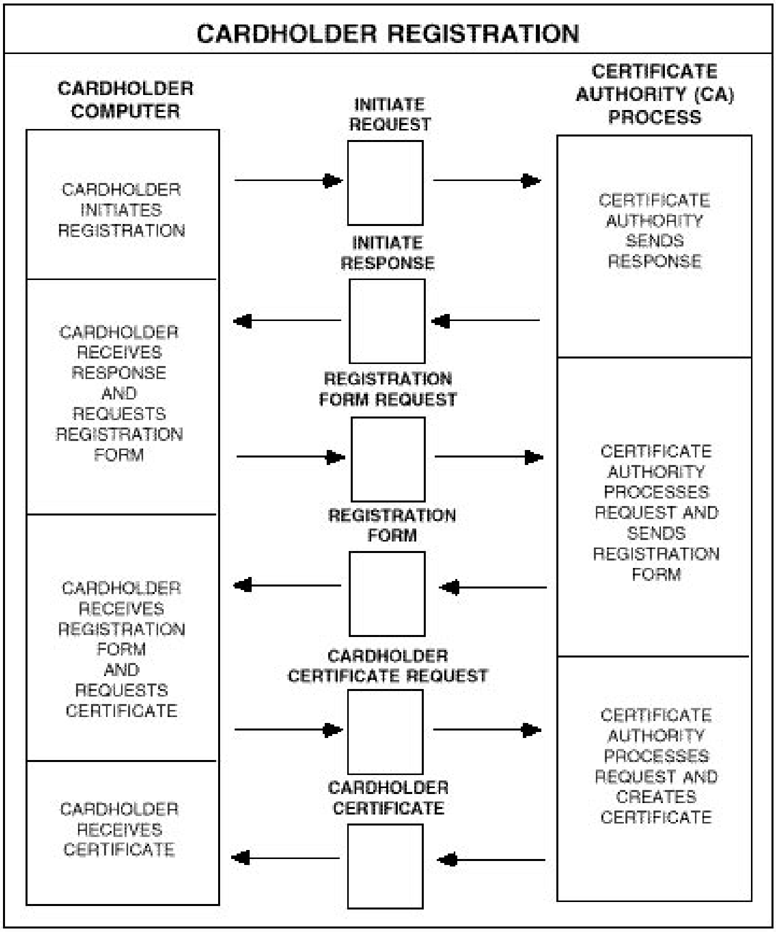

Fig. 1. Cardholder Registration in SET

$$
\begin{array}{lll}
1\,.\,C & \longrightarrow CA : & C,\ N_{C_1} \\
2\,.\,CA & \longrightarrow C : & \mathtt{Sign}_{CA}\,\{C,\ N_{C_1}\},\ \mathtt{CertE}_{RCA}\,\{CA\},\ \mathtt{CertS}_{RCA}\,\{CA\} \\
3\,.\,C & \longrightarrow CA : & \{C,\ N_{C_2},\ H(PAN)\}_{K_{C_1}},\ \mathtt{Encr}_{CA}\,\{K_{C_1},\ PAN,\ H(C,\ N_{C_2})\} \\
4\,.\,CA & \longrightarrow C : & \mathtt{Sign}_{CA}\,\{C,\ N_{C_2},\ N_{CA}\},\ \mathtt{CertE}_{RCA}\,\{CA\},\ \mathtt{CertS}_{RCA}\,\{CA\} \\
5\,.\,C & \longrightarrow CA : & \Big\{m,\ \{H(m,\ PAN,\ N_{SecC})\}_{privSK_C}\Big\}_{K_{C_3}}, \\
 & & \mathtt{Encr}_{CA}\,\{K_{C_3},\ PAN,\ N_{SecC}\} \\
 & & \quad \text{where } m = C,\ N_{C_3},\ K_{C_2},\ pubSK_C \\
6\,.\,CA & \longrightarrow C : & \big\{\mathtt{Sign}_{CA}\,\{C,\ N_{C_3},\ CA,\ N_{SecCA}\},\ \mathtt{CertS}_{CA}\,\{C\}, \\
 & & \ \mathtt{CertS}_{RCA}\,\{CA\}\big\}_{K_{C_2}}
\end{array}
$$

Fig. 2. High-level View of Cardholder Registration

Registration Form Request. C verifies the certificates of CA and the signature in the response. Then he sends a registration form request with his PAN. The request is encrypted with a random symmetric key that is encrypted along with the PAN in a digital envelope, sealed using the CA's public encryption key.

Registration Form. The CA unpacks the digital envelope, signs the appropriate registration form and returns it to C.

Cardholder Certificate Request. C verifies the certificate of CA and the signature on the received message. Now C fills in the registration form with the information the issuing bank deems necessary to identify him as a valid cardholder. C generates a signature key pair and a random number which will be used by CA to generate the certificate. Then, C creates a certificate request containing the registration form, the proposed public key and a random symmetric key used by CA to encrypt the response. This message is signed with C's private key. This signature yields no authentication—the corresponding public key is not yet certified—but it proves to CA that the requester knows the private key. The signed message is then encrypted with another fresh symmetric key; this key is encrypted along with the PAN and the random number, and the resulting message is sent to CA.

Cardholder Certificate. CA decrypts the request, checks C's signature and verifies the information on the registration form. In addition, CA should check that the key has not been registered by another cardholder; this obvious check is not mentioned in the specifications. Next, CA generates a random number and combines it with the one created by C to generate a secret value. Then CA generates the cardholder certificate by signing a message containing the public key and a hash of PAN and secret value. The certificate and a response message (with the CA half of the secret value) are encrypted with the symmetric key from the previous message and sent to C, together with CA's certificates.

Finally, before storing the certificate, C verifies it by comparing the hash contained in it with the correct value he can generate.

3 Making Sense of the Documents

The starting point of a formal analysis is defining an abstract model of the protocol. We eliminate technology dependent features and other inessential details. This must be done with care: keeping too many details results in an unmanageable model, while neglecting essential details allows unrealistic properties to be proved.

Usually, we can abstract away from particular cryptographic algorithms. The difference between SHA-1 and MD5 is inessential, since we merely expect hashing to be collision-free and non-invertible. We typically assume perfect encryption; in consequence, we can eliminate message components that are introduced only to circumvent the limitations of real-world cryptosystems.

The next obvious step is the elimination of all optional parts. But with SET, we found that this cannot be so easily done, as some options are not options at all. This is one of the major problems of the current SET specifications.

Here is an example. The task of the registration phases is to distribute certificates, which contain either encryption or signature keys. Both components are declared optional in the formal specification [12, p. 171], but omitting both 'options' would make the protocol vacuous. Only the informal text outside the definition [12, p. 170] says that at least one of them must be present.

Another example concerns symmetric keys. The specification of Cardholder Registration says that, at a certain stage, the cardholder C should send a symmetric key in order to get from the CA a message encrypted with that key. The field where the key is stored is tagged as optional [12, p. 171]. But the Programmer's Guide [13, p. 177] states that if the key field is missing then CA replies with an error message.

The SET designers do acknowledge this problem. The API reference guide to the SET reference implementation version 1.0 by Visa and MasterCard (available in the CD-ROM packaged with Loeb [9]) mentions this problem in the NOTES section at the end of the manual pages of the code:

> There is a difference between non-required and optional. Non-required fields may be omitted according to the SET protocol. Optional fields may be omitted according to ASN.1 encoding rules. In some messages, a field may be optional according to ASN.1, but still required by the SET protocol. In these cases, it is incumbent on the application to fill in these fields.

It is hard to believe that such a statement could be part of the specification of a security protocol. The manual pages do not distinguish the non-required and optional parts.

Another example is the use of certificates. According to the Business Description, CA always sends the certificates (or thumb-prints) for the signing keys. The reason, we think, is that Cardholder Registration is designed so that it can be interrupted and resumed later (we do not know whether this is intended) and that CA may want to offer C the possibility of always getting the most recent key. However, certificates for the encryption keys are sometimes missing, and this difference in managing the two different kinds of certificates is not explained. The handling of certificates can only be understood by making reference to the PKCS standards. So, to decide which message component is required for our analysis, we need a case-by-case analysis, which can only be done with a full comprehension of the protocol goals and structure.

Such problems of formalization are sadly typical of commercial protocols. Once a protocol reaches the stage of an RFC (Request for Comments), it is often specified in an elaborate but unsatisfactory manner. The meanings of the fields of a message are only given informally. Often it is hard to know precisely what the recipient of a message will do with it or when a particular message will be sent. It is often unclear what counts as a successful outcome. When it comes to syntax, we have the opposite problem: too much detail. The fields of each message are specified down to the last bit. Current verification technology cannot cope with detailed descriptions.

The SET protocol documentation suffers from the same problem. SET-Book 3 [12] provides a Formal Protocol Definition in ASN.1. It specifies each message field and yet it is not sufficient, as we have already noted. To resolve issues one must look elsewhere, such as in the Programmer's Guide [13] or in the Business Description [11], but sometimes they contradict each other. The SET designers state, 'In the event of discrepancy between this and any other description of the protocol, the ASN.1 in Part II takes precedence' [12, p. 1] but (as we have seen) this is sometimes impossible.

The Business Description [11] is most misleading — its figures especially. For instance, the description of the Payment Authorization phase suggests that each merchant will receive from the payment gateway a digital envelope containing in clear the primary account number of the customer [11, p. 66]. This contradicts the text which forbids such eventuality [11, p. 12]: 'SET ensures that in the cardholder's interactions with the merchant, the payment card account information remains confidential.' To complicate matters, in the fine print of the formal definition, a field in the certificate gives some merchants the privilege of receiving the cardholder account information in clear. The explicit requirement of confidentiality specified by the SET designers can then be overruled by an implementation which adheres to the specifications. The effect of this trap-door on the formal analysis of the payment phase remains to be seen.

We used the Programmer's Guide [13] as the ultimate reference. Other readers of the specification may resolve its ambiguities in other ways.

4 Cryptographic Functions for Cardholder Registration

This section presents the Cardholder Registration phase in a format similar to those used in security protocol papers. It also shows the relationship between the SET documentation and its Isabelle formalization. We introduce some notation and explain how we encode the complex combinations of hashing, strong and weak encryption that are used in SET and in the PKCS#7 standard.

Below, M and m denote messages, s denotes a sender, r denotes a receiver (s and r are SET entities). Here are the building blocks of our construction:

C, CA, RCA: entities involved in the CR-phase, respectively Cardholder, Certification Authority and Root Certification Authority

N_X, K_X: nonce N and symmetric key K generated by an agent X

$pubEK_X$, $privEK_X$: public and private encryption keys held by agent X

$pubSK_X$, $privSK_X$: public and private signature keys held by agent X

PAN_X: Primary Account Number of agent X

$\{X, Y, Z, \ldots\}$: a sequence (tuple) of zero or more data elements

$H(X)$: hash of tuple X

$\{X\}_K$: encryption of X with key K

The *DigestedData* $DD(M)$ is a simple construction. It is defined as the concatenation of a message and its digest, but in this case the plain message is absent; so we have only $H(M)$ for a message M. The *Linkage L(M,m)* is a shorthand for $\{M, DD(m)\}$. It links m to M as only someone possessing m or a trusted hash of m can verify the linkage. The intuition is that m contains some pieces of M. We express it as $\{M, H(m)\}$.

The *Signature only* $SO(s, M)$ is the signature of entity s on message M, omitting the plaintext of M, and corresponds to a PKCS#7 *SignedData* with the 'Content' field set to 'absent.' In our notation it is $\{H(M)\}_{privSK_s}$. The *Signed message S(s,M)* is a shorthand for $\{M, SO(s, M)\}$. It represents a full signature, the concatenation of a message M and its signed digest, and corresponds to a PKCS#7 *SignedData*. In our model it is expressed as $\{M, \{H(M)\}_{privSK_s}\}$ and is abbreviated as $\texttt{Sign}_s\,\{M\}$.

The *Asymmetric encryption* $E(r, M)$ uses a mixture of symmetric and asymmetric encryption. Given a message M, this is encrypted with a fresh symmetric key K and K itself is encrypted with the receiver r's public encryption key. It corresponds to the PKCS#7 *EnvelopedData* combining RSA encryption and Bellare-Rogaway Optimal Asymmetric Encryption Padding (OAEP). Since OAEP just aims at strengthening the cryptographic algorithms, we simply code this primitive as the pair $\{\{M\}_K, \{K\}_{pubEK_r}\}$ and abbreviates it as $\texttt{Encr}_r\,\{M\}$.

The *Extra encryption with integrity* $EXH(r, M, m)$ is a more complex form of digital envelope. Here m is a message usually containing payment card information and a nonce useful to foil dictionary attacks. Extra encryption with integrity is implemented in accordance with the following procedure:

- The hash of m is concatenated to M (as in a linkage);
- this is encrypted with a fresh symmetric key K obtaining a message m';
- a digital envelope containing K, m, and the hash of M is encrypted with the receiver r's public key;
- m' and the envelope are concatenated.

After being expanded and simplified, $EXH(r, M, m)$ can be expressed as

$$\{\{M, H(m)\}_K, \{K, m, H(M)\}_{pubEK_r}\}.$$

Hashing is used to verify integrity, but this primitive uses no signature and cannot authenticate the sender.

Simple Encapsulation with signature $Enc(s, r, M)$ models both digital signature and digital envelope and is an instance of PKCS#7 *SignedData* encapsulated in *EnvelopedData*. The message M is first signed with the private key of s and then encrypted with a fresh symmetric key K sent to r in a digital envelope. Thus *Enc(s,r,M)* is equivalent to *E(r,S(s,M))* and is expanded as $\{\{M, \{H(M)\}_{privSK_s}\}_K, \{K\}_{pubEK_r}\}$. This primitive authenticates the sender and provides confidentiality.

Simple Encapsulation with signature and provided key data $EncK(kd, s, M)$ is an instance of PKCS#7 *SignedData* encapsulated in *EncryptedData*. It models a message M first signed with the sender's private key and then encrypted with a symmetric key K. (The word 'provided' means that the key data must have been provided in advance by some other message of the protocol.) It is typically used if K is a symmetric key which has been previously sent by r. With this latter hypothesis, it guarantees both data confidentiality and sender's authentication. It boils down to $\{M, \{H(M)\}_{privSK_s}\}_K$.

The *Extra encapsulation with signature EncX(s,r,M,m)* is also a signed and sealed message, but it requires a more complex procedure:

- The DER[1] encoding of m is concatenated to M;
- the sender s signs a digest of the resulting message, yielding m';
- message m' is concatenated to M;
- the message resulting from the previous step is encrypted with a fresh symmetric key K, yielding a message m'';
- K and m are sealed using the public encryption key of r;
- finally m'' and this envelope are concatenated.

Since DER encoding is injective, we can replace *DER(m)* with m. Expanding the primitive results in $\{\{M, \{H(M, m)\}_{privSK_s}\}_K, \{K, m\}_{pubEK_r}\}$.

SET certificates make reference to X.509 Version 3 certificates [6], but includes the use of X.509 extensions (as defined in PKCS#6 [20]) and further SET-specific extensions. The point of including a set of attributes is to extend the certification process to other information about an entity, not just its public key. For instance, such information includes whether a merchant is allowed to get hold of his customer's PAN.

For simplicity, our certificate has only the attributes relevant to the formal analysis. The obvious attributes to take into consideration include the *Subject* (the certificate owner's identity) and the *SubjectPublicKeyInfo* (the certified public key). The issuing certificate authority is identified by the key signing the certificate. Unusually, the CA signs the entire plaintext (not just an hash of it). So, a certificate containing information I is implemented as $\{I, \{I\}_{privSK_{CA}}\}$. Since an hash is much smaller than the actual message, by signing the whole message the CA might have a better defence against brute force cryptanalysis.

Several attributes specify which entity is the certificate owner, the intended use of the certified key and whether this key may be used to issue other certificates (the default answer is no). There is also a flag F to distinguish between signature and encryption certificates. We omit certificate thumbprints, validity periods and revocation lists. Finally we obtain the following encoding of the certificates for cardholders and certification authorities:

$$\begin{aligned} certC &= \{\{H(PAN, PANSecret), pubSK_C, F\}, \\ &\qquad \{H(PAN, PANSecret), pubSK_C, F\}_{priSK_{CA}}\} \\ certCA &= \{CA, pubK_{CA}, F\}, \{CA, pubK_{CA}, F\}_{priSK_{RCA}}\} \end{aligned}$$

[1] DER (Distinguished Encoding Rules) is a set of rules for representing OSI's (Open Systems Interconnection) abstract objects as strings of ones and zeros.

Note that $PANSecret$ is obtained as the exclusive 'or' of two nonces N_{SecC}, provided by the Cardholder C, and N_{SecCA}, provided by the Certification Authority CA.

We need a few more abbreviations. By $\texttt{CertE}_{CA}\,\{X\}$ we denote the certificate of the public encryption key of X by the certification authority CA (the flag F is set to 0) and by $\texttt{CertS}_{CA}\,\{X\}$ we denote the certificate of the public signing key of X by the certification authority CA (the flag F is set to 1).

Finally, we can compose all above constructs, fill the gaps in the specifications that we have mentioned in Sect. 3. We obtain the high-level model of SET Cardholder Registration (Fig. 2 above).

5 Modelling Cardholder Registration in Isabelle

Modelling SET-CR requires new techniques. The generation of a pair of asymmetric keys (step 5) and the verification preceding the issue of certificates (step 6) have never been formalised before. To deal with the incompleteness of the official SET specifications, we decided to adopt the following policy: *the model should allow everything that the official specifications do not forbid*.

Isabelle includes generic theories for analysing security protocols. In order to handle SET, we had to modify and extend them, especially the model of key management. In this section we assume familiarity with the *inductive approach* to verifying protocols [18].

5.1 Agents

First of all, we need to model the SET certification chain. The model reduces the four levels of the actual hierarchy of trust to two, bypassing brand CAs and geo-political CAs: we denote by RCA the root certification authority, and introduce an unbounded number of first-level certification authorities CA's.

```
datatype agent = RCA | CA nat | Friend nat | Spy
```

5.2 Messages

The Primary Account Number (PAN) of the payment card is exchanged during SET sessions. We do not model a PAN as a nonce because it has a long lifetime, while fresh nonces are chosen for each run. We extend the Isabelle datatype for messages with a constructor Pan to allow PANs as message components.

```
datatype msg = Agent agent | Nonce nat | Number nat | Key key |
               Pan nat | Hash msg | MPair msg msg | Crypt key msg
```

A datatype definition introduces injective type constructors with disjoint ranges. Here this asserts that PANs cannot be confused with other numbers. Injectivity also implies that hashing is collision free and that an encrypted message corresponds to exactly one plaintext.

The model presupposes that PANs cannot be guessed. Therefore, the spy can synthesize them from a set H of message components, only if they are already available in the set, as stated by the theorem

$$\mathsf{Pan}\,P \in \mathsf{synth}\,H \Longrightarrow \mathsf{Pan}\,P \in H.$$

The function pan maps agents into naturals so that $\mathsf{Pan}\,(\mathsf{pan}A)$ formalises the message containing agent A's PAN.

5.3 Cryptographic Keys

Classical authentication protocols presuppose that each agent owns certain long-term keys and possibly acquires session keys. In contrast, certification protocols distribute long-term keys. We need to formalize this scenario and to understand new kinds of risks. In addition to distributing pairs of asymmetric long-term keys, SET-CR allows each agent to own several pairs, with the possibility of collision. The protocol uses different keys for different purposes: some for encryption, some for message signature, others for certificate signature. For simplicity, we identify the two kinds of signing keys. Thus, keys are associated to agents as follows.

- The root certification authority has a single pair of signature keys.
- A certification authority has a pair of encryption keys and a pair of signature keys. The SET specifications do not clearly state whether a CA may have more than one pair of each kind.
- A cardholder has no keys at the beginning but obtains them during a run. We assume that he can obtain more than one pair of keys regardless of which authority certifies them, as the SET specifications do not forbid this.
- The spy can obtain keys by running the protocol and also knows the keys of an unspecified set of certification authorities (see Sect. 5.4).

Note that the standard mapping of agents into keys by a single function is not acceptable: some keys that do not exist in reality would be associated to the cardholders. Hence, a bit more work is necessary for the Isabelle formalization.

Suitable rules state that all the keys existing before any protocol run are distinct. We call them *crucial* and put them in the set CrucialK.

5.4 Agents' Knowledge

Our model allows some certification authorities to collude with the spy in three different ways. An unspecified set badS of authorities has revealed their signature keys to the spy. Another set badE has revealed their encryption keys. A third set badN of authorities let the spy read any private notes, possibly containing crucial information, taken during the protocol sessions. The sets are unrelated and model many different scenarios. The Root CA is never compromised.

The existing formalisation of agents' knowledge [1,2] allows each agent to know the messages he alone sends or receives, while the spy knows all messages anybody sends or receives. This definition can easily be updated to capture the requirements stated above.

5.5 The Protocol Model

The signature of a message or of a certificate are two slightly different operations. Signing a message X by a key K returns the concatenation of X with the encryption by K of the hash of X. Signing a certificate X by a key K differs from the preceding operation by omitting the hashing. We model them as follows:

```
sign K X == {|X, Crypt K (Hash X)|}
signCert K X == {|X, Crypt K X|}
```

The protocol uses two different kinds of certificates: one issued by the cardholder, the other by the certification authorities. The cardholder issues a certificate by signing a message that contains his PAN and a nonce previously generated. The certification authorities issue certificates that bind an agent to a key. An extra parameter F distinguishes the encryption certificates ($F = 0$) from the signature certificates ($F = 1$).

```
certC PAN KA PS F SignK ==
   signCert SignK {|Hash{|Account PAN, Nonce PS|}, Key KA, Number F|}
certCA A KA F SignK == signCert SignK {|Agent A, Key KA, Number F|}
```

The formal protocol model is declared as a set of traces. A trace is a list of the events occurred during a particular history of the network.

```
consts set_cr :: event list set
```

The basic rules for defining set_cr provide the base of the induction, allow reception of the messages that have been sent and model the spy's illegal operations. They are omitted here, for they are common to all protocol models. The remaining rules formalise the protocol steps. Because of space limitations, we only quote the rules formalising the last two steps.

Rule SET_CR5. Two fresh nonces are needed: N_{C_3} establishes the freshness of the subsequent message, while N_{SecC} is used by the CA. The nonce N_{CA}, sent by the certification authority to verify the freshness of the subsequent message, is not sent back by the cardholder (it is optional in the formal definition), so its utility looks questionable. Two fresh symmetric keys are needed: K_{C_3} is used to create a digital envelope, K_{C_2} to encrypt the reply.

The cardholder generates a key *cardSK* that he wants certified as a public signature key. The only condition imposed on *cardSK* is that it should differ from the crucial keys. The model allows the cardholder to propose keys that have already been certified, possibly for a different cardholder or by a different authority.

```
[| evs5 ∈ set_cr; C ≠ (CA i);
   Nonce NC3 ∉ used evs5;  Nonce NSecC ∉ used evs5;  NC3≠NSecC;
   Key KC2 ∉ used evs5; isSymKey(KC2);
   Key KC3 ∉ used evs5; isSymKey(KC3); KC2≠KC3;
   ~isSymKey(cardSK);  cardSK ∉ crucialK;  invKey cardSK ∉ crucialK;
```

```
   Gets C {|sign (invKey SK) {|Agent C, Nonce NC2, Nonce NCA|},
            certCA (CA i) EK 0 priSK_RCA,
            certCA (CA i) SK 1 priSK_RCA |}  ∈ set evs5;
   Says C (CA i) {|Crypt KC1 {|Agent C, Nonce NC2,
                               Hash (Account (pan C)) |},
                  Crypt EK {|Key KC1, Account (pan C),
                               Hash {|Agent C, Nonce NC2|} |} |}
      ∈ set evs5 |]
==> Says C (CA i)
         {|Crypt KC3
              {|Agent C, Nonce NC3, Key KC2, Key cardSK,
                Crypt (invKey cardSK)
                  (Hash {|Agent C, Nonce NC3, Key KC2, Key cardSK,
                          Account (pan C), Nonce NSecC|} ) |},
            Crypt EK {|Key KC3, Account (pan C), Nonce NSecC|} |}
       # evs5 ∈ set_cr
```

Rule SET_CR6. When the certification authority receives a certificate request, he checks that the proposed public key has been used to sign the hash. Then he generates a certificate, which includes the fresh nonce N_{SecCA}. The condition

```
∀Y C'. Key cardSK ∈ parts{Y} → Says (CA i) C' Y ∉ set evs6
```

requires that the proposed key *cardSK* has never appeared in a message sent by the certification authority. So the cardholder can be assured that the same key has not been certified to some other agent. This condition is not explicitly required by the SET specifications, but the *Programmer's Guide* [13, p. 33] states the general principle that a certificate should 'bind a public key to a uniquely identified entity.' Our model does allow a private key of one agent to be certified as a public key of another one.

```
[| evs6 ∈ set_cr;  (CA i) ≠ C;
   Nonce NSecCA ∉ used evs6;
   ∀Y C'. Key cardSK ∈ parts{Y} → Says (CA i) C' Y ∉ set evs6;
   Gets (CA i)
        {|Crypt KC3
             {|Agent C, Nonce NC3, Key KC2, Key cardSK,
               Crypt (invKey cardSK)
                 (Hash {|Agent C, Nonce NC3, Key KC2, Key cardSK,
                         Account (pan C), Nonce NSecC|} ) |},
          Crypt (pubEK i) {|Key KC3, Account(pan C), Nonce NSecC|}|}
     ∈ set evs6 |]
==> Says (CA i) C
         (Crypt KC2
           {|sign (priSK i)
                 {|Agent C, Nonce NC3, Agent (CA i), Nonce NSecCA|},
             certC (pan C) cardSK (XOR(NSecC,NSecCA)) 1 (priSK i),
             certCA (CA i) (pubSK i) 1 priSK_RCA|})
```

6 Mechanically Proved Properties

At the start, we had to prove all over again the properties of the modified theory of messages, which form the basis of Paulson's inductive method. This totals around 200 theorems. Isabelle's automation made this task easy, allowing us to concentrate on the novel aspects of the theories under study.

Security properties from the protocol verification literature [10,18,22], such as authentication and agreement, are often impossible to prove for the whole protocol run. A cause is the optional use of nonces, which are sent but need not be returned. So, it impossible to establish linkages of messages throughout the protocol.

For the analysis of Cardholder Registration, we have proved around 30 technical lemmas focussed towards the proofs of two major properties: a key is only certified as belonging to one agent (at least by the same CA), and the PAN remains confidential.

For the first property, we have been successful. Formally, if each authority should certify a given key only once, this means that if there exist two messages from a CA that certify the same key, then those messages are identical. This is stated by the following theorem.

Theorem 1.

```
[| evs ∈ set_cr
   Says (CA i) C
        (Crypt KC2
         {|sign (priSK i) {|Agent C, Nonce NC3, Agent(CA i), Nonce Y|},
           certC (pan C) cardSK X 1 (priSK i),
           certCA (CA i) (pubSK i) 1 priSK_RCA|}) ∈ set evs;
   Says (CA i) C'
        (Crypt KC2'
         {|sign (priSK i) {|Agent C, Nonce NC3', Agent(CA i), Nonce Y'|},
           certC (pan C) cardSK X' 1 (priSK i),
           certCA (CA i) (pubSK i) 1 priSK_RCA|}) ∈ set evs
|] ⟹ C=C' & KC2=KC2' & NC3=NC3' & X=X' & Y=Y'
```

In the proof, after induction and simplification, the subgoal arising from the modelling of the last step of the protocol requires further consideration. In this step, an authority $\mathsf{CA}\, i'$ certifies a key *cardSK'*. If either $i \neq i'$ or $\mathit{cardSK}' \neq \mathit{cardSK}$, then the inductive formula concludes the proof. Otherwise, the authority $\mathsf{CA}\, i$ has certified *cardSK* twice, which our model forbids: contradiction.

While stressing that each authority certifies a specific key for a single cardholder, the theorem does not prevent different authorities to certify the same key for the same cardholder. Such a scenario is a subject for future analysis.

We have not managed to prove confidentiality of the PAN. In particular, we have not been able to eliminate a subgoal stating that no collection of keys can help the spy in getting the CAs' private keys. Again, the difference with other protocols is that here asymmetric key pairs are generated on the fly. The novelty of the problem seems to require novel proof techniques.

7 Conclusions

The introduction described other work concerning the SET protocol. But no previous work completely formalizes a phase of this protocol. One reason may be that the official specifications of SET [11,12,13] describe the composition of messages in minute detail, while failing to give them a satisfactory semantics.

Our inductive specification provides an operational semantics for the Cardholder Registration phase. We have unveiled some potentially dangerous omissions. We have proved that if a trusted CA keeps track of the registered keys, the protocol is robust enough to guarantee that two different agents will never get the same key certified by the same certification authority. However, different agents may collude and register the same key with different CAs. We have not investigated the consequences of this scenario, which might limit the accountability of the payment phase.

Our future work will cover the remaining phases of the protocol. Having digested the specifications, the greatest task is behind us. We expect to be able to derive further properties of SET with an acceptable amount of effort.

Acknowledgements. F. Massacci acknowledges the support of CNR and MURST grants. Part of this work has been done while P. Tramontano was visiting the Computer Laboratory in Cambridge under a visiting scholarship for master students by the University of Roma I "La Sapienza". L. Paulson was funded by the EPSRC grant GR/K57381 *Mechanizing Temporal Reasoning*.

References

1. G. Bella. Message reception in the inductive approach. Research Report 460, University of Cambridge — Computer Laboratory, 1999.
2. G. Bella. Modelling agents' knowledge inductively. In *Proceedings of the 7th International Workshop on Security Protocols*, volume 1796 of *Lecture Notes in Comp. Sci.* Springer-Verlag, 1999.
3. G. Bella and L. C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In *Proc. of the 5th Eur. Sym. on Res. in Comp. Sec.*, volume 1485 of *Lecture Notes in Comp. Sci.*, pages 361–375. Springer-Verlag, 1998.
4. S. Brackin. Automatic formal analyses of two large commercial protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, September 1997. Available on the web at `http://www.arca.com/proj_papers/brackin/dimacs.pdf`.
5. S. Brackin. Automatically detecting authentication limitations in commercial security protocols. In *Proceedings of the 22nd National Conference on Information Systems Security*, October 1999. Available on the web at `http://www.arca.com/proj_papers/brackin/Nissc99.pdf`.
6. CCITT. *Recommendation X.509: The Directory - Authentication Framework*, 1988. Available electronically at `http://www.itu.int/itudoc/itu-t/rec/x/x500up/x509.html`.
7. R. Kailar. Reasoning about accountability in protocols for electronic commerce. In *Proc. of the 14th IEEE Sym. on Sec. and Privacy*, pages 236–250. IEEE Comp. Society Press, 1995.

8. V. Kessler and H. Neumann. A sound logic for analysing electronic commerce protocols. In J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann, editors, *Proc. of the 5th Eur. Sym. on Res. in Comp. Sec.*, volume 1485 of *Lecture Notes in Comp. Sci.* Springer-Verlag, 1998.
9. L. Loeb. *Secure Electronic Transactions: Introduction and Technical Reference.* Computer Science. Artech House, 1998. Enclosed a CD-ROM with SET Reference Implementation Version 1.0.
10. G. Lowe. A hierarchy of authentication specifications. In *Proc. of the 10th IEEE Comp. Sec. Found. Workshop*, pages 31–43. IEEE Comp. Society Press, 1997.
11. Mastercard & VISA. *SET Secure Electronic Transaction Specification: Business Description*, May 1997. Available electronically at `http://www.setco.org/set_specifications.html`.
12. Mastercard & VISA. *SET Secure Electronic Transaction Specification: Formal Protocol Definition*, May 1997. Available electronically at `http://www.setco.org/set_specifications.html`.
13. Mastercard & VISA. *SET Secure Electronic Transaction Specification: Programmer's Guide*, May 1997. Available electronically at `http://www.setco.org/set_specifications.html`.
14. C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proceedings of DISCEX 2000*, pages 237–250. IEEE Comp. Society Press, 2000.
15. C. Meadows and P. Syverson. A formal specification of requirements for payment transactions in the SET protocol. In R. Hirschfeld, editor, *Proceedings of Financial Cryptography 98*, volume 1465 of *Lecture Notes in Comp. Sci.* Springer-Verlag, 1998.
16. R. M. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Comm. of the ACM*, 21(12):993–999, 1978.
17. D. O'Mahony, M. Peirce, and H. Tewari. *Electronic payment systems.* The Artech House computer science library. Artech House, 1997.
18. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. of Comp. Sec.*, 6:85–128, 1998.
19. L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Trans. on Inform. and Sys. Sec.*, 2(3):332–351, 1999.
20. RSA Laboratories. *PKCS-6: Extended-Certificate Syntax Standard*, 1993. Available electronically at `http://www.rsasecurity.com/rsalabs/pkcs`.
21. RSA Laboratories. *PKCS-7: Cryptographic Message Syntax Standard*, 1993. Available electronically at `http://www.rsasecurity.com/rsalabs/pkcs`.
22. S. Schneider. Verifying authentication protocols in CSP. *IEEE Trans. on Software Engineering*, 24(9):741–758, 1998.