*Technical Report*

UCAM-CL-TR-531
ISSN 1476-2986

Number 531

**UNIVERSITY OF CAMBRIDGE**

**Computer Laboratory**

# Verifying the SET registration protocols

Giampaolo Bella, Fabio Massacci,
Lawrence C. Paulson

March 2002

JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

*http://www.cl.cam.ac.uk/*

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

*http://www.cl.cam.ac.uk/TechReports/*

Series editor: Markus Kuhn

ISSN 1476-2986

**Abstract**

SET (Secure Electronic Transaction) is an immense e-commerce protocol designed to improve the security of credit card purchases.

In this paper we focus on the initial bootstrapping phases of SET, whose objective is the registration of customers and merchants with a SET certification authority. The aim of registration is twofold: getting the approval of the cardholder's or merchant's bank, and replacing traditional credit card numbers with electronic credentials that customers can present to the merchant, so that their privacy is protected.

These registration sub-protocols present a number of challenges to current formal verification methods. First, they do not assume that each agent knows the public keys of the other agents. Key distribution is one of the protocols' tasks. Second, SET uses complex encryption primitives (digital envelopes) which introduce dependency chains: the loss of one secret key can lead to potentially unlimited losses.

Building upon our previous work, we have been able to model and formally verify SET's registration with the inductive method in Isabelle/HOL solving its challenges with very general techniques.

# Contents

# 1 Introduction

Cryptographic protocols allow people to communicate securely across an open network, even in the presence of hostile or compromised agents. Such protocols are hard to design and numerous researchers have developed ways of finding errors automatically [7, 10] or proving protocols correct [5, 11]. (Many additional references could be given.) Here we report our verification of the registration protocols of SET, a giant protocol for electronic commerce, proposed by Visa and MasterCard as an industry standard [8].

The idea behind the registration protocols of SET is that only registered customers and merchants can engage in transactions. A registered cardholder has been cleared by a bank and has a digital certificate to prove it. Subsequently, he can show his certificates rather than his credit card number to an equally certified merchant. The merchant will rest assured that there is a credit card behind the private key signing a bill, and the customer will be sure that incompetent or dishonest merchants will not publish his credit card details on the internet.

At a this level of abstraction, the registration protocols look trivial: they just distribute public key certificates. However, past experience shows that simplifying a protocol's encryption mechanisms can hide major errors [13]. SET presents two major challenges to formal methods:

1. it involves several levels of encryption, using many combinations of symmetric cryptography, asymmetric cryptography and hashing;

2. it does not assume that each agent has his own private key (so that the only problem is the distribution of the public keys), but allows customers and merchants to invent asymmetric keys at will.

The first challenge comes from SET's use of RSA *digital envelopes*. One part of a digital envelope is the main body of the message, encrypted using a fresh symmetric key. The other part contains that key and is encrypted with the recipient's public encryption key. The two parts may have some common data, possibly hashed, in order to confirm that they are tied together. This combination of symmetric and asymmetric encryption is more efficient and secure than using either form of encryption alone. However, it makes a protocol harder to analyze. For instance, assuming that long-term asymmetric keys are secure, as all verification techniques do, will not guarantee us that the data in a digital envelope is safe.

Furthermore, digital envelopes can be used to send keys, which are used to package new envelopes, ad infinitum. A complicated case is in the last message exchange of cardholder registration, where a digital envelope conveys a symmetric key that the recipient uses to encrypt the reply. These creates dependency chains such that the loss of a secret key can lead to a cascade of losses. Nothing like this can be found in the customary benchmark

for protocol verification methods, the Clark-Jacobs library [4]. Therefore, many protocol verification formalisms [6] assume that to prove secrecy it is enough to show that the long-term keys encrypting the short-term keys are safe. Past protocols were too simple to reveal this point.

The second challenging aspect of the SET protocols is the possibility for cardholders and merchants to invent private and public key pairs at will for their electronic credentials. The difference with session key and key agreement protocols is minimal: asymmetric keys join nonces and sessions keys among the objects that can be invented during a protocol run. We do not make the usual assumption that each agent knows the other agents' public keys.

However, all current verification approaches functionally associate asymmetric and other long-term keys to agents. This modelling choice substantially eliminates asymmetric keys from the hard part of the modelling: namely reasoning about what an agent can encrypt and decrypt and the introduction of fresh values. Once asymmetric keys are fixed from the outset, and at most are unknown to the intruder, reasoning about asymmetric encryption is substantially reduced to an equality check between the agent holding the message and the agent associated to the key. In model checking, these modelling choices have further advantages: the introduction of fresh values can be limited to nonces and symmetric keys, thus cutting the explosion of the state space.

We have not only verified these protocols but found what appears to be a general method for treating such protocol mechanisms.

This paper presents an introduction to the SET registration protocols (§2). Next, the formal model of the registration protocols is presented (§3). The main secrecy proofs for cardholder registration and presented (§4). A final section discusses related work and presents conclusions (§5).

## 2 The SET Registration Protocols

People normally pay for goods purchased over the Internet by giving the merchant their credit card details. To prevent eavesdroppers from stealing the card number, the message is encrypted using the SSL protocol. This arrangement requires the customer and merchant to trust each other. That requirement is undesirable even in face-to-face transactions, and across the Internet it admits unacceptable risks.

- The cardholder is protected from eavesdroppers but not from the merchant himself. Some merchants are dishonest and some merchants are incompetent at protecting sensitive information.

- The merchant has no protection against dishonest customers who supply an invalid credit card number or who claim a refund from their

bank without cause. Contrary to popular belief, it is the merchant who has the most to lose from fraud. Legislation in most countries protects the consumer.

As stated in the Introduction, SET aims to reduce fraud by introducing a preliminary registration process. Cardholders and merchants must register with a *certificate authority* (CA) before they can engage in transactions. The cardholder thereby obtains electronic credentials to prove that he is trustworthy. The merchant similarly registers and obtains credentials. Later, when the customer wants to make purchases, he and the merchant exchange their credentials. If both parties are satisfied then they can proceed. SET includes separate subprotocols (called *transactions*) for cardholder and merchant registration.

We focus first on *cardholder registration* (Figure 1), which is the more complicated of the two. The cardholder proves his identity by giving the CA personal information previously shared with his issuing bank. He chooses a private key, which he will use later to sign orders for goods, and registers the corresponding public key, which merchants can use to verify his signature. The cardholder receives a certificate, signed by the CA, that associates the public key to his identity. Notice that the protocol does not assume that the key submitted by the cardholder is unique, nor that is fresh. The usual assumption that the cardholder registers *his* key should be replaced by the working hypothesis that the cardholder registers *some* key.

The protocol is complicated because it has many objectives. It must certify a signature key and associate it with the credit card number, while keeping the latter secret. In this way the Merchant can be assured that an order signed by a key certified by Visa's CA is matched by corresponding credit card issued by Visa, even if he does not see the credit card number.

Cardholder registration consists of six messages. We have abbreviated some of the SET terminology, for instance Chall_C has become NC1. Note that SET requires each CA to have separate key pairs for signature and encryption.

**Initiate Request.** The Cardholder sends his name to the CA, with a freshness challenge (NC1).

$$1.\ C \rightarrow CA : C, \mathrm{NC1}$$

**Initiate Response.** The CA responds to the challenge and returns its public key certificates, which are signed by the Root Certificate Authority. The cardholder needs the CA's public keys for the various SET protocols.

$$2.\ CA \rightarrow C : \mathsf{Sign}_{CA}(C,\ \mathrm{NC1}),\ \mathsf{Cert}_{\mathsf{RCA}}(\mathsf{pubEK}\ CA),\ \mathsf{Cert}_{\mathsf{RCA}}(\mathsf{pubSK}\ CA)$$

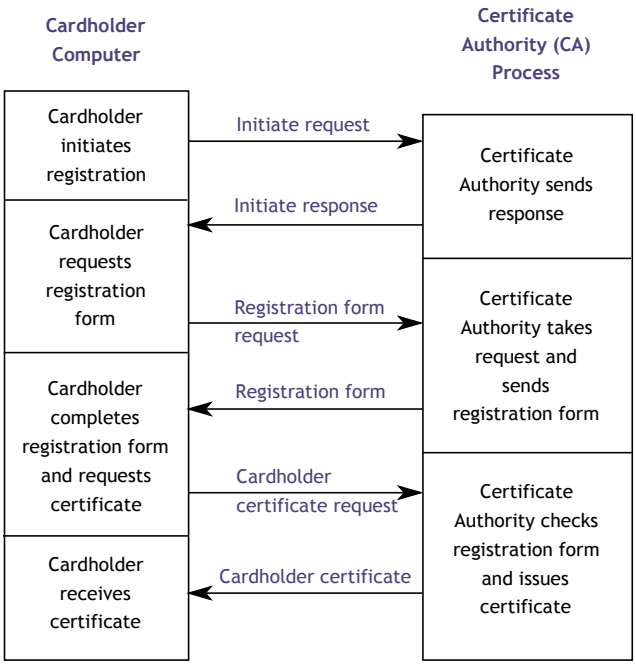


Figure 1: Cardholder Registration

**Registration Form Request.** The cardholder requests a registration form. In this message, he submits his credit card number to the CA. SET calls this the **PAN**, for Principal Account Number. This message is our first example of a digital envelope: some data is encrypted using the key KC1, which is itself encrypted using the CA's public key.

$$3.\ C \to CA : \mathsf{Crypt}_{\mathrm{KC1}}(C,\ \mathrm{NC2},\ \mathsf{Hash}\,\mathrm{PAN}),\\ \mathsf{Crypt}_{\mathsf{pubEK}\ CA}(\mathrm{KC1},\ \mathrm{PAN},\ \mathsf{Hash}(C,\ \mathrm{NC2}))$$

**Registration Form.** The CA uses the credit card number to determine the cardholder's issuing bank and returns an appropriate registration form. SET does not specify the details of such forms, which we therefore omit from the formalization. The CA again sends its public key certificates.

$$4.\ CA \to C : \mathsf{Sign}_{CA}(C,\ \mathrm{NC2},\ \mathrm{NCA}),\ \mathsf{Cert}_{\mathsf{RCA}}(\mathsf{pubEK}\ CA),\\ \mathsf{Cert}_{\mathsf{RCA}}(\mathsf{pubSK}\ CA)$$

**Cardholder Certificate Request.** The cardholder chooses an asymmetric signature key pair. He gives the CA the public key, $\mathsf{pubSK}\,C$, and the completed registration form. He also encloses CardSecret, a random number that must be kept secure permanently. This message is another digital envelope, using the key KC3. Another key, KC2, is sent to the CA to use

for encrypting the response. The proliferation of keys complicates reasoning about this protocol.

$$5.\ C \to CA : \mathsf{Crypt}_{\mathrm{KC3}}(m,\ \mathsf{Crypt}_{\mathsf{priSK}\,C}(\mathsf{Hash}(m,\ \mathrm{PAN},\ \mathrm{CardSecret}))),$$
$$\mathsf{Crypt}_{\mathsf{pubEK}\,CA}(\mathrm{KC3},\ \mathrm{PAN},\ \mathrm{CardSecret})$$
$$\text{where } m = C,\ \mathrm{NC3},\ \mathrm{KC2},\ \mathsf{pubSK}\,C$$

**Cardholder Certificate.** The bank checks the various details, and if satisfied, authorises the CA to complete the registration. The CA signs a certificate that includes the cardholder's public signature key and the cryptographic hash of PANSecret: a secret number known to the cardholder. PANSecret is the exclusive-OR of the CardSecret (chosen by the cardholder) and NonceCCA (chosen by the CA). The cardholder will use the PANSecret to prove his identity when making purchases.

$$6.\ CA \to C : \mathsf{Crypt}_{\mathrm{KC2}}(\mathsf{Sign}_{CA}\,C,\ NC3,\ CA,\ \mathrm{NonceCCA},$$
$$\mathsf{Cert}_{CA}(\mathsf{pubSK}\,C),\ \mathsf{Cert}_{\mathsf{RCA}}(\mathsf{pubSK}\,CA))$$

The *merchant registration* protocol (Figure 2) is simpler. No credit card number is involved. The CA determines the appropriate registration form merely on the basis of the merchant's name.[1]This eliminates one message exchange: there is no registration form request message. The merchant chooses two private keys, for signature and encryption, and registers the corresponding public keys (one at a time). The main goal of this protocol is to provide the merchant with certificates, signed by the CA, that associate the public keys to the merchant's identity. Here are the four messages in more detail.

**Initiate Request.** The merchant sends his name to the CA, with a freshness challenge (NM1).

$$1.\ M \to CA : M, \mathrm{NM1}$$

**Registration Form.** The CA determines the merchant's bank (known as the acquirer) and returns an appropriate registration form, along with its public key certificates.

$$2.\ CA \to M : \mathsf{Sign}_{CA}(M,\ \mathrm{NM2},\ \mathrm{NCA}),\ \mathsf{Cert}_{\mathsf{RCA}}(\mathsf{pubEK}\,CA),$$
$$\mathsf{Cert}_{\mathsf{RCA}}(\mathsf{pubSK}\,CA)$$

[1]Observe that there are many more customers than merchants and that becoming an accredited merchant costs money and time, so that a name search is feasible. As for privacy, the name of a merchant and his accepted credit cards are public and indeed the more public the better for the merchant himself.

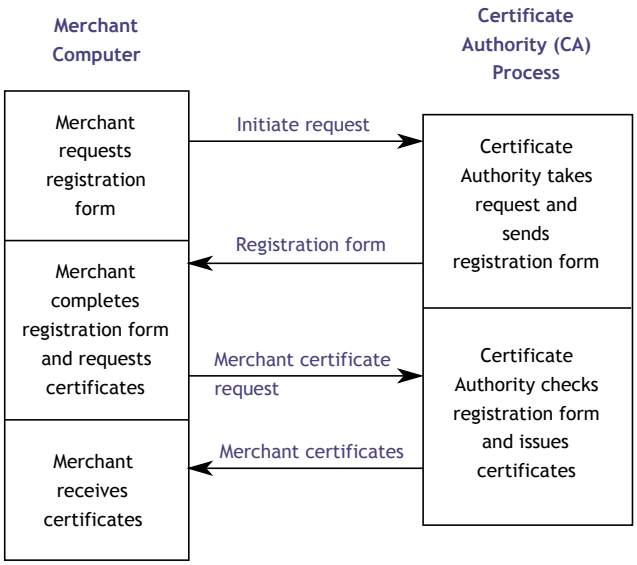


Figure 2: Merchant Registration

**Merchant Certificate Request.** The merchant chooses two asymmetric public/private key pairs, one for signature, the other for encryption. He submits the two public keys, $\mathsf{pubSK}\,M$ and $\mathsf{pubEK}\,M$, along with the completed registration form to the CA, who forwards it to the bank. This message is yet another digital envelope, using the session key KM1.

$$3.\ M \rightarrow CA : \mathsf{Crypt}_{\mathrm{KM1}}(\mathsf{Sign}_{\mathsf{priSK}\,M}(M, \mathrm{NM2},\ \mathsf{pubSK}\,M,\ \mathsf{pubEK}\,M)),\\ \mathsf{Crypt}_{\mathsf{pubEK}\,CA}\,\mathrm{KM1}$$

**Merchant Certificates.** The bank checks the various details, and if satisfied, authorises the CA to issue certificates. The CA signs two certificates, one including the merchant's public signature key and the merchant's identity, the other including the merchant's public encryption key and the merchant's identity. The CA wraps up the two certificates in a single message using no hashing, and sends it to the merchant. When the merchant receives the certificates, he is ready to sell goods over the Internet.

$$4.\ CA \rightarrow M : \mathsf{Crypt}_{\mathrm{KC2}}(\mathsf{Sign}_{CA}(M,\ NM3,\ CA,\ \mathrm{NonceCCA}),\\ \mathsf{Cert}_{CA}(\mathsf{pubSK}\,M),\ \mathsf{Cert}_{\mathsf{RCA}}(\mathsf{pubSK}\,CA))$$

What is the point of verifying SET's registration protocols? The subsequent purchase protocols perform the actual E-commerce, and protocol verifiers often assume that participants already possess all needed credentials. However, the registration protocols are difficult, particularly when it comes to proving that cardholder registration actually keeps the PANSecret

secret, an explicit goal of SET [8]. The digital envelopes introduce many keys and nonces, with non-trivial dependency chains.

# 3 Modelling the Registration Protocols

Our models of the registration protocols are largely the work of Piero Tramontano, who devoted many hours to help us decipher and interpret 1000 pages of SET documentation [2]. Our aim was to capture the essential protocol mechanisms while omitting optional parts and needless complications.

We use the inductive method of protocol verification, which has been described elsewhere [11]. This operational semantics assumes a population of honest agents obeying the protocol and a dishonest agent (the Spy) who can steal messages intended for other agents, decrypt them using any keys at his disposal and send new messages as he pleases. Some of the honest agents are *compromised*: the Spy has full access to their secrets. A protocol is modelled by the set of all possible traces of events that it can generate. Events are of three forms:

- `Says A B X` means $A$ sends message $X$ to $B$.
- `Gets A X` means $A$ receives message $X$.
- `Notes A X` means $A$ stores $X$ in its internal state.

Notice that for the says action we have no guarantee that $A$'s message will ever reach $B$.

We have flattened SET's hierarchy of certificate authorities [8]. The Root Certificate Authority is responsible for certifying all the other CAs. Our model includes compromised CAs, though we assume that the root is uncompromised. The compromised CAs complicate the proofs — large numbers of session keys and other secrets fall into the hands of the Spy. But even if we assumed that all CAs were honest, a realistic model would have to include the possibility of secrets becoming compromised.

Here is a brief summary of the notation:

- `set_cr` is the set of traces allowed by cardholder registration;
- `set_mr` is the set of traces allowed by merchant registration;
- `used` is the set of items appearing in the trace, to express freshness;
- `symkeys` is the set of symmetric keys;
- `Nonce`, `Pan`, `Key`, `Agent`, `Crypt` and `Hash` are obvious message constructors;
- $\{\!|X_1, \ldots, X_n|\!\}$ is an $n$-component message;

- *sign* is the message constructor for signatures, defined by

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

  where *K* is a private signing key.

- *certC* is the message constructor for a cardholder's public-key certificates, which includes his *PAN* and the PanSecret, *PS*. It is defined by

  ```
  signCert K X             == {|X, Crypt K X|}
  certC PAN Ka PS T signK ==
        signCert signK {|Hash {|Nonce PS, Pan PAN|}, Key Ka, T|}
  ```

- *cert* is the message constructor for public-key certificates of CAs and merchants:

  ```
  cert A Ka T signK == signCert signK {|Agent A, Key Ka, T|}
  ```

### 3.1 Modelling Cardholder Registration in Isabelle/HOL

A taster for the formalization is shown in Figure 3. It contains a fragment of our inductive model for cardholder registration, under the simplifying assumptions that the *Cardholder* has a unique public/private key pair functionally associated to his name.

The figure presents the full rules for messages 5 and 6, which are *SET_CR5* and *SET_CR6* respectively. The rules modelling the early messages of the protocol and the rules that are common to most protocols, such as the definition of the Spy's capabilities, are omitted.

Each rule details how to extend a given trace of the protocol (*#* is the list "cons" operator) and refers to a typical CA, namely *CA i*, and a typical cardholder *C*, defined using the *Cardholder* constructor:

```
C = Cardholder k.
```

In rule *SET_CR5*, variable *evs5* refers to the current event trace. The preconditions of the rule require the cardholder to issue two fresh nonces *NC3* and *CardSecret*, and two fresh symmetric keys, *KC2* and *KC3*. Also, two events must have occurred in *evs5*: the *Says* event signifies that *C* sent an appropriate instance of message 3 to the CA; the *Gets* event signifies that *C* received the CA's reply, which carries a certificate signed by the root certification authority and establishing *EKi* to be the CA's public encryption key[2]. Another certificate states that *SKi* is the CA's public signature key.

[2]The flag *onlyEnc* in the certificate indicates that it refers to an encryption key, while *onlySig* indicates a signature key.

```
SET_CR5:
  "⟦evs5 ∈ set_cr;  C = Cardholder k;
    Nonce NC3 ∉ used evs5; Nonce CardSecret ∉ used evs5;
    NC3 ≠ CardSecret;
    Key KC2 ∉ used evs5; KC2 ∈ symKeys;
    Key KC3 ∉ used evs5; KC3 ∈ symKeys; KC2≠KC3;
    Gets C {|sign (invKey SKi) {|Agent C, Nonce NC2, Nonce NCA|},
              cert (CA i) EKi onlyEnc (priSK RCA),
              cert (CA i) SKi onlySig (priSK RCA)|}
        ∈ set evs5;
    Says C (CA i)
         {|Crypt KC1 {|Agent C, Nonce NC2, Hash (Pan (pan C))|},
          Crypt EKi {|Key KC1, Pan (pan C),
                               Hash {|Agent C, Nonce NC2|}|}|}
         ∈ set evs5⟧
⟹ Says C (CA i)
         {|Crypt KC3
          {|Agent C, Nonce NC3, Key KC2, Key (pubSK C),
           Crypt (priSK C)
             (Hash {|Agent C, Nonce NC3, Key KC2,
                    Key(pubSK C), Pan(pan C), Nonce CardSecret|})|},
          Crypt EKi {|Key KC3, Pan (pan C), Nonce CardSecret|}|}
    # evs5 ∈ set_cr"


SET_CR6:
  "⟦evs6 ∈ set_cr;
    Nonce NonceCCA ∉ used evs6;
    KC2 ∈ symKeys;  KC3 ∈ symKeys;  cardSK ∉ symKeys;
    Notes (CA i) (Key cardSK) ∉ 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, Pan(pan C), Nonce CardSecret|})|},
                  Crypt (pubEK (CA i)) {|Key KC3, Pan (pan C),
                                         Nonce CardSecret|}|}
      ∈ set evs6⟧
⟹ Says (CA i) C (Crypt KC2
                   {|sign (priSK (CA i))
                    {|Agent C,Nonce NC3,Agent(CA i), Nonce NonceCCA|},
                     certC (pan C) cardSK (XOR(CardSecret,NonceCCA))
                                   onlySig (priSK (CA i)),
                     cert (CA i) (pubSK(CA i)) onlySig (priSK RCA)|})
  # Notes (CA i) (Key cardSK)
  # evs6 ∈ set_cr"
```

Figure 3: Modelling cardholder registration (fragment)

Then, *C* encrypts using *EKi* a message containing his credit card number (*pan C*) and the key *KC3*, and encrypts using *KC3* a message containing the symmetric key *KC2* and the public signature key to be certified. The two encrypted messages constitute a digital envelope, which *C* sends to the CA.

In rule *SET_CR6*, variable *evs6* refers to the current event trace. The rule may fire when the CA receives an instance of message 5, requesting a certificate for the key *cardSK*. The rule lets CA send protocol message 6, a digital envelope containing the desired certificate and encrypted by a symmetric key received from the cardholder. The certificate also contains the PANSecret, which is computed as the exclusive-OR of the *CardSecret* (sent by the cardholder) and *NonceCCA* (generated by the CA). While sending the message, the CA stores the key just certified in order to prevent its being certified more than once. The rule for message 6 checks that the key *cardSK* has not previously been registered by imposing the precondition

*Notes (CA i) (Key cardSK)* $\notin$ *set.*

Since our earlier work on this protocol [2], we have streamlined the model. For example, the *Notes* event in *SET_CR6* elegantly replaces a stronger precondition. Other notions, such as the set of crucial keys for decrypting SET messages have also been eliminated. We have found a simple formalization of the many types of agents and their keys.

Modelling the generation of fresh public/private key pairs is not difficult, as we have reported already [2]. It involves replacing pubSK $C$ and privSK $C$ with variables ranging over keys, and extend the preconditions of *SET_CR5*, with a new requirement: the newly introduced public keys are not used and are not equal to any symmetric key.

### 3.2 Modelling Merchant Registration

The message constructors defined above can be reused to specify merchant registration. The inductive rules modelling the last two messages of the protocol appear in Figure 4.

Rule *SET_MR3* specifies that the merchant *M* generates a single session key *KM1* and asks for certification of both his public keys. This differs from the previous protocol, where cardholder generates two session keys and asks for certification of his signature key only. The rule may fire only if the merchant sent message 1 of the protocol, as stated by the *Says* event, and received message 2, as stated by the *Gets* event.

If the CA agrees to certify the merchant's keys, it must also record them, as stated by rule *SET_MR4*. The conclusion of the rule adds the three corresponding events to the current trace. The merchant's certificates have the same form as the CA's certificate — indeed, all of them are expressed using the same message constructor, *cert*. These certificates are sent in clear, since the message for issuing certificates "shall be signed but not

```
SET_MR3:
  "⟦evs3 ∈ set_mr; M = Merchant k; Nonce NM2 ∉ used evs3;
    Key KM1 ∉ used evs3;  KM1 ∈ symKeys;
    Gets M {|sign (invKey SKi) {|Agent X, Nonce NM1, Nonce NCA|},
            cert (CA i) EKi onlyEnc (priSK RCA),
            cert (CA i) SKi onlySig (priSK RCA)|}
      ∈ set evs3;
    Says M (CA i) {|Agent M, Nonce NM1|} ∈ set evs3⟧
 ⟹ Says M (CA i)
         {|Crypt KM1 (sign (priSK M) {|Agent M, Nonce NM2,
                                       Key(pubSK M), Key(pubEK M)|}),
           Crypt EKi (Key KM1)|}
       # evs3 ∈ set_mr"


SET_MR4:
  "⟦evs4 ∈ set_mr; M = Merchant k;
    merSK ∉ symKeys;  merEK ∉ symKeys;
    Notes (CA i) (Key merSK) ∉ set evs4;
    Notes (CA i) (Key merEK) ∉ set evs4;
    Gets (CA i) {|Crypt KM1 (sign (invKey merSK)
                          {|Agent M, Nonce NM2, Key merSK, Key merEK|}),
                  Crypt (pubEK (CA i)) (Key KM1)|}
      ∈ set evs4⟧
⟹ Says (CA i) M {|sign (priSK (CA i))
                    {|Agent M, Nonce NM2, Agent (CA i)|},
                  cert   M   merSK  onlySig (priSK (CA i)),
                  cert   M   merEK  onlyEnc (priSK (CA i)),
                  cert (CA i) (pubSK (CA i)) onlySig (priSK RCA)|}
      # Notes (CA i) (Key merSK)
      # Notes (CA i) (Key merEK)
      # evs4 ∈ set_mr"
```

Figure 4: Modelling merchant registration (fragment)

encrypted if the [certificate recipient] is a Merchant or Payment Gateway" [9, p.191]. However, SET requires the last message of cardholder registration to be encrypted.

Merchant registration is simpler than cardholder registration. It involves fewer sensitive components. There is no equivalent of the PAN or of the PANSecret, and there are fewer digital envelopes.

# 4 Secrecy Proofs for Cardholder Registration

For cardholder verification we proved 64 theorems in total; for merchant registration we proved 31. These include all the main goals for these protocols and all necessary lemmas. We have space to present only a small selection. We concentrate on the most difficult and interesting proofs concerning secrecy in cardholder registration. Here we have introduced the new methods to deal with digital envelopes.

A primary goal is that cardholder registration guarantees secrecy of the PANSecret. No message of the protocol sends this number, not even in encrypted form. Rather, both parties compute it as the exclusive-OR of other numbers. So, do those numbers remain secret? Since they are encrypted using symmetric keys, the proof requires a lemma that symmetric keys remain secret.

The first complication is that some symmetric keys do *not* remain secret, namely those involving a compromised CA. The second, major complication is that some symmetric keys are used to encrypt others: the loss of one key can compromise a second key, leading possibly to unlimited losses.

The problem of one secret depending on another has occurred previously, with the Yahalom [12] and Kerberos [3] protocols. Both of these are comparatively simple: the dependency relation links only two items. Cardholder registration has many dependency relationships. It also has a dependency chain of length three: in the last message, a secret number is encrypted using a key (`KC2`) that was itself encrypted using another key (`KC3`).

To solve this problem, we have generalized the method described in earlier work to chains of any length. While the definitions become more complicated than before, they follow a uniform pattern. The idea is to define a relation, for a given trace, between pairs of secret items: $(K, X)$ are related if the loss of the key $K$ leads to the loss of the key or nonce $X$. Two new observations can be made about the dependency relation:

- It should ignore messages sent by the Spy, since we can only hope to prove secrecy for honest participants. This greatly simplifies some proofs.

- It must be transitive, since a dependency chain leading to a compromise could have any length. Past protocols were too simple to reveal

```
KeyCryptKey_Nil:
  "KeyCryptKey DK K [] = False"

KeyCryptKey_Cons:
  "KeyCryptKey DK K (ev # evs) =
   (KeyCryptKey DK K evs ∨
    (case ev of
       Says A B Z ⇒
         ((∃N X Y. A ≠ Spy ∧
                   DK ∈ symKeys ∧
                   Z = {|Crypt DK {|Agent A, Nonce N, Key K, X|}, Y|}) ∨
          (∃C. DK = priEK C))
     | Gets A' X ⇒ False
     | Notes A' X ⇒ False))"
```

Figure 5: Association between keys in cardholder registration

this point.

Secrecy of session keys is proved as it was for Kerberos IV [3], by defining the relation *KeyCryptKey DK K evs* that takes two keys *DK* and *K*, and an event trace *evs*. It holds on a trace containing a message in which the first key encrypts the second key. As we shall see, reasoning about *KeyCryptKey* will allow us to prove that most symmetric keys remain secure.

From that result, one might think it would be easy to prove that nonces encrypted using those keys remain secret. However, secrecy proofs for nonces appear to require the same treatment as secrecy proofs for keys. We must define the dependency relation between keys and nonces. Then the proofs can be carried forward as it was for Yahalom [12], except that there are many key-nonce relationships rather than one.

## 4.1 Relations between secrets

The relation *KeyCryptKey DK K evs* is defined as a primitive recursive function in Figure 5. Rule *KeyCryptKey_Nil*, the base case of the recursion, states that the relation is false on an empty trace. Rule *KeyCryptKey_Cons* formalizes the recursive step. For the relation to hold on the extended trace *ev#evs* the relation must either hold on the original trace *evs*, or the new event *ev* must have a specific structure. It could be an instance of message 5 in which some principal who is not Spy uses *KC3* to encrypt *KC2* in the event trace *evs*. Alternatively, *ev* could be any event in which somebody encrypts *KC3* using a public key. In the latter case, *KeyCryptKey* holds of the corresponding private key, which can decrypt the message. In reading the definition, note that "∨" denotes logical disjunction, while "*|*" is part of the "*case*" syntax.

Figure 6 defines the dependency relation for nonces. Here are some hints

```
KeyCryptNonce DK N (ev # evs) =
 (KeyCryptNonce DK N evs ∨
  (case ev of
    Says A B Z ⇒
     A ≠ Spy ∧
     ((∃X Y. Z = {|Crypt DK {|Agent A, Nonce N, X|}, Y|}) ∨
      (∃K i X Y.
        Z = Crypt K {|sign (priSK i) {|Agent B, Nonce N, X|}, Y|} ∧
        (DK=K ∨ KeyCryptKey DK K evs)) ∨
      (∃K i NC3 Y.
        Z = Crypt K
              {|sign(priSK i) {|Agent B, Nonce NC3, Agent(CA i), Nonce N|},
               Y|} ∧
        (DK=K ∨ KeyCryptKey DK K evs)) ∨
      (∃i. DK = priEK i))
  | Gets A' X ⇒ False
  | Notes A' X ⇒ False))
```

Figure 6: Association between keys and nonces in cardholder registration

towards understanding this definition. The only important case involves *Says* events. The first disjunct refers to message 5 (shown above in §3), where key *KC3* encrypts nonce *NC3*; it also covers a similar encryption in message 3. The second and third disjuncts refer to message 6; they involve *KeyCryptKey* because that encryption uses a key received from outside. The fourth disjunct essentially says that we are not interested in asymmetric keys (they are never sent, so there is no risk of compromise).

## 4.2 Verification of Secrecy Properties

Now we outline the verification of cardholder registration. The handling of fresh public keys does not add technical difficulties thanks to Isabelle's level of automation. So, we have streamlined the model in this section to the case where the public/private key pair proposed by the cardholder is not invented but rather functionally assigned to him in the model. However, no agent knows another agent's public keys at the start of a run, but instead uses keys supplied in public key certificates.

Secrecy properties cause all our difficulties and we concentrate on them here. We first sketch the key steps of the proof informally to give an overview of the proof effort. We begin with three major lemmas:

- keys can be compromised only through the disclosure of other keys;
- keys sent by cardholders to uncompromised CAs are never disclosed;
- nonces cannot be compromised through the disclosure of keys;

Building on these lemmas, we are able to prove our key theorems:

- the CardSecret is secure, if the cardholder sends the **certificate request** message to an uncompromised CA;
- the NonceCCA is secure, if it is contained in a **cardholder certificate** received by the cardholder from an uncompromised CA;
- the PAN is secure, unless the cardholder has sent a **certificate request** message to a compromised CA.

Obviously, we have to trust the certification authority. The CA's task is to certify that there is a correspondence between a certificate and a credit card number. To obtain this goal, the CA must be able to see the credit card number.

In the sequel, each theorem is stated first in English and then using Isabelle notation. Each has been mechanically verified with Isabelle/HOL, typically by some form of induction. Some of them are so-called regularity properties, which are easy to prove [11]. For example, one protocol goal is almost trivial: if a certificate bears the signature of an uncompromised CA, then it was sent by the CA.

To prove our main results we need a number of preliminary technical lemmas. For example, we never have `KeyCryptKey DK K evs` where `DK` is fresh (in the trace `evs`), since a fresh key cannot have been used to encrypt anything. Several other obvious properties of `KeyCryptKey` turn out to be needed in the proofs below.

We can then move on to the *session key compromise theorem*. It states that a key can be lost only by the keys related to it by `KeyCryptKey`. It is used in other proofs to reason about situations in which some session keys might be compromised.

**Lemma 1 (symKey_compromise)** *No symmetric key can be compromised through the disclosure of other keys except in trivial cases.*

```
⟦evs ∈ set_cr; SK ∈ symKeys; ∀K∈KK. ¬ KeyCryptKey K SK evs⟧
⟹ (Key SK ∈ analz (Key ‘ KK ∪ knows Spy evs)) =
    (SK ∈ KK ∨ Key SK ∈ analz (knows Spy evs))
```

We can interpret this theorem as asserting that `KeyCryptKey` expresses all circumstances in which a symmetric key can become compromised. The proof is a big, difficult induction consisting of 10 proof commands. The simplification step requires a specialized set of rewrite rules, including lemmas about `KeyCryptKey`, and is relatively slow (14 seconds). The peculiar form of the lemma represents the generalization needed to make the induction succeed. Here are the preconditions in detail:

- *evs* ∈ *set_cr* simply means that *evs* is a trace of cardholder registration. All proofs about the protocol will include this assumption.
- *SK* ∈ *symKeys* means that *SK* is a symmetric key.
- ∀*K*∈*KK*. ¬ *KeyCryptKey K SK evs* means that *KK* is a set of keys, none of which immediately compromises *SK* in the trace *evs*.

In the conclusion, *Key SK* ∈ *analz (Key ‘ KK* ∪ *knows Spy evs)* means that *SK* can be derived from *KK* together with the Spy's knowledge (which mainly consists of observable traffic). The right-hand side of the conclusion is

```
(SK ∈ KK ∨ Key SK ∈ analz (knows Spy evs))
```

which means that either *SK* is itself a member of the set *KK* or *SK* is already derivable from the Spy's knowledge alone.

We could simplify the right-hand side above, and many other formulas, by defining *KeyCryptKey* to be reflexive. The intuition is attractive, for then *KeyCryptKey* would hold when there was a chain of decryptions from one key to another, of length possibly zero. However, this change would strengthen the precondition of Lemma 1, making it harder to prove (when we need to use the induction hypothesis) and harder to apply.

**Lemma 2 (symKey_secrecy)** *Symmetric keys sent by cardholders to uncompromised CAs are never disclosed.*

```
⟦CA i ∉ bad; K ∈ symKeys; evs ∈ set_cr;
 Says (Cardholder k) (CA i) X ∈ set evs; Key K ∈ parts {X}⟧
⟹ Key K ∉ analz (knows Spy evs)
```

This result follows from Lemma 1, but not trivially. It states a general law for any symmetric key that is part of (*Key K* ∈ *parts {X}*) any message sent by the cardholder:

```
Says (Cardholder k) (CA i) X ∈ set evs
```

Since the proof requires examination of all protocol steps, it involves another induction and the simplification again needs special rewrite rules concerning secrecy. It is not an explicit protocol goal — symmetric keys are just part of the underlying machinery — but it is obviously desirable.

**Lemma 3 (Nonce_compromise)** *No nonce can be compromised through the disclosure of keys except in trivial cases.*

```
⟦evs ∈ set_cr; ∀K∈KK. ¬ KeyCryptNonce K N evs⟧
⟹ (Nonce N ∈ analz (Key ‘ KK ∪ knows Spy evs)) =
    (Nonce N ∈ analz (knows Spy evs))
```

In both statement and proof, this result resembles Lemma 1. In particular, the precondition $\forall K{\in}KK.\ \neg$ *KeyCryptNonce K N evs* means that *KK* is a set of keys, none of which immediately compromises the nonce *N*. The conclusion is that the Spy could derive *N* with the help of the set *KK* only if he could have derived *N* without using that set. The situation is complicated by the many different nonces used in cardholder registration, only some of which are kept secret. So the proof is even longer than that of Lemma 1, despite its appealing to that lemma; it requires 14 proof steps and involves reasoning about both *KeyCryptKey* and *KeyCryptNonce*.

**Theorem 1 (CardSecret_secrecy)** *If a cardholder sends the* ***certificate request*** *message to an uncompromised CA, then the chosen CardSecret will remain secure.*

```
⟦CA i ∉ bad;
 Says (Cardholder k) (CA i)
      {|X, Crypt EKi {|Key KC3, Pan p, Nonce CardSecret|}|} ∈ set evs;
 Gets A {|Z, cert (CA i) EKi onlyEnc (priSK RCA),
             cert (CA i) SKi onlySig (priSK RCA)|} ∈ set evs;
 KC3 ∈ symKeys;  evs ∈ set_cr⟧
⟹ Nonce CardSecret ∉ analz (knows Spy evs)
```

This is an important goal: SET purchases are safe only if CardSecret is uncompromised. The proof involves another complicated induction despite its use of Lemmas 1 and 3. In the preconditions, note that the cardholder builds the digital envelope using any symmetric key *KC3*; the public key, *EKi*, is bound to the CA through a certificate signed by *RCA*. The main body of the argument is an induction that additionally assumes *KC3* to be uncompromised; later, an appeal to Lemma 2 removes that assumption.

**Theorem 2 (NonceCCA_secrecy)** *If a cardholder sends the* ***certificate request*** *message to an uncompromised CA and receives in response the* ***cardholder certificate****, then the value of NonceCCA contained in the latter will remain secure.*

```
⟦CA i ∉ bad;
 Gets (Cardholder k)
    (Crypt KC2
     {|sign(priSK(CA i)) {|Agent C, Nonce N, Agent(CA i), Nonce NonceCCA|},
       X, Y|}) ∈ set evs;
 Says (Cardholder k) (CA i)
    {|Crypt KC3 {|Agent C, Nonce NC3, Key KC2, X'|}, Y'|} ∈ set evs;
 Gets A {|Z, cert (CA i) EKi onlyEnc (priSK RCA),
             cert (CA i) SKi onlySig (priSK RCA)|} ∈ set evs;
 KC2 ∈ symKeys;  evs ∈ set_cr⟧
⟹ Nonce NonceCCA ∉ analz (knows Spy evs)
```

This result is as important as Theorem 1, since both CardSecret and NonceCCA are ingredients of the all-important PANSecret. The proof resembles that of Theorem 1 but is a bit more complicated, since it refers to two protocol messages. Variables such as `X` and `Y` refer to irrelevant parts of messages.

This theorem is a guarantee to the cardholder: it is expressed in terms of events that the cardholder can verify. We have not proved the analogous guarantees for the CA. Although NonceCCA originates with the CA, its compromise would do the CA no harm.

Theorems 1 and 2 are as close as we can come to expressing the secrecy of the PANSecret, since our model does not let us reason about exclusive-OR. Our next goal is to prove secrecy of the PAN: the credit card number.

**Lemma 4 (analz_insert_pan)** *No PAN can be compromised through the disclosure of symmetric keys.*

```
⟦evs ∈ set_cr;  K ∉ invKey ‘ pubEK ‘ range CA⟧
⟹ (Pan P ∈ analz (insert (Key K) (knows Spy evs))) =
    (Pan P ∈ analz (knows Spy evs))
```

This result resembles Lemma 3 but is much easier to prove because PANs are encrypted only with public keys. As the model does not allow public keys to be broken during a trace, no key dependency chains complicate the reasoning. In essence, the inductive argument examines all protocol messages to confirm that symmetric keys are never used to encrypt PANs.

The obscure premise `K ∉ invKey ‘ pubEK ‘ range CA` states that the key `K` is not the private encryption key of any CA. The lemma says that if the Spy can discover a PAN with the help of `K`, then he could have discovered the PAN without using `K`. To make the induction work, we had to prove a stronger statement (not shown); it generalizes the lemma above to replace `K` by a set of symmetric keys. The final guarantee about the PAN says that it remains secure unless it is sent to a compromised CA (for its private keys are known to the Spy).

**Theorem 3 (pan_confidentiality)** *If a PAN has been disclosed, then the cardholder has sent a* ***certificate request*** *message to a compromised CA.*

```
⟦Pan (pan C) ∈ analz(knows Spy evs); C ≠ Spy; evs ∈ set_cr⟧
⟹ ∃i X K HN.
      Says C (CA i) {|X, Crypt (pubEK(CA i)) {|Key K, Pan(pan C), HN|}|}
        ∈ set evs
   ∧ (CA i) ∈ bad
```

This result is proved by induction, using Lemma 4 and the usual rewriting rules for secrecy proofs.

Merchant registration is much easier to analyze than cardholder registration. The simpler form of the **certificate request** message eliminates the dependence between symmetric keys. The lack of the fields PAN, CardSecret and NonceCCA leaves us with little to prove secret.

# 5 Related Work and Conclusions

The general treatment of secrecy proofs, as exemplified in `KeyCryptKey` and `KeyCryptNonce`, is a major outcome of our work. The definitions of these relations are complicated but conform to an obvious pattern that could be automated. Verifying the registration protocols was valuable preparation for our later verification of the purchase protocols [1].

As far as we are aware, no other group has attempted to verify the SET registration protocols. Of the many other efforts into protocol verification, the most relevant is TAPS by Ernie Cohen [5]. Given a protocol, Cohen's system automatically generates a *secrecy invariant*, which serves the same purpose as the relations `KeyCryptKey` and `KeyCryptNonce`. Potentially, TAPS could verify cardholder registration, though the protocol's size and complexity may present difficulties in the automatic generation of invariants.

The complicated RSA digital envelopes and signature conventions make proofs difficult and slow. Compared with other protocols that researchers have verified, cardholder registration uses encryption heavily, resulting in gigantic terms or complex case splits. Sometimes Isabelle presents the user with subgoals spanning several pages of text. One should not attempt to prove such a monstrosity directly. One useful strategy is to look for terms that can be simplified and prove the corresponding rewrite rules. This may cut the monstrosity down to size.

Our model does not include the algebraic properties of exclusive-OR, such as $X \oplus Y \oplus X = Y$, and this prevents us from proving the security of the PANSecret. We assume that $X \oplus Y$ is secure if both $X$ and $Y$ are. Our treatment of the PANSecret amounts to assuming that it is computed as the hash of $X$ and $Y$, which would certainly be an improvement over exclusive-OR. A bad CA can force the PANSecret to take on a chosen value $N$ by setting NonceCCA to be CardSecret $\oplus$ $N$. The cardholder has no defence against this attack unless he knows the value of $N$.


## Acknowledgments

F. Massacci was supported by CNR and MURST grants. Part of this work was done while P. Tramontano was visiting the Computer Laboratory in Cambridge under a visiting scholarship for masters students by the University of Roma I "La Sapienza." The work at Cambridge was funded by the EPSRC grant GR/R01156/R01 *Verifying Electronic Commerce Protocols.*

# References


[1] G. Bella, F. Massacci, and L. C. Paulson. Verifying the SET purchase protocols. Technical Report 524, Computer Laboratory, University of Cambridge, Nov. 2001.

[2] G. Bella, F. Massacci, L. C. Paulson, and P. Tramontano. Formal verification of cardholder registration in SET. In F. Cuppens, Y. Deswarte, D. Gollman, and M. Waidner, editors, *Computer Security — ESORICS 2000*, LNCS 1895, pages 159–174. Springer, 2000.

[3] G. Bella and L. C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In J.-J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann, editors, *Computer Security — ESORICS 98*, LNCS 1485, pages 361–375. Springer, 1998.

[4] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Technical report, University of York, Department of Computer Science, November 1997. Available on the web at `http://www-users.cs.york.ac.uk/~jac/`. A complete specification of the Clark-Jacob library in CAPSL is available at `http://www.cs.sri.com/~millen/capsl/`.

[5] E. Cohen. TAPS: A first-order verifier for cryptographic protocols. In *Proc. of the 13th IEEE Comp. Sec. Found. Workshop*, pages 144–158. IEEE Comp. Society Press, 2000.

[6] J. Guttman. Security goals: Packet trajectories and strand spaces. In R. Focardi and F. Gorrieri, editors, *Foundations of Security Analysis and Design - Tutorial Lectures*, volume 2171 of *Lecture Notes in Comp. Sci.*, pages 197–261. Springer-Verlag, 2001.

[7] G. Lowe. Casper: A compiler for the analysis of security protocols. *J. of Comp. Sec.*, 6(18-30):53–84, 1998.

[8] Mastercard & VISA. *SET Secure Electronic Transaction Specification: Business Description*, May 1997. Available electronically at `http://www.setco.org/set_specifications.html`.

[9] Mastercard & VISA. *SET Secure Electronic Transaction Specification: Programmer's Guide*, May 1997. Available electronically at `http://www.setco.org/set_specifications.html`.

[10] C. Meadows. Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer. In *SSP-99*, pages 216–231. IEEE Comp. Society Press, 1999.

[11] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. of Comp. Sec.*, 6:85–128, 1998.

[12] L. C. Paulson. Relations between secrets: Two formal analyses of the Yahalom protocol. *J. of Comp. Sec.*, 9(3):197–216, 2001.

[13] P. Ryan and S. Schneider. An attack on a recurive authentication protocol. a cautionary tale. *Inform. Processing Lett.*, 65(15):7–16, 1998.