SET Cardholder Registration: the Secrecy Proofs

(Extended Abstract)

Lawrence C. Paulson

Computer Laboratory, Univ. of Cambridge, Cambridge, England
lcp@cl.cam.ac.uk

1 Introduction

Security protocols aim to protect the honest users of a network from the dishonest ones. *Asymmetric* (public key) cryptography is valuable, though it is normally used in conjunction with *symmetric* cryptography, where two users share a secret key, Asymmetric cryptography is typically used to securely exchange symmetric keys, which carry the bulk of the traffic. This mode of operation is faster than using expensive public-key encryption exclusively. It is also more secure, since the symmetric keys can be changed frequently.

The protocol used to set up of this type of communication must be designed with care. For example, each message typically includes a random number that the other party includes in his reply as proof that it is not an old message replayed by an intruder. This random number, or *nonce*, may be 20 bytes long, making the chance of an accidental collision infinitesimal. Many flaws have been discovered in security protocols [5].

Security protocol verification technologies have progressed in recent years. A variety of tools are available for analyzing protocols. Model checking is excellent for debugging a protocol, finding attacks in seconds [6, 7]. Theorem proving is valuable too: it can analyze protocols in more detail and handles the protocols that are too big for model checking. Subgoals presented to the user suggest possible failure modes and give insights into how the protocol operates.

Past work on protocol verification has focused on protocols arising from the academic community. Only seldom have deployed protocols been investigated, such as Kerberos [3], SSL [8] and SSL's successor, TLS [12]. Past work has largely focused on *key exchange* protocols. Such protocols allow two participants (invariably called Alice and Bob) to agree on a *session key*: a short-term symmetric key. In this paper, I would like to describe a project, joint with Bella, Massacci and Tramontano, to verify a very large commercial protocol: SET, or Secure Electronic Transactions [15].

2 The SET Protocol

People normally pay for goods purchased over the Internet using a credit card. They give their card number to the merchant, who claims the cost of the goods against it. To prevent eavesdroppers from stealing the card number, the transaction is encrypted using the SSL protocol. This arrangement requires the customer and merchant to trust each other: an undesirable requirement 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: pornographers have charged more than the advertised price, expecting their customers to be too embarrassed to complain. Some merchants are incompetent: a million credit card numbers have recently been stolen from Internet sites whose managers had not applied patches (available free from Microsoft) to fix security holes [9].
- 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 not the cardholder but the merchant who has the most to lose from fraud. Legislation in most countries protects the consumer.

The SET protocol aims to reduce fraud by introducing a preliminary registration phase. Both 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. These credentials do not contain sensitive details such as credit card numbers. Later, when the customer wants to make purchases, he and the merchant exchange their credentials. If both parties are satisfied then they can proceed with the transaction. Credentials must be renewed every few years, and presumably are not issued to known fraudsters.

SET comprises 15 subprotocols, or *transactions*, in all. Some observers, noting its extreme complexity, predict that it will never be deployed. However, the recent large rise in credit card fraud [1] suggests that current arrangements are unsustainable. SET or a derivative protocol may well be deployed in the next several years. To a researcher, SET has a further attraction: it makes heavy use of primitives such as digital envelopes that protocol verifiers have not examined before now.

3 Cardholder Registration

As described above, each cardholder must register before he is allowed to make purchases. He proves his identity by supplying 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. In keeping with normal practice, SET

requires each participant to have separate key pairs for signature and encryption.

Cardholder registration comprises six messages:

1. The cardholder contacts the CA to request registration.
2. The CA replies, returning its public key certificates. These contain the CA's public keys (which the cardholder needs for the next phase) and are signed by the Root Certificate Authority (so that the cardholder knows they are genuine).
3. 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.)
4. The CA uses the credit card number to determine the cardholder's issuing bank and returns an appropriate registration form.
5. The cardholder chooses an asymmetric public/private key pair. He submits the public key along with the completed registration form to the CA, who forwards it to the bank.
6. The bank checks the various details, and if satisfied, authorises the CA to issue credentials. The CA signs a certificate that includes the cardholder's public signature key and the cryptographic hash[1] of a secret number known to the cardholder. When making purchases, the cardholder will use this number. the **PANSecret**, as proof of identity. Finally the cardholder receives the credentials and is ready to go shopping.

Does verifying cardholder registration serve any purpose? The payment phase performs the actual E-commerce, and protocol verifiers often assume that participants already possess all needed credentials. However, cardholder registration is a challenging protocol, particularly when it comes to proving that the PANSecret is actually secret.

The most interesting feature of cardholder registration, from the viewpoint of verification, is its use of *digital envelopes*. To send a long message to the CA, the cardholder generates a fresh symmetric key and encrypts the message, using public key encryption only to deliver the session key to the CA. As mentioned at the start of this paper, this combination of symmetric and asymmetric encryption is more efficient and secure than using asymmetric encryption alone. However, the two-stage process makes a protocol harder to analyze. The most complicated case is with the last message exchange, where the cardholder sends the CA *two* session keys. One of these keys encrypts the cardholder's message and the other encrypts the CA's reply.

We could simplify the protocol by eliminating digital envelopes and removing unnecessary encryption. However, the resulting protocol would be trivial. Experience shows that simplifying out implementation details can hide major errors [14]. Cardholder registration is valuable preparation for the eventual verification of the purchase phase.

[1] A *cryptographic hash* maps messages to fixed-length blocks. It is infeasible to recover a message from its hash or to create two messages having the same hash.

4 The Inductive Model

We use the inductive method of protocol verification, which has been described elsewhere [11, 13]. 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, meaning 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:

- $\mathsf{Says}\, A\, B\, X$ means A sends message X to B.
- $\mathsf{Gets}\, A\, X$ means A receives message X.
- $\mathsf{Notes}\, A\, X$ means A stores X in its internal state.

The model of Cardholder Registration is largely the work of Bella, Massacci and Tramontano, who devoted many hours to decrypting 1000 pages of SET documentation [2]. We have flattened the hierarchy of certificate authorities. The Root Certificate Authority is responsible for certifying all the other CAs. Our model includes compromised CAs — as naturally it should — though we assume that the root is uncompromised. The compromised CAs complicate the proofs considerably, since large numbers of session keys and other secrets fall into the hands of the Spy. Here is a brief summary of the notation:

- *set_cr* is the set of traces allowed by Cardholder Registration
- *used* is the set of items appearing in the trace, to express freshness
- *symkeys* is the set of symmetric keys[2]
- *Nonce*, *Key*, *Agent*, *Crypt* and *Hash* are message constructors
- $\{\!|X_1, \ldots, X_n|\!\}$ is an n-component message

Here is part of the specification, the inductive rule for message 5. Variable *evs5* refers to the current event trace:

```
⟦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;
 cardSK ∉ symKeys;  ...
 Gets C ... ∈ set evs5;
 Says C (CA i) ... ∈ 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, Pan(pan C), Nonce CardSecret|})|},
        Crypt EKi {|Key KC3, Pan (pan C), Nonce CardSecret|}|}
 # evs5 ∈ set_cr
```

[2] In an implementation, a symmetric key occupies 8 bytes while an asymmetric one occupies typically 128 bytes, so the two types are easily distinguishable.

Much has been elided from this rule, but we can see several things:

- the generation of two fresh nonces, `NC3` and `CardSecret`
- the generation of two fresh symmetric keys, `KC2` and `KC3`, to be used as session keys
- a message encrypted using `EKi` (the CA's public key) and containing the credit card number (`pan C`) and the key `KC3`
- a message encrypted using `KC3` and containing the symmetric key `KC2` and the cardholder's public signature key, `cardSK`

The two encrypted messages constitute a digital envelope.

5 The Secrecy Proofs

Secrecy of the PANSecret must be proved. This number, mentioned in §3, is never transmitted, but is computed as the exclusive-OR of other secret numbers generated by the cardholder and the CA. So, do these numbers remain secret? Since they are encrypted using symmetric keys, the proof requires a lemma that symmetric keys remain secret. Two complications are that some symmetric keys do *not* remain secret, namely those involving a compromised CA, and that some symmetric keys are used to encrypt others. The latter point means that 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 [10] and Kerberos [3] protocols. Both of these are 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`).

Fortunately, the method described in earlier work generalizes naturally to this case and 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 nothing belonging to him counts as secret. 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 this point.

Secrecy of session keys is proved as it was for Kerberos IV [3], by defining the relation `KeyCryptKey DK K evs`. This relation captures instances of message 5 in which somebody other than the Spy uses `KC3` to encrypt `KC2` in the event trace `evs`. The *session key compromise theorem* states that a given key can be lost only by the keys related to it by `KeyCryptKey`. The form of this lemma has been discussed elsewhere [10]; it handles cases of the induction in which some session keys are compromised. Using this lemma, we can prove that no symmetric keys are lost in a communication between honest participants:

```
⟦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)
```

Any symmetric key that is part of a message `X` sent by a cardholder (that is, `Key K ∈ parts {X}`) is not derivable from material visible to the Spy (that is, `Key K ∉ analz (knows Spy evs)`).

Given that the session keys are secure, we might hope to find a simple proof that nonces encrypted using those keys remain secret. However, secrecy proofs for nonces require the same treatment as secrecy proofs for keys. We must define the dependency relation between keys and nonces and prove a lemma analogous to the one shown above.

Secrecy of nonces is proved as it was for Yahalom [10], except that there are many key-nonce relationships rather than one. Here is the dependency relation:

```
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))
```

Here are some hints towards understanding this definition. The only important case involves `Says` events. The first disjunct refers to message 5 (shown above in §4), 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).

Finally, we can show that the secrets exchanged by the parties in the final handshake remain secure.

```
⟦CA i ∉ bad;
 Says (Cardholder k) (CA i)
  {|X, Crypt EKi {|Key KC3, Pan p, Nonce CardSecret|}|} ∈ set evs;
 ...; evs ∈ set_cr⟧
⟹ Nonce CardSecret ∉ analz (knows Spy evs)
```

This theorem concerns the cardholder's secret. There is an analogous one to confirm the security of the CA's secret. These two numbers, remember, are the ingredients of the all-important PANSecret.

G. Bella has proved that the credit card number also remains secret. It looks straightforward: the number is encrypted using the CA's public key, which is secure provided the CA is uncompromised. As usual, however, the proof is harder than it looks. It requires a lemma stating that no symmetric keys are of any use to the spy for stealing a credit card number. This lemma looks obvious too, but both it and the main theorem are non-trivial inductions. Their proofs together require about one CPU minute.

Why are proofs so difficult and slow? The digital envelopes and digital signature conventions are to blame. Compared with other protocols analyzed using the inductive method, cardholder registration has nested encryption, resulting in huge case splits. The verifier is sometimes presented with a giant subgoal spanning many pages of text. One should not attempt to prove such a monstrosity but instead to improve the simplification so that it does not occur again.

6 Observations about Cardholder Registration

The proofs suggest that cardholder registration is secure. However, some anomalous features come to light. These do not derive from the formal analysis but merely by a close inspection of the protocol. A benefit of formal methods is that they encourage a close scrutiny of the object being verified.

There is unnecessary encryption. The cardholder's signature verification key is encrypted, even though it is a public key! The cardholder certificate is also encrypted, when it is of no use to anyone but the cardholder. Public-key certificates are nearly always sent in clear; this encryption is presumably intended to strengthen confidence in SET and to reassure cardholders. Nonces whose purpose is to ensure freshness do not have to be encrypted, but in SET they usually are. This forces `KeyCryptNonce` to take them into account, increasing the expression blow-up in secrecy proofs. We have a paradox: protocol designers who are concerned about security will include additional encryption, but that encryption actually makes the protocol more difficult to verify.

I observed two insecurities. The cardholder is not required to generate a fresh signature key pair, but may register an old one. There is a risk that this old one could be compromised. SET accordingly includes a further security measure: the PANSecret. The PANSecret is the exclusive-OR of numbers chosen by the two parties (see §5), and the cardholder chooses his number before the CA does. Since exclusive-OR is invertible, a criminal working for a CA can give every cardholder the same PANSecret.

This combination of insecurities introduces some risk that a criminal could impersonate the cardholder. The cardholder's implementation of SET can repair the first defect by always generating a fresh signature key pair. The second defect is, in principle, easy to fix: simply change the calculation of the PANSecret, replacing the exclusive-OR by cryptographic hashing. But that unfortunately is

a change to the protocol itself. If this change were made, then secrecy of the PANSecret would follow from secrecy of either of its two ingredients. With SET as it stands now, secrecy of the PANSecret is not provable in my model — nor should it be.

7 Conclusions

Our joint work has been fruitful. We had been able to specify and verify cardholder registration. Our model is abstract but retains much detail. We can prove secrecy in the presence of digital envelopes. We have strengthened our previous work on the relationships between secrets. There must be a connection with Cohen's secrecy invariant [4], though I am not sure of the details. We look forward to analyzing the remainder of SET.

Acknowledgements. Thanks above all to my colleagues G. Bella, F. Massacci and P. Tramontano for their many months devoted to understanding the SET specifications. (By contrast, the secrecy proofs reported above took only days.) This work was funded by the EPSRC grant GR/R01156/01 *Verifying Electronic Commerce Protocols.*

References

1. Credit card fraud rises by 50%. On the Internet at `http://news.bbc.co.uk/hi/english/business/newsid_1179000/1179590.stm`, February 2001. BBC News (Business).
2. Giampaolo Bella, Fabio Massacci, Lawrence C. Paulson, and Piero 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. Giampaolo Bella and Lawrence 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. Ernie Cohen. TAPS: A first-order verifier for cryptographic protocols. In *13th Computer Security Foundations Workshop*, pages 144–158. IEEE Computer Society Press, 2000.
5. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems: second international workshop, TACAS '96*, LNCS 1055, pages 147–166. Springer, 1996.
6. Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.
7. Catherine Meadows. Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer. In *Symposium on Security and Privacy*, pages 216–231. IEEE Computer Society, 1999.

8. John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0 and related protocols. In Hilarie Orman and Catherine Meadows, editors, *Workshop on Design and Formal Verification of Security Protocols*. DIMACS, September 1997.
9. Alan Paller. Alert: Large criminal hacker attack on Windows NTE-banking and E-commerce sites. On the Internet at `http://www.sans.org/newlook/alerts/NTE-bank.htm`, March 2001.
10. Lawrence C. Paulson. Relations between secrets: Two formal analyses of the Yahalom protocol. *Journal of Computer Security*. in press.
11. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
12. Lawrence C. Paulson. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, August 1999.
13. Lawrence C. Paulson. Proving security protocols correct. In *14th Annual Symposium on Logic in Computer Science*, pages 370–381. IEEE Computer Society Press, 1999.
14. Peter Y. A. Ryan and Steve A. Schneider. An attack on a recursive authentication protocol: A cautionary tale. *Information Processing Letters*, 65(1):7–10, January 1998.
15. SETCo. *SET Secure Electronic Transaction Specification: Business Description*, May 1997. On the Internet at `http://www.setco.org/set_specifications.html`.