# Modelling Security Protocols Based on Smart Cards

Giampaolo Bella
Computer Laboratory — University of Cambridge
New Museums Site, Pembroke Street
Cambridge CB2 3QG (UK)

Giampaolo.Bella@cl.cam.ac.uk

## Abstract

*Security protocols based on smart cards lack formal analysis. Shoup & Rubin's theoretical treatment, which is based on previous work by Bellare & Rogaway, is the best example in this small field. I propose an alternative approach obtained as an extension to Paulson's Inductive Approach, which is based on theorem proving. The formal overhead is smaller, but the global expressiveness results adequate. For example, smart cards can be lost or cracked by an eavesdropper. The fairly intricate Shoup-Rubin protocol is chosen as a benchmark, and several guarantees are proved to show that the protocol model behaves as it is meant to. The proposed approach might make the formal analysis of protocols based on smart cards a more common practice.*

## 1 Introduction

Smart cards have become inexpensive. They can be used as secure tokens to store agents' cryptographic secrets (e.g. [2, 6]). However, because of their small dimensions, they run the risk to get lost or stolen by an eavesdropper. To limit the damage in such circumstances, they are often guarded with a PIN that is needed to activate their functions. Therefore, it is not surprising that traditional password-based systems are currently turning into smart-card-based ones. In particular, this applies to the authentication systems based on security protocols (e.g. [8]).

The new protocols claim to achieve stronger goals. However, the little effort done to prove these goals formally is rather surprising. Abadi et al. [1] have validated a range of protocols based on smart cards using a BAN [7] like formalism whose limitations are well known. Bellare & Rogaway [5] have recently proposed a well-founded formal notion of *provable security* for traditional protocols. However, their notion seems rather theoretical. Later, Shoup & Rubin have used it to design a new protocol based on smart cards and prove the protocol secure [15]. Their protocol has been implemented [9].

I set out to develop a formal yet intuitive approach for security protocols based on smart cards. I chose Paulson's Inductive Approach [14], which is based on the theorem prover Isabelle/HOL [11] and has been used successfully with traditional protocols [4, 12, 13]. This paper describes my extensions to the original approach (new network events, new notion of agents' knowledge), and provides general guidance on how to model any protocol.

I do not aim here at achieving the complete formal analysis of a particular protocol. My goal is providing a general approach to build models that are as close as possible to the real world. I argue that my treatment will be understandable to the readers outside the theorem-proving community. Formal analyses must be vouched for by a broad, non-specific readership in order to be industrially significant.

My approach is independent from the computational power of smart cards. It allows the agents to conspire with the eavesdropper, so revealing their PINs. The eavesdropper can access a set of cards that have been lost by the owners, and even use a set of duplicates of other agents' cards (the "cracked" cards). Cards have no failure modes, so the spy only gets significant data out of them.

In the sequel, I model a version of the Shoup-Rubin protocol [15] that provides each agent with a PIN to activate her smart card. I also show that the protocol model behaves as expected by proving several theorems about each agent's use of her own card.

## 2 The Inductive Approach idea

Consider an infinite population of agents running a security protocol. Various events (e.g. sending and noting messages) take place during their protocol sessions. A *trace* is a list of those events; the formal protocol model is the set of *all* possible traces. This notion captures all the real-world network configurations, and provides an operational model for the network.

Mathematical induction is used to specify such a set. The base case states that the empty trace belongs to the set; this formalises the initial situation when no events have occurred on the network. The inductive steps specify which messages can be sent at any point of the protocol history; e.g. if the event formalising the first step of the protocol appears, then the event formalising the second step *may* appear.

Agents are not forced to operate. A subset of agents (the "bad" agents) have conspired with the eavesdropper (the "spy"), revealing their keys to her. All properties are proved by induction on the formal protocol model, and proofs are mechanised by Isabelle/HOL [11]. The complete details can be found elsewhere [14].

## 3 An inductive approach for smart-card-based protocols

This section presents my inductive approach for protocols based on smart cards, pointing out the extensions to Paulson's approach for traditional protocols. I have formalised new events to allow the agents to use their smart cards — using a smart card means feeding it with an input of the expected form and obtaining its output. Smart cards can be *lost* to the spy, or even *cracked* if the spy manages to duplicate them; these two sets are not related to each other. I also include the smart cards' PINs, and define a novel notion of knowledge.

### 3.1 The network events

In the original Paulson's approach, agents are only allowed to send messages, and to take note of relevant items. Agent $A$ sending message $X$ to agent $B$ is formalised by the event $\mathsf{Says}\, A\, B\, X$; $A$ taking note of $X$ is formalised by $\mathsf{Notes}\, A\, X$. I have recently introduced a notion of reception: $A$ receiving $X$ is formalised by $\mathsf{Gets}\, A\, X$ [3]. If a message was sent, then it can eventually be received by the recipient.

In a scenario based on smart cards, the serial communication between an agent and her card is normally assumed to be secure (e.g. [9]). Therefore, I define two new events: $\mathsf{Inputs}\, A\, C\, X$ and $\mathsf{Outputs}\, C\, A\, X$ expressing respectively agent $A$ feeding card $C$ with message $X$, and $A$ obtaining $X$ from $C$. The spy will not be able to monitor them (see below).

### 3.2 The smart cards

I have introduced a new Isabelle type $\mathsf{card}$ to formalise the smart cards. To associate an agent to her card, define the function

$$\mathsf{Card} : \mathsf{agent} \longrightarrow \mathsf{card}$$

In the real world, smart cards can be lost. I allow the worst case: the spy has got hold of all lost cards. Moreover, despite the progress in smart card technology, which makes them more and more tamper resistant, I allow a set of cards to have been cracked by the spy. I can easily define these two sets of cards

$$\mathsf{lost} \subseteq \mathsf{card} \qquad \mathsf{cracked} \subseteq \mathsf{card}$$

There is no need to assume that some cards are not lost, or that some cards are not cracked. However, if a card is lost, my model will not allow its owner to use it because the card lays in the spy's hands.

Similarly, there is no need to state any relations between the two sets. In the real world, if an agent has currently her card at hand, she cannot assume it to be not cracked, because the spy could have stolen it, cracked it, and returned it. On the other hand, if a card is lost, the spy is not necessarily able to crack it. Moreover, no relation exists between an agent being bad and her card being lost or cracked. Conversely, I do state that the spy's smart card is not lost nor cracked, meaning that she can only use it in a legitimate way and not as an oracle performing other agents' cards' operations.

Each smart card stores its PIN key internally. Since an agent knows the PIN to activate her smart card, I prefer to associate PINs to agents

$$\mathsf{pin} : \mathsf{agent} \longrightarrow \mathsf{key}$$

rather than to cards. Therefore, the PIN of a card is regarded as the card's owner's PIN. The spy has to fetch the PINs of the lost cards in order to use them, but I assume she does know the PINs of the cracked ones (see below).

Each card also stores its owner's long-term key (shared with the trusted server), formalised by Paulson's function $\mathsf{shrK} : \mathsf{agent} \longrightarrow \mathsf{key}$, and its own long-term key, which I formalise straightforwardly by the function

$$\mathsf{crdK} : \mathsf{card} \longrightarrow \mathsf{key}$$

The notion of knowledge presented below will allow the spy to know only those keys contained in the cracked cards.

### 3.3 The agents' knowledge

I introduce an inductive definition of agents' knowledge taking into account the new events defined above. The base of the induction states:

- each honest agent knows the PIN that activates her smart card;
- the trusted server knows all keys (cards' keys, agents' keys, and PINs);
- the spy knows the PINs of bad agents (they have conspired with the spy), and all keys contained in cracked cards (cards' keys, cards' owners' keys, cards' owners' PINs).

The inductive steps must explain how agents extend their knowledge on each event that can occur on a trace.

**Message sending**: an agent knows those messages sent by herself; the spy knows all messages sent by anybody;

**Message noting**: an agent knows those messages noted by herself; the spy knows all messages noted by bad agents;

**Message reception**: an agent knows those messages received by herself; the spy knows all messages received by anybody;

**Card input**: an agent (spy included) knows those messages that the agent gives as input to any cards;

**Card output**: an agent (spy included) knows those messages that the agent receives as outputs from any cards;

The last two cases specify a secure communication between each agent and her card. Thus, the spy can only monitor her own communication with a smart card, but the protocol model will allow her to communicate with those cards that are cracked or lost by bad agents (the spy knows their PINs).

### 3.4 The protocol model

Specifying the operations of honest agents is rather straightforward. For each protocol step, I write an inductive rule of the form: if message $i$ is received by the intended recipient $P$, then $P$ can send message $i+1$.

More complex is specifying the operations of the spy. Paulson's original definition allows (by the "Fake" rule) the spy to enrich any trace *evs* by the event

$$\mathsf{Says}\,\mathsf{Spy}\,B\,X$$

where $X$ can be any spoof message, and $B$ can be any agent: on any network configurations, the spy can send spoof messages to anybody. The spoof messages are drawn from the set $\mathsf{synth}(\mathsf{analz}(\mathsf{knows}\,\mathsf{Spy}\,evs))$ obtained by breaking down the known ciphers encrypted under known keys (by analz) and then building up new messages by concatenation and encryption (by synth).

If $C$ is a card that is either cracked or lost by a bad agent, then the spy is able to feed it with a spoof message $X$. I allow this to happen in the model by extending the Fake rule with the event

$$\mathsf{Inputs}\,\mathsf{Spy}\,C\,X$$

One rule is sufficient to model the spy sending inputs to all cards she can access. On the contrary, to let the spy obtain the outputs of the cards she can access, given her own inputs, a rule per each possible card output is needed. Let `r` be the protocol rule formalising the output provided by a card to its owner $P$ when the card is fed correctly. Since $P$ is able to use his card, the rule premises must include

$$\mathsf{Card}\,P \notin \mathsf{lost}$$

The card only gives outputs on demand, therefore amongst its premises `r` contains

$$\mathsf{Inputs}\,P\,(\mathsf{Card}\,P)\,X \in evs$$

where $X$ is the message expected by the card, and *evs* is the current trace. The rule `r` will conclude, for some suitable message $Y$, that *evs* can be extended by the event

$$\mathsf{Outputs}\,(\mathsf{Card}\;P)\,P\,Y$$

Thus, the model must be extended by a new rule `rF` obtained by replacing the condition that allows $P$ to use his card by the condition discussed above that allows the spy to use it:

$$(\mathsf{Card}\,P \in \mathsf{lost}\,\&\,P \in \mathsf{bad}) \mid \mathsf{Card}\,P \in \mathsf{cracked}$$

The I/O events must now mention the spy:

$$\mathsf{Inputs}\,\mathsf{Spy}\,(\mathsf{Card}\,P)\,X \qquad \mathsf{Outputs}\,(\mathsf{Card}\;P)\,\mathsf{Spy}\,Y$$

| | | | | | | |
|---|---|---|---|---|---|---|
| I : | 1. | $A$ | $\rightarrow$ | $\mathsf{S}$ | : | $A, B$ |
| | 2. | $\mathsf{S}$ | $\rightarrow$ | $A$ | : | $\Pi_{ab}, \{\!|\Pi_{ab}, B|\!\}_{Ka}$ |
| II : | 3. | $A$ | $\rightarrow$ | $C_a$ | : | $A$ |
| | 4. | $C_a$ | $\rightarrow$ | $A$ | : | $Na, \{\!|Na|\!\}_{K_{Ca}}$ |
| III : | 5. | $A$ | $\rightarrow$ | $B$ | : | $A, Na$ |
| IV : | 6. | $B$ | $\rightarrow$ | $C_b$ | : | $A, Na$ |
| | 7. | $C_b$ | $\rightarrow$ | $B$ | : | $Nb, Kab, \{\!|Na, Nb|\!\}_{\pi_{ab}}, \{\!|Nb|\!\}_{\pi_{ab}}$ |
| V : | 8. | $B$ | $\rightarrow$ | $A$ | : | $Nb, \{\!|Na, Nb|\!\}_{\pi_{ab}}$ |
| VI : | 9. | $A$ | $\rightarrow$ | $C_a$ | : | $B, Na, Nb, \Pi_{ab}, \{\!|\Pi_{ab}, B|\!\}_{Ka}, \{\!|Na, Nb|\!\}_{\pi_{ab}}, \{\!|Na|\!\}_{K_{Ca}}$ |
| | 10. | $C_a$ | $\rightarrow$ | $A$ | : | $Kab, \{\!|Nb|\!\}_{\pi_{ab}}$ |
| VII : | 11. | $A$ | $\rightarrow$ | $B$ | : | $\{\!|Nb|\!\}_{\pi_{ab}}$ |

**Figure 1: The Shoup-Rubin protocol**

# 4 Modelling the Shoup-Rubin protocol

In 1996, Shoup & Rubin proposed "*a method by which smart cards can be used to enhance the security of session key distribution*" [15]. They design a protocol based on smart cards, which extended a protocol formerly proposed by Leighton & Micali [10]. It is surprising that they only discuss the message exchange of their protocol within the Bellare-Rogaway formalism, which makes the gist of the protocol rather hard to grasp ([15], pages 325, 329). Not surprisingly, the protocol implementors later noticed that "*the details of Shoup-Rubin are fairly intricate, in part to satisfy the requirements of an underlying complexity-theoretic framework*" [9]. Despite this, the implementors still use some notation without providing much explanation for it. For instance, they mention messages such as $\{r \cdot 1 \cdot 1\}_{K_{AC}}$, where the two occurrences of the number 1 seem irrelevant to the protocol goals. Moreover, they do not mention whether their smart cards are PIN operated.

Fig. 1 presents my understanding of the protocol, which I had to extrapolate from the original theoretical presentation. The key $Ka$ denotes agent $A$'s long-term key, while $C_a$ represents her smart card. The card's long-term key is $K_{Ca}$. Encryption is symmetric, so each agent owns a long-term key that is shared with the trusted server $\mathsf{S}$. The protocol aims at distributing confidential session keys robustly thanks to the fact that the agents do not know their long-term keys, which are stored in the smart cards.

Leighton & Micali [10] introduced the concept of *pairkey*. The pairkey for agents $A$ and $B$ is $\Pi_{ab} = \{\!|A|\!\}_{Kb} \oplus \{\!|B|\!\}_{Ka}$. $A$'s card can compute $\{\!|B|\!\}_{Ka}$ and then $\pi_{ab} = \{\!|A|\!\}_{Kb}$ from $\Pi_{ab}$. $B$'s card can compute $\pi_{ab}$ directly, so that the two cards share $\pi_{ab}$ to communicate. The function $\oplus$ has several known weaknesses that I do not address here. Therefore, in the current model the spy is not able to obtain $\pi_{ab}$ from $\Pi_{ab}$.

In **phase I**, $A$ tells the trusted server to want to initiate a session with $B$, and receives in return the pairkey $\Pi_{ab}$ and its verifier encrypted under her long-term key. $A$ takes note of the pairkey and its verifier.

In **phase II**, $A$ invokes her card and receives a fresh nonce and its verifier encrypted under the card's long-term key. Neither the inventors nor the implementors state how $A$ activates her card, so my choice of message 3 is arbitrary. $A$ takes note of the nonce and its verifier.

In **phase III**, $A$ contacts $B$ sending her identity and her nonce $Na$.

In **phase IV**, $B$ activates his card by means of the data received from $A$, and obtains a new nonce $Nb$ used to construct the session key $Kab$, a verifier for $Na$ and $Nb$, and a verifier for $Nb$. $B$ takes note of the session key and of the verifier for $Nb$.

In **phase V**, $B$ forwards his nonce $Nb$ and the verifier for $Na$ and $Nb$ to $A$.

In **phase VI**, $A$ feeds her card with her peer's identity, the two nonces (she has just received $Nb$), the pairkey and its verifier, the two verifiers for the nonces; she obtains the session key $Kab$ that her card computes as a function of the nonce $Nb$, and the verifier for $Nb$. $A$ takes note of the session key for future use.

In **phase VII**, $A$ forwards the verifier for $Nb$ to $B$.

## 4.1 The model

Let `sr` be the formal protocol model; recall that `sr` is an inductively defined set of traces.

The base case of the definition states that the empty trace belongs to the model. In the sequel, I present the inductive steps that specify the operations of honest agents running the Shoup-Rubin protocol. Such steps are expressed in the form of rules that build the set `sr`.

First, I introduce a rule to allow those messages that have been sent to be received by the intended recipients.

```
Reception:
[| evsR ∈ sr; Says A B X ∈ evsR |]
⟹ Gets B X # evsR ∈ sr
```

Technically speaking, the rule states that if $A$ has sent $X$ to $B$ on a trace `evsR` of the protocol, then the trace obtained by concatenation of the event `Gets` $B$ $X$ with `evsR` also is an admissible trace of the protocol. This rule is present in all protocol models [3].

**Phase I**. At any point any agent $A$ may initiate a protocol session with the server, so `SR1` can fire on any trace, even on the empty one. If the server receives a message of the expected form, it replies to the first agent quoted by the message with the pairkey and its verifier. Since the pairkey is never used to seal ciphers but merely as a datum to activate the smart cards, I associate type `Number` to it [14]. The spy can see it because it is sent in clear.

```
SR1:
evs1 ∈ sr  ⟹ Says A Server {|Agent A, Agent B|}
                # evs1 ∈ sr

SR2:
[| evs2 ∈ sr;
   Gets Server {|Agent A, Agent B|} ∈ evs2 |]
⟹ Says Server A {|Number (Pairkey A B),
                  (Crypt (shrK A)
                  {|Number (Pairkey A B), Agent B|})
                 |} # evs2 ∈ sr
```

**Phase II**. If $A$ receives a message from the server, she takes note of the pairkey and its verifier, and queries her smart card. Note that $A$ recognises the recipients of the pairkey only from its verifier, which quotes them explicitly — `PairK` is a mere sequence of bits. If $A$ has not lost her card, then the card can issue her with a new nonce and its verifier.

```
SR3:
[| evs3 ∈ sr;
   Gets A {|Number PairK,
            (Crypt (shrK A) {|Number PairK, Agent B|})
           |} ∈ evs3 |]
⟹ Inputs A (Card A) (Agent A)
  # Notes A {|Number PairK,
              (Crypt (shrK A) {|Number PairK, Agent B|})
              |} # evs3 ∈ sr

SR4:
[| evs4 ∈ sr; Nonce Na ∉ used evs4; Card A ∉ lost;
   Inputs A (Card A) (Agent A) ∈ evs4 |]
⟹ Outputs (Card A) A {|Nonce Na,
                       (Crypt (crdK (Card A))
                       (Nonce Na))
                      |} # evs4 ∈ sr
```

**Phase III**. Upon reception of a nonce and a verifier, $A$ notes them and forwards them to $B$. Although the form of the verifier is unintelligible to $A$, $A$ can realise whether it is a cleartext or not, i.e. whether it is obtained as concatenation of any two messages `p` and `q`. This prevents the rule to fire in the following phase V, when it is the turn of `SR8` to fire (see below). The protocol implementation must allow such check.

```
SR5:
[| evs5 ∈ sr;
   Outputs (Card A) A {|Nonce Na, Verifier|} ∈ evs5;
   ∀ p q. Verifier ≠ MPair p q |]
⟹ Says A B {|Agent A, Nonce Na|}
  # Notes A {|Nonce Na, Verifier|} # evs5 ∈ sr
```

**Phase IV**. If $B$ has not lost his card, he feeds the message received from $A$ to the card and obtains a new nonce, a session key built from it, and two verifiers. If in the implementors' paper [9] one leaves out the notational details, then the session key seems to have the same form of the second verifier, (I am discussing this with Peter Honeyman). Therefore, for the moment I model the session key by the function `newSK`, which may be refined in the future. The function `makeK` simply serves as type-translation in order to use items of type message as encryption keys, which the protocol requires.

```
SR6:
[| evs6 ∈ sr;
   Gets B {|Agent A, Nonce Na|} ∈ evs6 |]
⟹ Inputs B (Card B) {|Agent A, Nonce Na|}
  # evs6 ∈ sr
```

```
SR7:
[| evs7 ∈ sr; Card B ∉ lost;
   Nonce Nb ∉ used evs7; Key (newSK Nb) ∉ used evs7;
   Inputs B (Card B) {|Agent A, Nonce Na|} ∈ evs7|]
⟹ Outputs (Card B) B {|Nonce Nb, Key (newSK Nb),
              (Crypt (makeK (Crypt (shrK B) (Agent A)))
                        {|Nonce Na, Nonce Nb|}),
              (Crypt (makeK (Crypt (shrK B) (Agent A)))
                        (Nonce Nb)) |} # evs7 ∈ sr
```

**Phase V**. When $B$ obtains a message from his card, he notes the session key and the second verifier, and forwards the nonce and the first verifier to $A$. Although $B$ is able to check the form of the verifiers since they are encrypted under $\pi_{ab}$, this is not needed here because the communication with the card is assumed to be reliable.

```
SR8:
[| evs8 ∈ sr;
   Outputs (Card B) B {|Nonce Nb, Key SesK,
                        Verif1, Verif2|} ∈ evs8 |]
⟹ Says B A {|Nonce Nb, Verif1|}
  # Notes B {|Key SesK, Verif2|} # evs8 ∈ sr
```

**Phase VI**. Upon reception of the message from $B$, provided that $A$ has previously taken note of the pairkey, of his nonce, and their verifiers, (i.e. she has been actively interacting with $B$ on the given trace), she can feed her smart card with these data and obtain, if the card is not lost, her copy of the session key built on $Nb$ and the verifier for $Nb$.

```
SR9:
[| evs9 ∈ sr;
   Notes A {|Number PairK,
             (Crypt (shrK A) {|Number PairK, Agent B|})
             |} ∈ evs9;
   Notes A {|Nonce Na, Verif1|} ∈ evs9;
   Gets A {|Nonce Nb, Verif2|} ∈ evs9 |]
⟹ Inputs A (Card A) {|Agent B, Nonce Na, Nonce Nb,
                       Number PairK,
                       (Crypt (shrK A)
                       {|Number PairK, Agent B|}),
                       Verif2, Verif1|} # evs9 ∈ sr

SR10:
[| evs10 ∈ sr; Card A ∉ lost;
   Inputs A (Card A) {|Agent B, Nonce Na, Nonce Nb,
                       Number (Pairkey A B),
                       (Crypt (shrK A)
                    {|Number (Pairkey A B), Agent B|}),
              (Crypt (makeK (Crypt (shrK B) (Agent A)))
                       {|Nonce Na, Nonce Nb|}),
              (Crypt (makeK (Crypt (shrK B) (Agent A)))
                       (Nonce Na))|}  ∈ evs10 |]
⟹ Outputs (Card A) A {|Key (newSK Nb),
              (Crypt (makeK (Crypt (shrK B) (Agent A)))
                        (Nonce Nb))|} # evs10 ∈ sr
```

**Phase VII**. Once $A$ obtains a message containing a key as first component, she realises that it is the last but one step of the protocol. Thus, she notes the key and forwards the second component to $B$.

```
SR11:
[| evs11 ∈ sr;
   Outputs (Card A) A {|Key SesK, Verifier|} ∈evs11 |]
⟹ Says A B (Verifier)
  # Notes A (Key SesK) # evs11 ∈ sr
```

**The spy's activity**. The inductive definition presented so far must be extended with the steps formalising the operations of the spy. As seen above (sec. 3.4), the spy sends all the messages she can forge, and also uses the lost cards that belonged to bad agents and the cracked ones.

```
Fake:
[| evs ∈ sr;  X ∈ synth (analz (knows Spy evs));
   Card A ∈ cracked | (Card A ∈ lost & A ∈ bad) |]
⟹ Says Spy B X
  # Inputs Spy (Card A) X # evs ∈ sr
```

During a protocol session, $A$'s card provides two outputs, one by rule SR4 and one by rule SR10, and $B$'s card provides one output by rule SR7. I must allow the spy to obtain those outputs, in case she could use $A$'s card. As explained in sec. 3.4, from SR4 I introduce the rule SR4F.

```
SR4F:
[| evs4F ∈ sr; Nonce Na ∉ used evs4F;
   Card A ∈ cracked | (Card A ∈ lost & A ∈ bad);
   Inputs Spy (Card A) (Agent A) ∈ evs4F |]
⟹ Outputs (Card A) Spy {|Nonce Na,
                         (Crypt (crdK (Card A))
                         (Nonce Na))|} # evs4F ∈ sr
```

Rules SR7F and SR10F are defined according to the same procedure.

## 4.2 Proving guarantees about the model

To substantiate the adherence of the proposed model to the real world, I present here some of the theorems I have proved about it. Each proof is down to three or four Isabelle commands: induction is first applied to the goal to prove, then Isabelle's powerful simplifier closes the proof. The runtime is less than five seconds per proof on a 300 Mhz Pentium.

In general, a possibility property establishes that there are traces containing the last message of the protocol. Proving this detects any gross errors in the model. With the Shoup-Rubin protocol, this property states that agents who have not lost their smart cards can successfully complete a protocol session.

**Theorem 1 (Possibility property)**

```
[| (Card A) ∉ lost; (Card B) ∉ lost |]
⟹ ∃ Verifier. ∃ evs ∈ sr. Says A B (Verifier) ∈ evs
```

The model allows honest agents to query only their own smart cards when they are not lost, and forces the agents to feed the cards suitably, often with some information received from previous network events. This is proved by the following theorems.

**Theorem 2 (A's card first input)**

```
[| evs ∈ sr; A ≠ Spy;
   Inputs A C (Agent A) ∈ evs |]
⟹ C = (Card A) & (Card A) ∉ lost &
    (∃ PairK Verifier.
       Gets A {|PairK, Verifier|} ∈ evs)
```

**Theorem 3 (B's card input)**

```
[| evs ∈ sr; B ≠ Spy;
   Inputs B C {|Agent A, Nonce Na|} ∈ evs |]
⟹ C = (Card B) & (Card B) ∉ lost &
    Gets B {|Agent A, Nonce Na|} ∈ evs
```

**Theorem 4 (A's card second input)**

```
[| evs ∈ sr; A ≠ Spy;
   Inputs A C {|Agent B, Nonce Na, Nonce Nb, PairK,
                Verif1, Verif2, Verif3|} ∈ evs |]
⟹ C = (Card A) & (Card A) ∉ lost &
    Notes A {|PairK, Verif1|} ∈ evs &
    Notes A {|Nonce Na, Verif3|} ∈ evs &
    Gets A {|Nonce Nb, Verif2|} ∈ evs
```

In other words, such guarantees state that honest agents must follow the steps of the protocol. Similar guarantees establish that the smart cards of honest agents must work properly, i.e. they give outputs only when they are queried with the correct inputs.

**Theorem 5 (A's card first output)**

```
[| evs ∈ sr;
   Outputs C A {|Nonce Na,
                 (Crypt (crdK (Card A)) (Nonce Na))|}
     ∈ evs |]
⟹ C = (Card A) & (Card A) ∉ lost &
    Inputs A (Card A) (Agent A) ∈ evs
```

**Theorem 6 (B's card output)**

```
[| evs ∈ sr; B ≠ Spy;
   Outputs C B {|Nonce Nb, Key SesK,
                 Verif1, Verif2|} ∈ evs |]
⟹ C = (Card B) & (Card B) ∉ lost &
    (∃ A Na.
       Inputs B (Card B) {|Agent A, Nonce Na|} ∈ evs)
```

**Theorem 7 (A's card second output)**

```
[| evs ∈ sr;  A ≠ Spy;
   Outputs C A {|Key SesK, Verifier|} ∈ evs |]
⟹ C = (Card A) & (Card A) ∉ lost &
    (∃ B Na Nb PairK Ver1 Ver2 Ver3.
       Inputs A (Card A) {|Agent B, Nonce Na, Nonce Nb,
                           PairK, Ver1, Ver2, Ver3|}
         ∈ evs)
```

Note that theorems 6 and 7 do apply only to agents other than the spy. Such assumption is needed to solve the cases arising from SR7F and SR10F respectively. On the contrary, theorem 5 does not require this. The explicit form of the verifier (the second component) of the Outputs event of theorem 5 is necessary to distinguish the premises from those of theorem 6. The explicit verifier also binds the agent $A$ receiving the output, which solves case SR4F because $A$ should be the spy and at the same time own a card either lost or cracked (absurd).

Therefore, although theorem 5 may seem stronger for relying on one assumption fewer, it needs a deeper assumption about the card output. Obviously, both theorems 6 and 7 could be modified in the same fashion.

I claimed that my model forces the spy to only use her own smart card, or a cracked one, or a lost one of which she knows the PIN. This is demonstrated by the following theorem.

**Theorem 8 (Spy's cards' inputs)**

```
[| evs ∈ sr; Inputs Spy C X ∈ evs |]
⟹ C = (Card Spy) |  C ∈ cracked |
    (∃ A. (C ∈ lost & C = Card A & A ∈ bad))
```

The same theorem holds if the Inputs event is replaced by an Outputs one.

# 5 Conclusion

Carrying out secure communication sessions over modern, insecure networks has become as hard as ever, especially with the growth of electronic commerce and of the capitals involved. In several cases an eavesdropper has been able to corrupt some network agents and obtain their long-term keys (e.g. [2]), which means acquiring the agents' entire knowledge. Smart cards have been introduced to store honest agents' secrets in order to increase the global robustness of the network. The need to use formal methods to verify whether this goal is met is unquestionable.

The theoretic approach of Bellare & Rogaway [5] is a fine piece of work. However, non-theoreticians have found it hard to get the intuition of their notion of provably secure protocol. Within their notion, Shoup & Rubin verify a novel protocol design, but the design seems forced to adapt to the framework. This raises

the risk of verifying a different design: "*We found that several modifications to the protocol were necessary to obtain our proof of security, even though it is not clear that without these modifications the protocol is insecure*" ([15], page 324). Not surprisingly, I had to put half of the effort to model the Shoup-Rubin protocol in the process of understanding the protocol. The implementors' paper [9] helped me greatly, although some details of the protocol are still unclear. I am currently discussing them with Peter Honeyman, one of the implementors.

I have developed a new approach by extending an existing one based on theorem proving [14]. In this case, it is the framework that scales up to the protocol design, as emphasised in section 4. The proposed approach contains realistic features such as the possibility of agents to conspire with the spy, and of smart cards to be lost to the spy or cracked.

I have built an inductive model for my understanding of the Shoup-Rubin protocol, and have provided several guarantees, in the form of theorems proved by Isabelle, that the model behaves as I claim it does. Although I did not aim at analysing the protocol thoroughly, I am currently designing a guarantee assessing that the protocol keeps the session keys confidential, provided that certain smart cards cannot be used by the spy. This would imply that the robustness of the protocol is entirely determined by the robustness of the smart cards, with obvious pros and cons.

I believe that the proposed treatment will be understandable to the entire smart card community, and hope that it will attract more and more man-power to the field. I can see no point in affording smart cards if the protocol design that will include them has not been verified formally.

### Acknowledgements.

Larry Paulson and James Margetson commented extensively on a draft of this paper. Peter Honeyman kindly explained to me some important details of the protocol implementation. Work supported by EPSRC grant GR/K57381 'Mechanizing Temporal Reasoning'.

## References

[1] M. Abadi, M. Burrows, C. Kaufman, B. Lampson. Authentication and Delegation with Smart-cards. DIGITAL Technical Report 67, California, 1990.

[2] R. Anderson. Making Smartcard Systems Robust. Proc. of *First Smart Card Research and Advanced Application Conference*, 1994.

[3] G. Bella. Message Reception in the Inductive Approach. Cambridge University, Computer Laboratory, *Technical Report No. 460*, 1999. `http://www.cl.cam.ac.uk/~gb221/crypt.html`

[4] G. Bella, L. C. Paulson. Kerberos Version IV: Inductive Analysis of the Secrecy Goals. Proc. of *Fifth European Symposium on Research in Computer Security*, Springer, LNCS 1485, 1998.

[5] M. Bellare, P. Rogaway. Provably Secure Session Key Distribution – The Three Party Case; Proc. of *27th ACM Symposium on Theory of Computing*, ACM Press, 1995.

[6] L. Blain, Y. Deswarte. A Smartcard Fault-tolerant Authentication Server. Proc. of *First Smart Card Research and Advanced Application Conference*, 1994.

[7] M. Burrows, M. Abadi, R. M. Needham. A logic of authentication. *Proceedings of the Royal Society of London,* 426:233-271, 1989.

[8] N. Itoi, P. Honeyman. Smartcard Integration with Kerberos V5. CITI Technical Report 98-7, University of Michigan, 1998.

[9] R. Jerdonek, P. Honeyman, K. Coffman, J. Rees, K. Wheeler. Implementation of a Provably Secure, Smartcard-based Key Distribution Protocol. Proc. of *Third Smart Card Research and Advanced Application Conference*, 1998.

[10] T. Leighton, S. Micali. Secret-key agreement without public-key cryptography. Proc. of *Advances in Cryptology — CRYPTO'93*, Springer, 1996.

[11] L. C. Paulson. *Isabelle: A Generic Theorem Prover.* Springer, 1994. LNCS 828.

[12] L. C. Paulson. Relations Between Secrets: Two Formal Analyses of the Yahalom Protocol. Cambridge University, Computer Laboratory, *Technical Report No. 432*, 1997.

[13] L. C. Paulson. Inductive Analysis of the Internet Protocol TLS. Cambridge University, Computer Laboratory, *Technical Report No. 440*, 1997.

[14] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85-128, 1998.

[15] V. Shoup, A. Rubin. Session Key Distribution using Smart Cards. Proc. of *Advances in Cryptology — EUROCRYPT'96*, LNCS1070, Springer, 1996.