Inductive Verification of Smart Card Protocols

Giampaolo Bella
Computer Laboratory, University of Cambridge
15 JJ Thomson Avenue, Cambridge CB3 0FD (UK)
giampaolo.bella@cl.cam.ac.uk

Dipartimento di Matematica e Informatica, Università di Catania
Viale A. Doria 6, I-95125 Catania (Italy)
giamp@dmi.unict.it

Abstract

An existing approach based on induction and theorem proving is tailored to the verification of security protocols that make use of smart cards. Smart cards are modelled operationally, hence only their functionalities, rather than their implementative technicalities, are of interest. The spy can steal certain smart cards, and clone others while learning their stored secrets. In terms of generality, the approach scales up to protocols that assume secure or insecure means between agents and smart cards, as well as to smart cards being PIN-operated or PIN-less. In terms of extensibility, new, application-dependent smart card functionalities can be easily included.

The approach is demonstrated on the key distribution protocol designed by Shoup and Rubin [30], and the assumptions are studied that are necessary on the smart cards for the protocol goals to be met. It is found that, if the data buses of the smart cards are unreliable as to produce outputs in an unspecified order, then the protocol does not confirm to the peers its goals of confidentiality, authentication, and key distribution because of lack of explicitness. A simple fix is introduced and proved.

1 Introduction

Safeguarding the long-term secrets to use in security protocol sessions was among the primary motives for the development of smart cards. Although some researchers believe that no hard device can ever be totally tamper-resistant, modern smart cards do offer a level of physical security that is considered sufficient for many applications. Today, a cheap *integrated-circuit memory card* hosts a few kilobytes of sensitive information. Additionally, an *integrated-circuit microprocessor card* embeds an 8-bit microprocessor that can perform relatively simple operations such as DES encryption/decryption. In consequence, smart card functionalities have been added either to existing security protocols (e.g. [15, 17]), or to newly designed ones (e.g. [30]). Below, we refer to those protocols that employ smart cards as *smart card protocols*, as opposed to *traditional protocols*, which do not.

Smart card protocols are intended to achieve stronger goals than traditional ones, but this should be confirmed formally. Somewhat to our surprise, the literature only features two significant attempts, which we detail in the next section, in that direction, whereas traditional protocols have been studied extensively by a large number of formal approaches (e.g. [13, 25, 26, 29]). The goal of our research was to develop an approach to analysing smart card protocols so that their strengths and weaknesses can be reasonably easily explained in detail with formal support. Achieving our goal would contribute to boosting E-commerce. We have pursued the goal [6, 8] by extending Paulson's Inductive Approach to analysing traditional protocols [26], and believe to have now reached it.

Proving deep properties is more complicated of smart card protocols than of traditional ones. A profound understanding of the roles played by the cards in the protocol sessions is necessary even to state putative properties. Then, proving those properties imposes a careful study of how the spy might exploit the cards. In our model, the smart cards are associated with a new type of the formal language. They can, as in reality, interact with their respective owners by receiving and sending messages. Each card stores a basic set of long-term secrets but additional ones, which may depend on the specific protocol, can be easily included. The cards are not forced to perform any computations, and may skip some or repeat others. The spy has stolen an unspecified set of cards but must discover their PINs, if they are PIN-operated, to be able to use them. She has cloned another set of cards, discovering their internal secrets. So, the spy can use the cloned cards freely even if they do not legally belong to her, while every other agent can only use his own card. Our treatment of the smart cards is entirely reusable for analysing future protocols.

Several smart card protocols make the *assumption of secure means*, signifying that the spy cannot interpose between the agents and their cards. So, on those channels, the messages can be exchanged in the clear, while each agent's knowledge of long-term secrets reduces to nothing if the cards are PIN-less, or to just the PIN to operate his card if the cards are PIN-operated. We account for all these alternatives by simple variations to the definition of spy's knowledge. For the sake of brevity, often in the following text *over secure means* replaces *when the assumption of secure means holds*, and *over insecure means* replaces *when the assumption of secure means does not hold*.

We demonstrate our approach on the key-distribution smart card protocol due to Shoup and Rubin [30], which makes the assumption of secure means and employs PIN-less cards. This protocol is an important benchmark for its negotiation of session keys, and because it can be implemented on commercially available smart cards (as done at the University of Michigan Center for Information Technology Integration [18]). The smart cards play the crucial role of computing the session keys and of making them available to their owners. While the session keys never traverse the network, some of the data used to compute them are exchanged in the clear. The protocol authors claim it provably secure.

Our contribution is an *inductive* study of the protocol goals of authenticity, unicity, confidentiality, authentication and key distribution. These are formulated as properties that are preserved by every protocol message. We also allow for the data buses of the cards to be unreliable in the sense that they may produce outputs in an unspecified order. We are not aware whether this was assumed to be impossible by the protocol designers. However, "often protocols are used in environments other than the ones for which they were originally intended, so it makes sense to analyze them under different security conditions than the ones they were originally designed for" [23]. We discover that, in these conditions, confirming to the peers that those goals are met necessitates of extra explicitness to two of the protocol messages. In the light of our findings, it seems fair to claim that computer-aided analysis of complete smart card protocols is now feasible.

This paper sets out with some background about existing approaches to analysing smart card protocols (§2). Our approach is then presented (§3) and the Shoup-Rubin protocol introduced (§4). Then, the protocol is modelled (§5) and verified (§6). Finally, an upgraded protocol that confirms its goals to the peers even with unreliable card buses is introduced and verified (§7). The paper ends with some concluding remarks (§8)

2 Background

Modern smart cards can run mobile Java code, thus contributing to cut the costs of applications such as pay-TV, mobile or public phones and credit cards. Forrester Research estimates that e-commerce will attract 40 million clients in the USA within the first three years of the new millennium [14]. Hence, smart card readers are bound to become inexpensive devices for home computers running smart card middleware. Nevertheless, large investments may be required, for example, for a bank to replace the totality of its magnetic cards by smart cards, as well as its traditional protocols by smart card ones. That bank will seek expert advice asking, for example, if an intruder will be able to acquire sensitive information after the discovery of some PINs or the cloning of some cards. Our approach tackles these delicate issues by advancing guarantees for the protocol goals and, most importantly, by asserting the minimal conditions necessary for those guarantees to hold.

Abadi et al. [1] develop a simple extension of the BAN logic [12] to model smart card protocols for mutual authentication between agents and workstations. The calculus of the logic is used to prove the mutual authentication and delegation goals of three protocols that require different computational resources of the cards. Confidentiality issues are not considered, as the early belief logics are notoriously inadequate for the purpose. Although their treatment is limited to the specific goals of the protocols considered, we believe that it could be broadened thanks to the large experience accumulated during the last decade in the area of belief logics.

Bellare and Rogaway's fine notion of *provable security* [10] characterises in terms of probability and complexity theory that a traditional protocol achieves confidentiality of session keys. They design a protocol and show it provably secure under the assumption that pseudo-random functions exist [16]. Later, Shoup and Rubin employ the same approach on a traditional session-key distribution protocol due to Leighton and Micali [20] but state: "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." [30, §2.2]. Modifying a protocol for the sake of its formal verification raises the risk of verifying a different protocol.

Shoup and Rubin also design a new protocol, based on that by Leighton and Micali, for session key distribution in a three-agent setting where each agent is endowed with an ideal smart card whose encryption function is a pseudo-random function. (The authors also mention that pseudo-random functions can be implemented with reasonable approximation in terms of efficient hash functions). Finally, they extend Bellare and Rogaway's approach to account for smart cards, and argue that the new protocol is secure in terms of two properties. First, the peers share a session key at the end of a protocol session in which the spy does not prevent the delivery of messages. This property, which may not be obvious to readers unfamiliar with the formalism, is not proved. Second, the spy has a "negligible advantage". This property, concluded by mathematical proof, signifies that the spy only has a negligible probability of learning the session key.

Our approach appears to be little related to those mentioned above. Although it models the network operationally, it implicitly asserts the beliefs that agents derive from the observation of certain network events. There is no complexity theory involved, for encryption as well as pseudo-random functions are merely assumed to be injective functions. Moreover, the only proof strategy adopted is induction, and a large variety of formal guarantees, including confidentiality, can be established. In particular, while no modifications are required of the protocol design, we can also establish whether the protocol goals are *available* to the peers [7], [29, §9.9], namely whether the goals rely on conditions that the peers are able to verify.

There is a constant tension between formal approaches attempting to be as general as possible, and the extensions that become necessary on them due to their upcoming applications. For example, when treating traditional symmetric-key protocols, Paulson formalises the three-party case, hence his approach requires a few extensions to deal with additional features such as the double trusted server of Kerberos IV [9]. Our approach for smart card protocols shall have similar fate because of the large variety of real-world contexts in which smart cards may be used. For example, a card for key distribution protocols typically stores the card owner's long-term key, while a card for pay-TV protocols stores the remaining credit of the card owner. Nevertheless, the basic set of card secrets that we define below is easily extendible.

3 An Inductive Approach to Analysing Smart Card Protocols

Security protocols must face a number of real-world threats, some of which may be difficult to account for, or just unknown. Therefore, even when a protocol has been "proved" to enforce certain goals, a skeptical protocol purchaser might still suspect that not all realistic scenarios have been considered under the given threats, or that certain, possibly crucial, threats have been omitted.

Paulson proposes the use of mathematical induction to dissipate the first suspect, and develops an approach that appears to be sufficiently flexible to tackle the second suspect [28]. The gist of the approach is essentially as follows. Once the possible threats, such as the spoof cipher-texts built by the spy, have been identified, they are modelled as an inductively defined set. The protocol goals are then proved by induction over the whole set. Should a purchaser point out novel threats, these could be modelled as additional inductive rules, and the proofs reconsidered.

More precisely, given a traditional protocol $\mathcal{P}$, Paulson takes the following view. An unlimited population of agents can run $\mathcal{P}$. Among the agents is the spy, who monitors the entire network and knows the long-term secrets of an unspecified set of compromised agents. The network traffic develops according to the decisions taken by the agents while they are executing $\mathcal{P}$, each interleaving an unlimited number of protocol sessions. A history of the network traffic may be represented by the list of the events occurred during that history, that is a *trace*. As in Ryan and Schneider's CSP work [29], each trace has unlimited length. The set P of all possible traces is an operational model for the network where $\mathcal{P}$ is executed. Generally, P is referred to as the *formal protocol model for* $\mathcal{P}$. We adopt the same view, and extend it for smart card protocols.

3.1 Smart cards

We aim at representing the operational aspects of smart cards, so we introduce a new free type card with several associated functions, and abstract from the hardware/software of the cards. Clearly, other models are feasible of smart cards. The following shows that our model is simple though realistic enough for an abstract verification.

To endow each agent with a smart card, we declare an injective function between agents and smart cards

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

Cards interact with their owners by sending them outputs and receiving inputs from them, as we formalise later (§3.2) in terms of specific events. We assume that (the CPUs of) the cards only provide correct but limited outputs. Precisely, a card produces a correct output only if fed the corresponding input. If a card can compute, for example, a session key K from an input X, the card must necessarily be fed X in order to obtain K. The formal protocol model can easily account for this (§5). It shall only allow for the outputs encompassed by the protocol, under the condition that the cards are fed the corresponding, specific inputs. It shall not construct other outputs, even from cloned cards. In consequence, there exists no card whose use can give the spy unlimited power. So, card outputs are assumed correct in the sense that their forms and components are as they should be. But the data buses of the cards cannot be trusted, as explained below.

3.1.1 Card Vulnerabilities

We formalise a number of realistic card vulnerabilities, which are due to theft, cloning and internal failures.

Theft. The small dimensions of the smart cards confer their portability but also raise the risks of loss or theft. In the worst case, all cards that have been lost by their owners or stolen from them will end up in the spy's hands. These cards, which can no longer be used by their owners, are modelled by the set stolen, such that $\mathsf{stolen} \subseteq \mathsf{card}$.

Cloning. If the cards are PIN-operated, the spy cannot use a stolen card actively unless she knows its PIN. Nevertheless, she could use modern techniques, such as *microprobing* [3], to break the physical security of the card, and access its EEPROM (Electrically-Erasable Programmable Read-Only Memory) where the long-term secrets are stored. Now, she could, in the worst case, reverse engineer the whole circuitry, acquiring the ability to to build a clone of the card for her own use. If so, the card belongs to the set cloned in the model, and $\mathsf{cloned} \subseteq \mathsf{card}$.

Cloning without apparent theft. All cloning techniques that are currently known are *invasive*, in the sense that they spoil the original card. The card chip must be disembedded from its frame by suitable chemicals, and its layout often modified using laser cutter microscopes. These alterations are irreversible. However, the spy might steal a card, build two clones of it and return one to the card owner, who would perhaps notice no irregularities. In the near future, the spy might even be able to tailor *non-invasive* techniques (such as *fault generation* [19, 22] by exploiting the power and clock supply lines) for cloning, and return the original card to its owner after building a clone for herself. Modelling these possibilities simply requires stating no relations between the sets stolen and cloned, so that a card could be cloned and yet not be stolen. In this scenario, the real world features two cards — the original and its clone — but the model for simplicity allows the same card to be used both by its legal owner and by the spy, granting them identical computational resources.

Data bus failure. The data buses of the cards are corrupted so that the messages in transit can be either forgotten (due to electronic decay), permuted or fed to the CPU repeatedly (due to simple layout modifications). Therefore, a smart card can omit some computations or repeat others. We deliberately forbid message leakage or alteration at this level because they would clearly make it impossible to formalise the assumption of secure means — this is consistent with the previous assumption that card outputs are correct.

In the worst case, the spy has manipulated all cards in this fashion even before they are delivered to their respective owners, leaving no visible trace of tampering. To model this, the formal protocol model shall only allow events to occur by firing of inductive rules, but rules shall not be forced to fire even when their preconditions were met. Also, rules shall be enabled to fire in any order and to fire more than once, and each of these possibilities shall be recorded by a corresponding trace.

Because cards are deteriorated, an agent cannot feed an input m to his card, and deduce that the first output m' obtained immediately afterwards was computed out of m, unless m' includes all the necessary message components to pinpoint m. Hence, our model cards are particularly appropriate to discover possible lack of explicitness in the protocol messages.

Global failure. Smart cards may suffer unexpected failure at some point, and stop working permanently. The formal protocol model accounts for this by including traces that, after some event, no more involve those cards.

There are also traces that, along some finite fragments do not feature certain cards. Such traces may be interpreted as modelling temporary global failures of those cards.

3.1.2 Card Usability

Agents other than the spy only conduct legal operations, while the spy can act both legally and illegally. A card that has not been stolen can be used only by its owner, namely it can be used legally. The spy cannot use a non-stolen card unless it is her own. Definition 1 captures the notion of *legally usable card*. This clearly is independent from the assumption of secure means, and from the existence of card PINs. (If cards are PIN-operated, then we shall simply guarantee (§3.3) that each agent knows the PIN to activate her card).

Definition 1. $\mathsf{legallyU}(\mathsf{Card}\,A) \ \triangleq\ (\mathsf{Card}\,A) \notin \mathsf{stolen}.$

Over insecure means, the spy can listen in between agents and smart cards. So, she has electronic access only to those cards of which she knows the PINs, even without physically getting hold of them. This is due to the fact that a PIN-operated card accepts no communication unless it is activated by means of its PIN. PINs are transmitted on the means between agents and cards (never on the network), so the spy might learn some of them on certain traces. On the other hand, should the cards be PIN-less, the spy could use all of them on any trace. Definition 2 expresses the notion of *illegally usable card over insecure means.* The predicate *the spy knows A's PIN on trace evs* will be refined below using the formal definition of agents' knowledge (§3.3).

Definition 2. Over insecure means

$$\mathsf{illegallyU}(\mathsf{Card}\,A)\ \mathsf{on}\ evs \ \triangleq \begin{cases} \textit{the spy knows } A\textit{'s PIN on trace } evs \\ \qquad\qquad \text{if cards are PIN-operated} \\ \mathsf{true} \\ \qquad\qquad \text{if cards are PIN-less} \end{cases}$$

Over secure means, the spy needs to gain physical access to the cards in addition to the knowledge of their PINs. She cannot monitor the means between agents and cards, and cannot discover PINs from the network events because PINs never traverse the network. The spy only has the chance of knowing them initially (§3.3), hence the definition of illegal usability needs not depend on traces. If the cards are PIN-less, we only need to characterise the physical access to the card. Definition 3 conveys the notion of *illegally usable card over secure means*. The predicate *the spy knows A's PIN* will be refined below.

Definition 3. Over secure means

$$\mathsf{illegallyU}(\mathsf{Card}\,A) \ \triangleq \begin{cases} (\mathsf{Card}\,A) \in \mathsf{cloned} \ \vee \ ((\mathsf{Card}\,A) \in \mathsf{stolen} \ \wedge \\ \qquad\qquad\qquad\qquad \textit{the spy knows } A\textit{'s PIN}) \\ \qquad\qquad\qquad\qquad \text{if cards are PIN-operated} \\ (\mathsf{Card}\,A) \in \mathsf{cloned} \ \vee \ (\mathsf{Card}\,A) \in \mathsf{stolen} \\ \qquad\qquad\qquad\qquad \text{if cards are PIN-less} \end{cases}$$

The spy must be given the opportunity to act legally, so we must allow her too to use her own card legally. But she does not need to use her card illegally because she cannot acquire additional knowledge from it. So, we state $\mathsf{Card\ Spy} \notin \mathsf{stolen} \cup \mathsf{cloned}$. The same can be stated in the three-party case about the card that belongs to the trusted server.

We emphasise that, since certain cards may be cloned and at the same time not be stolen, there may exist cards that are both legally and illegally usable.

3.1.3 Card Secrets

A smart card typically contains two long-term symmetric keys: the PIN to activate its functionalities, and the card key. So, we declare

$$\mathsf{pinK} : \mathsf{agent} \longrightarrow \mathsf{key} \qquad\qquad \mathsf{crdK} : \mathsf{card} \longrightarrow \mathsf{key}$$

Since PINs are both known to agents and stored in the cards, the first function could be equivalently declared on cards. Recall that a card key is used to save on the RAM of the card, for example, as follows. Each card issues a nonce along with a copy that is encrypted under the card key. This pair can later authenticate the nonce to the card, even if the card did not store the nonce.

In the case of key distribution protocols, each card also stores its owner's long-term key, which, in contrast with traditional protocols, is not known to the agent. Nevertheless, we keep the original declaration for these keys [26, §3.5]

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

Since the model is operational, the fact that the smart cards *store* some secrets does not need to be formalised explicitly. We will simply define below (§3.3) who, and in which circumstances, learns the secrets of a card. This increases the flexibility of the approach. Should the smart cards store additional secrets in certain applications, once such secrets are formalised by suitable functions, only the definition of agents' knowledge shall be updated.

We assume that collision of keys is impossible, so all functions declared above are injective and their ranges are disjoint.

3.2 Events

We upgrade the Isabelle datatype for events with four new constructors.

$$
\begin{array}{lclllll}
\mathsf{datatype\ event} & \triangleq & \mathsf{Says} & \mathsf{agent} & \mathsf{agent} & \mathsf{msg} \\
 & & \mathsf{Notes} & \mathsf{agent} & & \mathsf{msg} \\
 & & \mathsf{Gets} & \mathsf{agent} & & \mathsf{msg} \\
 & & \mathsf{Inputs} & \mathsf{agent} & \mathsf{card} & \mathsf{msg} \\
 & & \mathsf{CGets} & \mathsf{card} & & \mathsf{msg} \\
 & & \mathsf{Outputs} & \mathsf{card} & \mathsf{agent} & \mathsf{msg} \\
 & & \mathsf{AGets} & \mathsf{agent} & & \mathsf{msg}
\end{array}
$$

The known *network events* (sending, noting [26] and receiving a message [5] — the first three above) are now extended with the novel *card events*. Agents may send inputs to the cards (Inputs) and the cards may receive them (CGets); similarly, the cards may send outputs to the agents (Outputs) and the agents may receive them (AGets). An agent can distinguish the messages received from the network from those received from his smart card reader because they arrive on separate channels, so we provide two different events. However, in both cases the received messages may be forged by the spy.

The events CGets and AGets can be omitted over secure means, where a card certainly receives its owner's inputs, and an agent certainly receives the outputs of his card. Consequently, a smart card C can verify whether an event $\mathsf{Inputs}\,A\,C\,X$ occurred, and an agent A can check whether an event $\mathsf{Outputs}\,C\,A\,X$ occurred, while both checks are impossible over insecure means.

Paulson formalises the notion of freshness via the function

$$\mathsf{used} : \mathsf{event\ list} \longrightarrow \mathsf{msg\ set}$$

which yields the components of the initial states of all agents, and the components of all messages that appear on the given trace. We quote here the original definition (cases 0 to 2 [26, §3.5] and case 3 [4, §3]) and extend it to cope with the new events.

0. $\mathsf{used}\,[\,] \;\triangleq\; \bigcup B.\, \mathsf{parts}(\mathsf{initState}\,B\,)$

1. $\mathsf{used}((\mathsf{Says}\,A\,B\,X)\ \#\ evs) \triangleq \mathsf{parts}\{X\} \cup \mathsf{used}\ evs$
2. $\mathsf{used}((\mathsf{Notes}\,A\,X)\ \#\ evs) \triangleq \mathsf{parts}\{X\} \cup \mathsf{used}\ evs$
3. $\mathsf{used}((\mathsf{Gets}\,A\,X)\ \#\ evs) \triangleq \mathsf{used}\ evs$
4. $\mathsf{used}((\mathsf{Inputs}\,A\,C\,X)\ \#\ evs) \triangleq \mathsf{parts}\{X\} \cup \mathsf{used}\ evs$
5. $\mathsf{used}((\mathsf{CGets}\,C\,X)\ \#\ evs) \triangleq \mathsf{used}\ evs$
6. $\mathsf{used}((\mathsf{Outputs}\,C\,A\,X)\ \#\ evs) \triangleq \mathsf{parts}\{X\} \cup \mathsf{used}\ evs$
7. $\mathsf{used}((\mathsf{AGets}\,A\,X)\ \#\ evs) \triangleq \mathsf{used}\ evs$

Function parts extracts all components but encryption keys from a set of messages assuming readable cipher-texts [26, §3.2], while an updated definition of function initState is given in the next section. Notice that, since any received message must have been sent (§3.5), the reception events do not extend the inductive set. Cases 5 and 7 are omitted over secure means.

3.3 Agents' Knowledge

Recall that three different kinds of agents are modelled [26, §3.1]: the trusted third party, Server (often abbreviated in S); a malicious eavesdropper, Spy; unlimited "friendly" agents, Friend i (i being a natural number). Agents could be interpreted as humans, machines, or processes, as Abadi and Needham point out [2, §2]. We prefer the last interpretation. For example, if a confidentiality guarantee is available to a process, then it is not necessarily available to the human that owns the process because the spy could break in at any level.

The function initState formalising the agents' initial knowledge [26, §3.5] must be redefined to account for the secrets stored in the smart cards.

The initial knowledge of the server consists of all long-term secrets.

$$\mathsf{initState}\,\mathsf{S} \triangleq \{\mathsf{Key}\,(\mathsf{pinK}\,A)\} \cup \{\mathsf{Key}\,(\mathsf{crdK}\,C)\} \cup \{\mathsf{Key}\,(\mathsf{shrK}\,A)\}$$

Friendly agents' initial knowledge consists of their respective PINs.

$$\mathsf{initState}\,(\mathsf{Friend}\,i) \triangleq \{\mathsf{Key}\,(\mathsf{pinK}\,(\mathsf{Friend}\,i))\}$$

The spy's initial knowledge consists of the compromised agents' initial knowledge and the secrets contained in the cloned cards (even if some cards store the secrets in a blinded or an encrypted form, the spy may discover them in the worst case).

$$\begin{aligned}\mathsf{initState}\,\mathsf{Spy} \triangleq\ & \{\mathsf{Key}\,(\mathsf{pinK}\,A) \mid A \in \mathsf{bad}\ \vee\ (\mathsf{Card}\ \ A) \in \mathsf{cloned}\} \cup \\ & \{\mathsf{Key}\,(\mathsf{crdK}\,C) \mid C \in \mathsf{cloned}\} \cup \\ & \{\mathsf{Key}\,(\mathsf{shrK}\,A) \mid (\mathsf{Card}\ \ A) \in \mathsf{cloned}\}\end{aligned}$$

This definition considers cards that are PIN-operated, otherwise it simplifies straightforwardly. However, the definition is not influenced by the assumption of secure means because it formalises the situation before any protocol session has initiated.

The knowledge that agents can extract from the observation of the traffic on a given trace can now be formalised. We declare

$$\mathsf{knows} \ : \ [\,\mathsf{agent}, \mathsf{event\ list}\,] \longrightarrow \mathsf{msg\ set}$$

and define it inductively on the length of the trace. The definition, as one would expect, depends on the assumption of secure means. First, we reason over insecure means.

The base case and those concerning the network events (0 to 3 below) have been previously published [4, §3.1], while the remaining are new.

0. An agent knows his initial state.

   $\mathsf{knows}\, A\, [\,] \ \triangleq \ \mathsf{initState}\, A$

1. An agent knows what he alone sends to anyone on a trace; in particular, the spy also knows all messages ever sent on the trace.

   $\mathsf{knows}\, A\, ((\mathsf{Says}\, A'\, B\, X) \# \mathit{evs}) \ \triangleq$

   $$\begin{cases} \{X\} \cup \mathsf{knows}\, A\ \mathit{evs} & \text{if } A = A' \ \vee \ A = \mathsf{Spy} \\ \mathsf{knows}\, A\ \mathit{evs} & \text{otherwise} \end{cases}$$

2. An agent knows what he alone notes on a trace; in particular, the spy also knows compromised agents' notes.

   $\mathsf{knows}\, A\, ((\mathsf{Notes}\, A'\, X) \# \mathit{evs}) \ \triangleq$

   $$\begin{cases} \{X\} \cup \mathsf{knows}\, A\ \mathit{evs} & \text{if } A = A' \ \vee \\ & \quad (A = \mathsf{Spy} \ \wedge \ A' \in \mathsf{bad}) \\ \mathsf{knows}\, A\ \mathit{evs} & \text{otherwise} \end{cases}$$

3. An agent, except the spy, knows what he alone receives on a trace. The spy's knowledge must not be extended with any of the received messages, as the formal protocol model (§3.5) only allows reception of those messages that were sent, so the spy already knows them (by case 1).

   $\mathsf{knows}\, A\, ((\mathsf{Gets}\, A'\, X) \# \mathit{evs}) \ \triangleq$

   $$\begin{cases} \{X\} \cup \mathsf{knows}\, A\ \mathit{evs} & \text{if } A = A' \ \wedge \ A \neq \mathsf{Spy} \\ \mathsf{knows}\, A\ \mathit{evs} & \text{otherwise} \end{cases}$$

4. An agent knows what he alone inputs to any card on a trace; in particular, the spy also knows all messages ever input on the trace.

   $\mathsf{knows}\, A\, ((\mathsf{Inputs}\, A'\, C\, X) \# \mathit{evs}) \ \triangleq$

   $$\begin{cases} \{X\} \cup \mathsf{knows}\, A\ \mathit{evs} & \text{if } A = A' \ \vee \ A = \mathsf{Spy} \\ \mathsf{knows}\, A\ \mathit{evs} & \text{otherwise} \end{cases}$$

5. No agent, including the spy, can extend his knowledge with any of the messages received by any smart card on a trace. The formal protocol model insists that a card may only receive those messages that were previously

input, so the spy and the message originator already know them (by case 4).

$\mathsf{knows}\, A\, ((\mathsf{CGets}\, C\, X)\ \#\ \mathit{evs}) \ \triangleq\ \mathsf{knows}\, A\ \mathit{evs}$

6. An agent knows no card outputs on any trace, as the means is insecure; the spy knows all of them, as she controls the means.

   $$\mathsf{knows}\, A\, ((\mathsf{Outputs}\, C\, A'\, X)\ \#\ \mathit{evs}) \ \triangleq \begin{cases} \{X\} \cup \mathsf{knows}\, A\ \mathit{evs} & \text{if } A = \mathsf{Spy} \\ \mathsf{knows}\, A\ \mathit{evs} & \text{otherwise} \end{cases}$$

7. An agent, except the spy, knows what he alone receives from his card on a trace. The spy's knowledge must not be extended with any of the messages received from the smart cards, as the formal protocol model only allows reception of those messages that were output, so the spy already knows them (by case 6).

   $$\mathsf{knows}\, A\, ((\mathsf{AGets}\, A'\, X)\ \#\ \mathit{evs}) \ \triangleq \begin{cases} \{X\} \cup \mathsf{knows}\, A\ \mathit{evs} & \text{if } A = A' \ \wedge\ A \neq \mathsf{Spy} \\ \mathsf{knows}\, A\ \mathit{evs} & \text{otherwise} \end{cases}$$

Definition 2 can be now refined as definition $2'$. Recall that the function analz extracts all message components from a set of messages using keys that are recursively available [26, §3.2].

Definition $2'$. Over insecure means

$$\mathsf{illegallyU}(\mathsf{Card}\, A)\ \mathsf{on}\ \mathit{evs} \ \triangleq \begin{cases} \mathsf{Key}\,(\mathsf{pinK}\, A) \in \mathsf{analz}(\mathsf{knows}\, \mathsf{Spy}\ \mathit{evs}) \\ \qquad\qquad \text{if cards are PIN-operated} \\ \mathsf{true} \\ \qquad\qquad \text{if cards are PIN-less} \end{cases}$$

Our definition remarks that the illegal usability of a card over insecure means does not necessarily imply the spy's physical access to the card.

Over secure means, the definition of knows simplifies as follows. The base case and those corresponding to the network events remain unchanged. Cases (5) and (7) must be pruned, for the corresponding events are no longer defined. If an agent sends an input to his card, or the card sends him back an output, both messages are certainly received because the spy cannot intercept them. Hence, cases (4) and (6) must be amended accordingly.

$4'$. An agent, including the spy, knows what he alone inputs to any card on a trace.

$$\mathsf{knows}\, A\, ((\mathsf{Inputs}\, A'\, C\, X)\ \#\ \mathit{evs}) \ \triangleq \begin{cases} \{X\} \cup \mathsf{knows}\, A\ \mathit{evs} & \text{if } A = A' \\ \mathsf{knows}\, A\ \mathit{evs} & \text{otherwise} \end{cases}$$

$6'$. An agent, including the spy, knows what he alone is output from any card on a trace.

$$\mathsf{knows}\, A\, ((\mathsf{Outputs}\, C\, A'\, X) \# \mathit{evs}) \triangleq \begin{cases} \{X\} \cup \mathsf{knows}\, A\ \mathit{evs} & \text{if } A = A' \\ \mathsf{knows}\, A\ \mathit{evs} & \text{otherwise} \end{cases}$$

As it is required, these cases forbid the spy from learning anything from the card events. Therefore, she knows a PIN if and only if she knows it initially, which is equivalent, by definition of initState and base case of knows, to the fact that the owner of the PIN is compromised. Hence, definition 3 can be refined as definition 3′.

Definition 3′. Over secure means

$$\mathsf{illegallyU}(\mathsf{Card}\, A) \triangleq \begin{cases} (\mathsf{Card}\, A) \in \mathsf{cloned} \ \vee \ ((\mathsf{Card}\, A) \in \mathsf{stolen} \ \wedge \ A \in \mathsf{bad}) \\ \qquad\qquad\qquad\qquad \text{if cards are PIN-operated} \\ (\mathsf{Card}\, A) \in \mathsf{cloned} \ \vee \ (\mathsf{Card}\, A) \in \mathsf{stolen} \\ \qquad\qquad\qquad\qquad \text{if cards are PIN-less} \end{cases}$$

The definition emphasises the spy's physical access to the illegally usable cards over secure means. Moreover, only when the cards are PIN-less over secure means, does it hold that if a card is not illegally usable, then it is legally usable. This does not hold in general, nor does the converse.

The function knows extends and replaces Paulson's function spies, which only specified the spy's knowledge. Our function could be extended on smart cards, but reasoning about the knowledge of the cards seems rather uninteresting due to their limited RAM.

3.4 Spy's Illegal Behaviour

The spy's illegal behaviour with traditional protocols is typically specified by a single inductive rule, named *Fake*, that is added to the protocol model. (Only the model of the TLS protocol [27] contains a second rule that allows the spy to construct a session key by a public algorithm). Figure 1 provides the rule template for a hypothetical protocol model trad_p [4, §3.2]. The rule allows

```
Fake
[| evsF ∈ trad_p; X ∈ synth(analz(knows Spy evsF)) |]
⟹ Says Spy B X # evsF ∈ trad_p
```

Figure 1: *Fake* rule for traditional protocols

the spy to send any-one a message that she can synthesise (by encryption and concatenation, via the function synth [26, §3.2]) out of the components extracted (via the function analz) from her knowledge over the trace *evs*. This may induce the other agents to send new messages, which the spy will learn by the inductive case (1) of the definition of knows.

In modelling smart card protocols, the spy must be allowed to exploit the illegally usable smart cards. Over insecure means, not only can the spy send fake messages as inputs to the illegally usable cards, but she can also send fake outputs to any agents, pretending that her own card could produce them. This is done in addition to sending the fake messages on the network because receiving the same message from the card reader or from the network may induce an agent to react differently. The *Fake* rule must be amended as outlined in figure 2. Notice the condition of illegal usability over insecure means stated on A's card, and the three events that at the same time extend the given trace.

```
Fake
[| evsF ∈ smart_p_insecure_m; illegallyU(Card A) on evsF;
   X ∈ synth (analz (knows Spy evsF)) |]
⟹ Says Spy B X # Inputs Spy (Card A) X # Outputs (Card Spy) C X
      # evsF ∈ smart_p_insecure_m
```

Figure 2: *Fake* rule for smart card protocols over insecure means

Over secure means, the spy cannot send fake card outputs to the agents. Figure 3 presents the *Fake* rule to use in this case rather than those above. Notice the condition of illegal usability over secure means stated on A's card, and the two events that at the same time extend the given trace.

```
Fake
[| evsF ∈ smart_p_secure_m; illegallyU(Card A);
   X ∈ synth (analz (knows Spy evsF)) |]
⟹ Says Spy B X # Inputs Spy (Card A) X
      # evsF ∈ smart_p_secure_m
```

Figure 3: *Fake* rule for smart card protocols over secure means

In this scenario, by definition of knows, the spy gains no knowledge from the card events that do not concern her. Therefore, we must ensure that an illegally usable card outputs towards the spy rather than towards its owner. This is realistic because, over secure means, the spy has physical access to the illegally usable cards. Suppose that A's card outputs X' when it is fed X. The formal protocol model shall contain two rules according to the template in figure 4. Rule *Name* formalises part of the legal behaviour of an agent, so it involves a card that is legally usable. Should the card happen to be illegally usable, its functionality could be exploited by the spy, as formalised by rule *Name_Fake*. In this case, the spy would also need to fake the specific input X. Any extra assumptions required in *Name* must be kept in *Name_Fake*. Notice that agent A in rule *Name* may be the spy. This would signify that the spy is using her own card legally. But agent A in rule *Name_Fake* cannot be the spy (because her card is not illegally usable). This remarks that the spy is illegally using someone else's card. Rule *Name_Fake* is unnecessary over insecure means, where the spy monitors all card events.

```
Name
[| evsN ∈ smart_p_secure_m; legallyU(Card A);
   Inputs A (Card A) X ∈ set evsN |]
⟹ Outputs (Card A) A X' # evsN ∈ smart_p_secure_m

Name_Fake
[| evsNF ∈ smart_p_secure_m; illegallyU(Card A);
   Inputs Spy (Card A) X ∈ set evsNF |]
⟹ Outputs (Card A) Spy X' # evsNF ∈ smart_p_secure_m
```

Figure 4: The two rules for each card output over secure means

It may be surprising that whether the cards are PIN-operated or PIN-less has played no role in this subsection. This is in fact only apparent, as that feature influences the definitions of illegal usability of cards.

3.5 Formal Protocol Model

The inductive definition of the formal model for smart card protocols requires additional rules only over insecure means. Smart cards must be allowed to receive the inputs that they were sent from agents and, likewise, agents must be allowed to receive the outputs sent from cards. For these purposes, figure 5 introduces rules *CReception* and *AReception*, inspired to our rule *Reception* (omitted here but demonstrated below, §5.1) for reception of messages sent over the network ([4, Appendix]). Since the rules are not forced to fire, no

```
CReception
[| evsRc ∈ smart_p_insecure_m; Inputs A (Card B) X ∈ set evsRc |]
⟹ CGets (Card B) X # evsRc ∈ smart_p_insecure_m

AReception
[| evsRa ∈ smart_p_insecure_m; Outputs (Card A) B X ∈ set evsRa |]
⟹ AGets B X # evsRa ∈ smart_p_insecure_m
```

Figure 5: The additional reception rules over insecure means

kind of reception (either on the network or on the agent-smart-card means) is guaranteed because the spy controls all means and may prevent deliveries. By contrast, these rules are not needed over secure means where reception is guaranteed from agents to smart cards and vice versa.

4 The Shoup-Rubin Protocol

As mentioned above, among Shoup and Rubin's contributions is the design of a smart card protocol for session-key distribution, which is discussed within an extension of the Bellare-Rogaway's framework [30].

The gist of the protocol may not be easy to grasp, as noted by the protocol implementors: "the details of Shoup-Rubin are fairly intricate, in part to satisfy the requirements of an underlying complexity-theoretic framework" [18, §1]. Figure 6 presents the protocol we derived from both the designers and the implementors' presentations. We denote an agent P's long-term key (shared with the server) by Kp, P's smart card by C_p, and the key of P's card by K_{Cp}. The protocol makes the assumption of secure means — hence the cards output the session keys in the clear — and employs smart cards that are PIN-less. (Although these features are not explicitly stated either in the designers or the implementors' papers, Peter Honeyman — one of the implementors — kindly clarified them during a private conversation).

The concept of *pairkey* (due to Leighton and Micali [20]) is used to establish a long-term secret between the smart cards of a pair of agents. However, the pairkey is historically referred to the pair of agents: the one for agents A and B is $\Pi_{ab} = \{\!|A|\!\}_{Kb} \oplus \{\!|B|\!\}_{Ka}$, where $\oplus$ is the bit-wise exclusive-or operator. While A's card can compute $\{\!|B|\!\}_{Ka}$ and then $\pi_{ab} = \{\!|A|\!\}_{Kb}$ from Π_{ab}, B's card can compute π_{ab} directly. Hence, the two cards share the long-term secret π_{ab}, which we call *pair-k* for A and B.

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

Figure 6: The Shoup-Rubin protocol

The protocol develops through seven phases. The odd-numbered ones take place over the network, while the even-numbered ones cover the communication between agents and smart cards.

Phase I. An initiator A tells the trusted server that she wants to initiate a session with a responder B, and receives in return the pairkey Π_{ab} and the certificate, encrypted under her long-term key, for the pairkey.

Phase II. A queries her card and receives a fresh nonce and the certificate, encrypted under the card key, for the nonce. The form of A's query is specified neither by the designers nor by the implementors, so our choice of message 3 is arbitrary.

Phase III. A contacts B sending her identity and her nonce Na.

Phase IV. B queries his card with the data received from A, and obtains a new nonce Nb, the session key Kab, a certificate for Na and Nb, and a certificate for Nb; Kab is constructed as a function of Nb and π_{ab}.

Phase V. B forwards his nonce Nb and the certificate for Na and Nb to A.

Phase VI. A feeds her card B's name, the two nonces (she has just received Nb), the pairkey and the certificate for it, the two certificates for the nonces; A's card computes π_{ab} from Π_{ab} and uses it with the nonce Nb to compute the session key Kab; the card outputs Kab and the certificate for Nb, which is encrypted under π_{ab}.

Phase VII. A forwards the certificate for Nb to B.

The protocol reveals the pairkey to the spy in step 2, as well as A's nonce in step 5, and B's nonce in step 8. An informal account for the consequences is difficult. One of our formal guarantees (§6.5) will state that, although the session key is computed out of B's nonce, the knowledge of that nonce does not help the spy to discover the key as long as she cannot use A and B's cards.

5 Modelling Shoup-Rubin

The protocol never uses pairkeys as cryptographic keys but merely as a means to establish the corresponding pair-k's. Therefore, we treat pairkeys as nonces, declaring

$$\mathsf{Pairkey} : \mathsf{agent} * \mathsf{agent} \longrightarrow \mathsf{nat}$$

On the contrary, pair-k's are used as proper cryptographic keys, as well as session keys, which are constructed from nonces and pair-k's. So, we declare

$$\mathsf{pairK} : \mathsf{agent} * \mathsf{agent} \longrightarrow \mathsf{key} \qquad\qquad \mathsf{sesK} : \mathsf{nat} * \mathsf{key} \longrightarrow \mathsf{key}$$

At the operational level, we do not need to explore the implementative details behind these components: by contrast, we are interested in their abstract properties. The function Pairkey cannot be declared collision-free because it represents an application of the exclusive-or operator. As expected, this will influence the corresponding confidentiality argument. Assuming that collision of keys is impossible, the other two functions are declared as collision-free, and their ranges as disjoint. Also, these are respectively disjoint from the ranges of the functions formalising other long-term keys (§3.1.3), so that any pair-k differs, for example, from any card key.

The definition of initState must be updated. The protocol relies on cards that are PIN-less, so all occurrences of the function pinK may be omitted. The initial knowledge of the server must also comprise all pairkeys and all pair-k's.

$$\begin{aligned}\mathsf{initState\,S} \;\triangleq\; & \{\mathsf{Key}\,(\mathsf{crdK}\,C)\} \cup \{\mathsf{Key}\,(\mathsf{shrK}\,A)\} \cup \\ & \{\mathsf{Key}\,(\mathsf{pairK}(A,B))\} \cup \{\mathsf{Nonce}\,(\mathsf{Pairkey}(A,B))\}\end{aligned}$$

The friendly agents' initial knowledge is empty, so they are not able to reveal any secrets to the spy.

$$\mathsf{initState}\,(\mathsf{Friend}\,i) \;\triangleq\; \{\}$$

Recall the definition of pairkey and pair-k from the previous section. The spy's initial knowledge must be extended by the pair-k for a pair of agents in case the card of the second agent is cloned, because the spy knows the agent's shared key. A pairkey must be included if both the corresponding cards are cloned.

$$\begin{aligned}\mathsf{initState\,Spy} \;\triangleq\; & \{\mathsf{Key}\,(\mathsf{crdK}\,C) \mid C \in \mathsf{cloned}\} \cup \\ & \{\mathsf{Key}\,(\mathsf{shrK}\,A) \mid (\mathsf{Card}\;\;A) \in \mathsf{cloned}\} \cup \\ & \{\mathsf{Key}\,(\mathsf{pairK}(A,B)) \mid (\mathsf{Card}\;\;B) \in \mathsf{cloned}\} \cup \\ & \{\mathsf{Nonce}\,(\mathsf{Pairkey}(A,B)) \mid (\mathsf{Card}\;\;A) \in \mathsf{cloned} \wedge \\ & \qquad\qquad\qquad\qquad\qquad\quad (\mathsf{Card}\;\;B) \in \mathsf{cloned}\}\end{aligned}$$

The formalisations of smart cards, events and spy are inherited from the general treatment presented in section 3. The model also features a smart card for the server, although it is never used in this protocol.

We declare the constant shouprubin as a set of lists of events. It designates the formal protocol model for the Shoup-Rubin protocol and is defined in the rest of the section by means of inductive rules. The notation should be self-explanatory, but general guidelines may be found elsewhere [26]. Since the protocol makes the assumption of secure means, definition $3'$ of illegal usability for cards that are PIN-less (§3.3) applies.

5.1 Basics

The basic rules for any formal protocol model are presented in figure 7. The empty trace formalises the initial scenario, in which no protocol session has taken place. Rule *Base* settles the base of the induction stating that the empty trace is admissible in the protocol model. All other rules represent inductive steps, so they detail how to extend a given trace of the model. In particular, rule *Reception* allows messages sent on the network to be received by their intended recipients.

Base
```
[] ∈ shouprubin
```

Reception
```
[| evsR ∈ shouprubin; Says A B X ∈ set evsR |]
⟹ Gets B X # evsR ∈ shouprubin
```

Figure 7: Modelling Shoup-Rubin: basics

5.2 Phase I

The rules modelling phase I of the protocol are presented in figure 8. Any agent may initiate a protocol session at any time, hence the corresponding event may extend any trace of the model (*SR1*). Upon reception of a message quoting two

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

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

Figure 8: Modelling Shoup-Rubin: phase I

agent names — initiator and responder of the session — the server computes the pairkey for them and sends it with a certificate to the initiator (*SR2*). Although the pairkey is sent in the clear, it does not reveal its peers in force of the exclusive-or application that yields it. This information is carried by the certificate, which explicitly associates pairkeys with peers.

5.3 Phase II

The rules modelling phase II of the protocol are presented in figure 9. The initiator of a session may query her own smart card provided that she received a message containing a nonce and a certificate (*SR3*). The initiator gets no assurance that the nonce is in fact the pairkey for her and the intended responder, or that the certificate is specific for the pairkey. Since the message traversed the network in the clear, the spy might have tampered with it. It would seem sensible that the agent forwarded the entire message to the smart card, which would be able to decrypt the certificate and verify the authenticity of the pairkey. However, the protocol design does not encompass this, so our model chooses a simpler input message containing only the initiator's name. Given the input, the card issues a fresh nonce and a certificate for it (*SR4*). The card keeps no record of the nonce in order to conserve memory. The certificate

```
SR3
[| evs3 ∈ shouprubin; legallyU(Card A);
   Says A Server {|Agent A, Agent B|} ∈ set evs3;
   Gets A {|Nonce Pk, Cert|} ∈ set evs3 |]
⟹ Inputs A (Card A) (Agent A) # evs3 ∈ shouprubin

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

Figure 9: Modelling Shoup-Rubin: phase II

will subsequently show the card the authenticity of the nonce. Both steps rest on a legally usable smart card because they express some of the legal operations performed by the card owner.

5.4 Phase III

The rules modelling phase III of the protocol are presented in figure 10. When the initiator obtains a nonce and a certificate from her smart card, she may forward the nonce along with her identity to the intended responder (*SR5*). Later (phase V, §5.6), the responder obtains a message of the same form with

```
SR5
[| evs5 ∈ shouprubin;
   Says A Server {|Agent A, Agent B|} ∈ set evs5;
   Outputs (Card A) A {|Nonce Na, Cert|} ∈ set evs5;
   ∀ p q. Cert ≠ {|p, q|} |]
⟹ Says A B {|Agent A, Nonce Na|} # evs5 ∈ shouprubin
```

Figure 10: Modelling Shoup-Rubin: phase III

a different certificate, and must perform different events. At that stage, should the responder initiate another protocol session with a third agent, he could not decide whether to behave according to phase III or to phase V unless he checks the certificate. If it is a one-component cipher-text, then phase III follows; if it is a compound message, then phase V follows. These alternatives may be discerned in practice by the length of the certificate. However, since they are mutually exclusive, our treatment of phase III simply requires the certificate not to be a compound message. Both the designers and the implementors of the protocol omit stating this check, introducing ambiguity . Incidentally, recall that, when the certificate is a cipher-text, no agent can check its internal structure because its encryption key is only known to some smart card.

5.5 Phase IV

The rules modelling phase IV of the protocol are presented in figure 11. This phase sees the responder forward a clear-text message received from the network to his smart card, provided that the card is legally usable (*SR6*). The smart

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

SR7
[| evs7 ∈ shouprubin; legallyU(Card B);
   Nonce Nb ∉ used evs7; Key (sesK(Nb,pairK(A,B))) ∉ used evs7;
   Inputs B (Card B) {|Agent A, Nonce Na|} ∈ set evs7|]
⟹ Outputs (Card B) B {|Nonce Nb, Key (sesK(Nb,pairK(A,B))),
                        Crypt (pairK(A,B)) {|Nonce Na, Nonce Nb|},
                        Crypt (pairK(A,B)) (Nonce Nb)|}
     # evs7 ∈ shouprubin
```

Figure 11: Modelling Shoup-Rubin: phase IV

card issues a fresh nonce, computes the pair-k for initiator and responder, and uses these components to produce a session key. The nonce being fresh, the session key is also fresh. Finally, the card outputs the nonce, the session key and two certificates (*SR7*). One certificate establishes the association between the initiator's nonce and the responder's, and will be inspected by the initiator's card in phase VI. The other certificate will be retained by the responder, who shall be expecting it again from the network in the final phase.

5.6 Phase V

The rules modelling phase V of the protocol are presented in figure 12. When the

```
SR8
[| evs8 ∈ shouprubin;
   Inputs B (Card B) {|Agent A, Nonce Na|} ∈ set evs8;
   Outputs (Card B) B {|Nonce Nb, Key K, Cert1, Cert2|} ∈ set evs8 |]
⟹ Says B A {|Nonce Nb, Cert1|} # evs8 ∈ shouprubin
```

Figure 12: Modelling Shoup-Rubin: phase V

responder obtains from his card a nonce followed by a key and two certificates, he prepares to sending the nonce and one of the certificates to the initiator (*SR8*). However, he must recall having previously quoted the initiator's identity to the card, trusting the card output to refer to his specific input. Notice that the three components following the nonce in the card output might be seen as a unique certificate, thus inviting the ambiguity discussed above (§5.4).

5.7 Phase VI

The rules modelling phase VI of the protocol are presented in figure 13. The scenario returns on the initiator. Before she queries her legally usable card, she verifies that she has taken hold of three messages, each containing a nonce and a certificate. She takes on trust the nonce *Pk* as the pairkey and *Cert1* as its

```
SR9
[| evs9 ∈ shouprubin; legallyU(Card A);
   Says A Server {|Agent A, Agent B|} ∈ set evs9;
   Gets A {|Nonce Pk, Cert1|} ∈ set evs9;
   Outputs (Card A) A {|Nonce Na, Cert2|} ∈ set evs9;
   Gets A {|Nonce Nb, Cert3|} ∈ set evs9;
   ∀ p q. Cert2 ≠ {|p, q|} |]
⟹ Inputs A (Card A) {|Agent B, Nonce Na, Nonce Nb, Nonce Pk,
                        Cert1, Cert3, Cert2|}
      # evs9 ∈ shouprubin

SR10
[| evs10 ∈ shouprubin; legallyU(Card A);
   Inputs A (Card A) {|Agent B, Nonce Na, Nonce Nb, Nonce (Pairkey(A,B)),
                       Crypt (shrK A) {|Nonce (Pairkey(A,B)), Agent B|},
                       Crypt (PairK(A,B)) {|Nonce Na, Nonce Nb|},
                       Crypt (crdK (Card A)) (Nonce Na)|} ∈ set evs10 |]
⟹ Outputs (Card A) A {|Key (sesK(Nb,pairK(A,B))),
                        Crypt (pairK(A,B)) (Nonce Nb)|}
      # evs10 ∈ shouprubin
```

Figure 13: Modelling Shoup-Rubin: phase VI

certificate. She recalls having obtained from her smart card a nonce *Na* with a certificate that is not a compound message, which signifies that the nonce was issued for her when she was acting as initiator. Then, she treats *Nb* as the responder's nonce and *Cert3* as a certificate for *Na* and *Nb*. Finally, she feeds these components to her smart card (*SR9*). The card checks that all the received components have the correct form and, if so, computes the pair-k from the pairkey and then produces the session key and a certificate for the responder's nonce (*SR10*).

5.8 Phase VII

The rules modelling phase VII of the protocol are presented in figure 14. It can

```
SR11
[| evs11 ∈ shouprubin;
   Says A Server {|Agent A, Agent B|} ∈ set evs11;
   Outputs (Card A) A {|Key K, Cert|} ∈ set evs11 |]
⟹ Says A B (Cert) # evs11 ∈ shouprubin
```

Figure 14: Modelling Shoup-Rubin: phase VII

be seen that upon reception of a cryptographic key and a certificate from her smart card, the initiator forwards the certificate to the responder.

5.9 Threats

In addition to the legal behaviour described above for a generic pair of agents, the spy can also act illegally. She observes the traffic on each trace, extracts all message components, and builds all possible fake messages to send on the network or to input to the illegally usable cards. This is modelled by rule *Fake* in figure 15 (derived from figure 3, §3.4).

```
Fake
[| evsF ∈ shouprubin; illegallyU(Card A);
   X ∈ synth (analz (knows Spy evsF)) |]
⟹ Says Spy B X # Inputs Spy (Card A) X # evsF ∈ shouprubin
```

Figure 15: Modelling Shoup-Rubin: threats on messages

We assume that the algorithm used by the cards to compute the session keys is publicly known. Therefore, should the spy know the relevant components of a session key, she would be able to compute the key. We allow this by Paulson's strategy for the TLS protocol [27] rather than by extending the definition of synth, which would complicate the mechanisation process. If the spy obtains a nonce and a pair-k, she can note the corresponding session key by the rule *Forge* in figure 16, thus acquiring knowledge of it. Since the pair-k's are never sent

```
Forge
[| evsFo ∈ shouprubin; Nonce Nb ∈ analz (knows Spy evsFo);
   Key (pairK(A,B)) ∈ knows Spy evsFo |]
⟹ Notes Spy (Key (sesK(Nb,pairK(A,B)))) # evsFo ∈ shouprubin
```

Figure 16: Modelling Shoup-Rubin: threats on session keys

on the network but merely used as encryption keys, they can only be known initially by definition of initState.

Since Shoup-Rubin makes the assumption of secure means, the model must be extended to allow the spy to obtain the outputs of the illegally usable cards. Following the template in figure 4 (§3.4), we introduce a further rule for each card output. Rule *SR4_Fake* in figure 17 is built from *SR4*, while analogous rules

```
SR4_Fake
[| evs4F ∈ shouprubin; illegallyU(Card A); Nonce Na ∉ used evs4F;
   Inputs Spy (Card A) (Agent A) ∈ set evs4F |]
⟹ Outputs (Card A) Spy {|Nonce Na, Crypt (crdK (Card A)) (Nonce Na)|}
      # evs4F ∈ shouprubin
```

Figure 17: Modelling Shoup-Rubin: threats on card outputs

SR7_Fake (built from *SR7*) and *SR10_Fake* (built from *SR10*) are also needed but here omitted for brevity.

5.10 Accidents

We complete the model by allowing accidents (or breaches of security) on session keys by the rules in figure 18. This is typically done by a single rule [26], or by two rules leaking two different kinds of session keys [9]. Shoup-Rubin requires

```
OopsB
[| evsOb ∈ shouprubin;
   Outputs (Card B) B {|Nonce Nb, Key K, Cert,
                        Crypt (pairK(A,B)) (Nonce Nb)|}
     ∈ set evsOb |]
⟹ Notes Spy {|Key K, Nonce Nb, Agent A, Agent B|} # evsOb ∈ shouprubin

OopsA
[| evsOa ∈ shouprubin;
   Outputs (Card A) A {|Key K, Crypt (pairK(A,B)) (Nonce Nb)|}
     ∈ set evsOa |]
⟹ Notes Spy {|Key K, Nonce Nb, Agent A, Agent B|} # evsOa ∈ shouprubin
```

Figure 18: Modelling Shoup-Rubin: accidents

both peers to handle the same session key, respectively in phases IV and VI. Therefore, the spy has a chance to discover the session key from both of them. In the worst case, she will also discover the nonce used to compute the key and the identity of its peers (*OopsA*, *OopsB*).

The spy cannot learn any pair-k's by accident because no agent ever sees any. By definition of initState, she can only know some initially by exploiting the relevant cloned cards.

6 Verifying Shoup-Rubin

For the sake of readability, the symbols for set membership, logical connectives and equality are often replaced by the equivalent wording within the theorem statements in this and the following sections. Each theorem is stated on a generic trace *evs* of the protocol model. The terminology "friendly agent" is abused to include also the server.

Our *reliability theorems* hold on the basis of the general assumptions (§3) we made on the server, the smart cards, and the friendly agents (§6.1). They also confirm that our model correctly implements those assumptions and, at the same time, show that messages 7 and 10 lack explicitness. Suitable regularity lemmas [26] can be expressed about all three kinds of long-term keys employed by the protocol (§6.2). While the authenticity argument (§6.3) only yields a single guarantee for the card that belongs to the protocol initiator, the unicity argument (§6.4) provides guarantees for both initiator and responder. We have discovered that the goals of confidentiality (§6.5), authentication (§6.6), and key distribution (§6.7) are weakened by the mentioned lack of explicitness in the sense that they hold on assumptions that the protocol peers cannot verify. Indeed, in general, if a guarantee is applicable by either protocol peer, then the corresponding goal is confirmed to that peer, namely it is *available* to him [7]. However, for example, the initiator cannot deduce the confidentiality of the session key without trusting that the responder is not the spy or the responder's card is not illegally usable. The initiator will never be able to verify whether her trust is justified. These and similar assumptions, typically about agents or cards, form the *minimal trust* [7] necessary for certain guarantees to hold.

The guarantees proved for a smart card protocol may also be expressed from the viewpoints of smart cards, helping optimise their hardware or software design. With Shoup-Rubin, a guarantee that requires inspecting the form of a certificate may be useful to cards but is never useful to agents, who cannot decipher any certificates since they know no long-term keys.

6.1 Reliability of the Shoup-Rubin Model

Proving the reliability theorems is inherently simple: first apply induction and then conclude by quick simplifications. But these theorems significantly increase our trust that the model is consistent with our assumptions.

6.1.1 On the Model Server

We can prove that the model server sends correct messages (theorem Reli1). But this result cannot be made useful to A, namely it cannot be proved on assumptions that A can verify. Indeed, should A receive message $\{\!|\mathsf{Nonce}\, Pk, \mathit{Cert}|\!\}$, she could not be guaranteed that the server sent it, because the message is compound; nor can she inspect the form of the certificate.

Theorem (Reli1). *If evs contains*

$$\mathsf{Says\ Server}\ A\ \{\!|\mathsf{Nonce}\, Pk, \mathit{Cert}|\!\}$$

then there exists B such that

$$Pk = \mathsf{Pairkey}(A, B) \quad \textit{and}$$
$$\mathit{Cert} = \mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|\mathsf{Nonce}\,(\mathsf{Pairkey}(A, B)), \mathsf{Agent}\, B|\!\} \qquad \square$$

Further guarantees concern the use that the protocol makes of the smart cards, the outputs produced by the cards, and the inputs sent by the friendly agents to the cards.

6.1.2 On the Use of the Smart Cards

If a friendly agent queries a smart card or receives a message from it, then the card must belong to that agent and must be legally usable (theorem Reli2). This confirms that a friendly agent can only use his own card and can only use it legally, as we assumed.

Theorem (Reli2). *If A is not the spy, and evs contains either*

$$\mathsf{Inputs}\,A\,C\,X \quad \textit{or} \quad \mathsf{Outputs}\,C\,A\,Y$$

then

$$C = (\mathsf{Card}\,A) \quad \textit{and} \quad \mathsf{legallyU}(\mathsf{Card}\,A) \qquad \square$$

Our spy can act both legally and illegally. In fact, if the spy uses a smart card, then the card must be either the spy's own card, which is legally usable, or some other agent's card that is illegally usable (theorem Reli3). Since the spy's card is not illegally usable, the agent A mentioned by the theorem certainly differs from the spy.

Theorem (Reli3). *If evs contains either*

$$\mathsf{Inputs}\,\mathsf{Spy}\,C\,X \quad \textit{or} \quad \mathsf{Outputs}\,\mathsf{Spy}\,A\,Y$$

then

$$\begin{aligned}&(C = (\mathsf{Card}\,\mathsf{Spy}) \quad \textit{and} \quad \mathsf{legallyU}(\mathsf{Card}\,\mathsf{Spy})) \quad \textit{or}\\ &(\exists A.\ C = (\mathsf{Card}\,A) \quad \textit{and} \quad \mathsf{illegallyU}(\mathsf{Card}\,A))\end{aligned} \qquad \square$$

6.1.3 On the Outputs of the Smart Cards

To confirm that the model smart cards work reliably, two categories of guarantees can be proved on the Outputs events.

One category states that the cards only give the correct outputs when fed the corresponding inputs, so the cards cannot grant the spy unlimited resources. The case for step 10 of the protocol is presented below (theorem Reli4), while those for steps 4 and 7 are similar and omitted here.

Theorem (Reli4). *If evs contains*

$$\begin{aligned}\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\mathsf{Key}\,(\mathsf{sesK}(Nb, \mathsf{pairK}(A,B))),\\ \mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb)|\!\}\end{aligned}$$

then there exists Na such that evs also contains

$$\begin{aligned}\mathsf{Inputs}\,A\,(\mathsf{Card}\,A)\,\{\!|&\mathsf{Agent}\,B, \mathsf{Nonce}\,Na, \mathsf{Nonce}\,Nb, \mathsf{Nonce}\,(\mathsf{Pairkey}(A,B)),\\ &\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Nonce}\,(\mathsf{Pairkey}(A,B)), \mathsf{Agent}\,B|\!\},\\ &\mathsf{Crypt}(\mathsf{pairK}(A,B))\{\!|\mathsf{Nonce}\,Na, \mathsf{Nonce}\,Nb|\!\},\\ &\mathsf{Crypt}(\mathsf{crdK}(\mathsf{Card}\,A))(\mathsf{Nonce}\,Na)\ |\!\}\end{aligned} \qquad \square$$

Another category confirms that the cards produce correct outputs. Precisely, given a specific card output, the form of its components can be tracked down. One such guarantee can be established on an instance of message 4 (theorem Reli5). The length of the certificate must be checked because of the protocol ambiguity already encountered (§5.4). Recall that an event $\mathsf{Outputs}\,C\,A\,X$ also models A's reception of X, so the theorem is applicable also by A.

Theorem (Reli5). *If evs contains*

$\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\mathsf{Nonce}\,Na, Cert|\!\}$

and Cert is not compound, then

$Cert = \mathsf{Crypt}(\mathsf{crdK}(\mathsf{Card}\,A))(\mathsf{Nonce}\,Na)$ □

Analogous considerations apply to message 7. Upon B's reception of an output, we can guarantee its form for some peer A and some nonce Na (theorem Reli6). The existential form of the assertion tells us that B receives the session key in a message that does not inform him of the peer with whom the key is to be used. This violates a well-known explicitness principle (perhaps unknown at the time of the design): "Every message should say what it means. The interpretation of the message should depend only on its content." [2, §2.1]. The underlying transport protocol cannot reveal the peer's identity either, and this somewhat weakens the goals of confidentiality, authentication and key distribution accomplished by the protocol, as discussed in the following.

Theorem (Reli6). *If evs contains*

$\mathsf{Outputs}\,(\mathsf{Card}\,B)\,B\,\{\!|\mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab, Cert1, Cert2|\!\}$

then there exist A and Na such that

$Kab = \mathsf{sesK}(Nb, \mathsf{pairK}(A, B))$ *and*
$Cert1 = \mathsf{Crypt}(\mathsf{pairK}(A, B))\{\!|\mathsf{Nonce}\,Na, \mathsf{Nonce}\,Nb|\!\}$ *and*
$Cert2 = \mathsf{Crypt}(\mathsf{pairK}(A, B))(\mathsf{Nonce}\,Nb)$ □

The card outputs are correct particularly in the case of message 10 (theorem Reli7). The existential form of the assertion reveals another lack of explicitness of the protocol design. When A receives the session key, she has to infer from the context the identity of the peer with whom to use the key. This task is entirely heuristic: the card might give outputs in the wrong order. Likewise, the nonce associated with the key is not explicit in the message.

Theorem (Reli7). *If evs contains*

$\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\mathsf{Key}\,Kab, Cert|\!\}$

then there exist B and Nb such that

$Kab = \mathsf{sesK}(Nb, \mathsf{pairK}(A, B))$ *and*
$Cert = \mathsf{Crypt}(\mathsf{pairK}(A, B))(\mathsf{Nonce}\,Nb)$ □

The theorem also shows that step 10 binds the session key to the card that creates it, and to the certificate. Therefore, should the structure of the session key be inspectable, the structure of the certificate could be derived (corollary Reli8), and vice versa.

Corollary (Reli8). *If evs contains*

$\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\mathsf{Key}\,(\mathsf{sesK}(Nb, \mathsf{pairK}(A', B))), Cert|\!\}$

then,

$A = A'$ *and* $Cert = \mathsf{Crypt}(\mathsf{pairK}(A, B))(\mathsf{Nonce}\,Nb)$ □

6.1.4 On the Friendly Agents' Use of the Smart Cards

Other guarantees show that the friendly agents indeed use the legally usable cards in a legal manner. We must verify that these agents produce inputs whose origin can be documented. For example, let us assume that an agent A queries a card as in step 9 of the protocol (by theorem Reli4, the card belongs to A) quoting an agent B. We can prove that A initiated a session with B via the server, and received the components of the query either from the network or from the card by means of suitable events (theorem Reli9).

Theorem (Reli9). *If A is not the spy, and evs contains*

$$\begin{aligned}&\mathsf{Inputs}\, A\, C\, \{\!|\mathsf{Agent}\, B, \mathsf{Nonce}\, Na, \mathsf{Nonce}\, Nb, \mathsf{Nonce}\, Pk,\\ &\qquad \mathit{Cert1}, \mathit{Cert2}, \mathit{Cert3}|\!\}\end{aligned}$$

then evs also contains

$$\begin{aligned}&\mathsf{Says}\, A\ \mathsf{Server}\, \{\!|\mathsf{Agent}\, A, \mathsf{Agent}\, B|\!\}\quad \mathit{and}\\ &\mathsf{Gets}\, A\, \{\!|\mathsf{Nonce}\, Pk, \mathit{Cert1}|\!\}\quad \mathit{and}\\ &\mathsf{Gets}\, A\, \{\!|\mathsf{Nonce}\, Nb, \mathit{Cert2}|\!\}\quad \mathit{and}\\ &\mathsf{Outputs}\ C\, A\, \{\!|\mathsf{Nonce}\, Na, \mathit{Cert3}|\!\}\end{aligned}$$

□

Although the first event of the conclusion highlights A's intention to communicate with B, none of the remaining events mentions B. So, A cannot be assured to be feeding her card the components meant for the session with B. Even if the Gets events mentioned B, his identity would not be reliable as the spy can tamper with compound messages coming from the network. The Outputs event could mention B reliably as it takes place over a secure means, but fails to do so. However, by theorem Reli4, A will get an output from her card only if she uses the correct components as input.

Similar theorems deal with the other queries to the smart cards, respectively steps 3 and 6 of the protocol, but are omitted here for brevity.

The form of the inputs created in steps 3 and 6 of the protocol are self-explanatory. Step 9 is more complicated. While most of such input has been exposed to the network risks (theorem Reli9), the third certificate has not, namely it is created off the network. So, its form can be derived (theorem Reli10), signifying that every agent uses her own card according to its functional interface, sending it correct inputs. Notice that the guarantee also holds for the spy's use of her own card.

Theorem (Reli10). *If evs contains*

$$\begin{aligned}&\mathsf{Inputs}\, A\, (\mathsf{Card}\, A)\, \{\!|\mathsf{Agent}\, B, \mathsf{Nonce}\, Na, \mathsf{Nonce}\, Nb, \mathsf{Nonce}\, Pk,\\ &\qquad\qquad \mathit{Cert1}, \mathit{Cert2}, \mathit{Cert3}|\!\}\end{aligned}$$

then

$$\mathit{Cert3} = \mathsf{Crypt}(\mathsf{crdK}(\mathsf{Card}\, A))(\mathsf{Nonce}\, Na)$$

□

Upon reception of a message, A's card can determine whether the message is an instance of message 9 by looking at its clear-text part. The card should inspect carefully the second and third certificates because they could be fake. Their form is in fact not provable in the model. Having proved the integrity of the third certificate may suggest that it is superfluous to the design, and that the card could avoid checking it. Nevertheless, should an agent insert a fake nonce as second component of message 9, the card would detect the misbehaviour by inspecting the third certificate.

6.2 Regularity

Friendly agents are never required to send long-term keys on the network, hence the spy can send such a key if and only if she knows it before the protocol begins. In consequence, the spy can learn a card key and a card owner's key only from cloning the card (by definition of initState, §5). Using the latter key, she can compute all the pair-k's meant for the card owner. These assertions are formalised and confirmed by the following regularity lemmas (Regu1, Regu2, and Regu3).

Lemma (Regu1).

$$(\mathsf{Key}\,(\mathsf{shrK}\,A) \in \mathsf{analz}(\mathsf{knows}\,\mathsf{Spy}\ \ evs)) \Longleftrightarrow (\mathsf{Card}\,A \in \mathsf{cloned}) \qquad \square$$

Lemma (Regu2).

$$(\mathsf{Key}\,(\mathsf{crdK}\,C) \in \mathsf{analz}(\mathsf{knows}\,\mathsf{Spy}\ \ evs)) \Longleftrightarrow (C \in \mathsf{cloned}) \qquad \square$$

Lemma (Regu3).

$$(\mathsf{Key}\,(\mathsf{pairK}(P, B)) \in \mathsf{analz}(\mathsf{knows}\,\mathsf{Spy}\ \ evs)) \Longleftrightarrow (\mathsf{Card}\,B \in \mathsf{cloned}) \qquad \square$$

They resemble Paulson's regularity lemmas for traditional protocols, but concern the smart cards. They are expressed here in terms of the analz operator, rather than parts, which makes them directly applicable in the following. The traditional proof strategies [26] scale up.

6.3 Authenticity

"Agents need guarantees (subject to conditions they can check) confirming that their certificates are authentic"[26, §4.7]. If a certificate appears to come from an agent A, and this can be proved, then the certificate is authentic.

In designing a theorem, the obvious first move is to identify those assumptions that seem to enforce the conclusion. In the case of an authenticity theorem, we must look for the assumptions that apparently prevent the spy from forging the certificate. Then, once the protocol step that created the certificate is found, our purpose becomes enforcing the corresponding event. If the proof succeeds for a certificate that is sealed under a long-term key (except a PIN), then it is not useful to the agents running a smart card protocol, for they know no long-term keys and therefore cannot recognise any certificates.

All Shoup-Rubin's certificates are sealed under long-term keys, so the agents get no authenticity guarantees about them. Those keys are stored in the smart cards, so the authenticity argument can be directed to the cards. Only in step 9 does a card get encrypted certificates. To reason about their authenticity, we develop some subsidiary lemmas, which are not directly applicable either by agents or by cards. Incidentally, if a card receives a certificate as part of an input on a trace *evs* over secure means, the definition of knows alone does not imply that the certificate is on the network traffic, namely in $\mathsf{parts}(\mathsf{knows\,Spy}\ \ evs)$

Along with a pairkey, the server issues a certificate that verifies it. When the certificate is on the traffic, we can prove that it originated with the server if the regularity lemma Regu1 is applicable. Therefore, given that the peer's card is not cloned, the certificate is authentic (lemma Auth1). At this stage, the form of the pairkey may be specified via theorem Reli1.

Lemma (Auth1). *If A's card is not cloned, and evs is such that*

$$\mathsf{Crypt}(\mathsf{shrK}\ A)\{\!|\mathsf{Nonce}\ Pk, \mathsf{Agent}\ B|\!\} \in \mathsf{parts}(\mathsf{knows\,Spy}\ \ evs)$$

then evs contains

$$\mathsf{Says\ Server}\ A\ \{\!|\mathsf{Nonce}\ Pk, \mathsf{Crypt}(\mathsf{shrK}\ A)\{\!|\mathsf{Nonce}\ Pk, \mathsf{Agent}\ B|\!\}|\!\} \qquad \square$$

We can verify formally that the certificate that associates A and B's nonces is built in step 7 (lemma Auth2). Since the certificate is sealed under the corresponding pair-k, investigating its origin requires an appeal to the regularity lemma Regu3, which prescribes B's card not to be cloned. However, a stronger assumption is needed on B's card to solve case *SR7_Fake*: the card must not be illegally usable, otherwise it could output towards the spy, rather than towards B.

Lemma (Auth2). *If B's card is not illegally usable, and evs is such that*

$$\mathsf{Crypt}(\mathsf{pairK}(A,B))\{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb|\!\} \in \mathsf{parts}(\mathsf{knows\,Spy}\ \ evs)$$

then evs contains

$$\begin{aligned}\mathsf{Outputs}\,(\mathsf{Card}\ B)\ B\ \{\!|\ &\mathsf{Nonce}\ Nb, \mathsf{Key}\,(\mathsf{sesK}(Nb, \mathsf{pairK}(A,B))),\\ &\mathsf{Crypt}(\mathsf{pairK}(A,B))\{\!|\mathsf{Nonce}\ Na, \mathsf{Nonce}\ Nb|\!\},\\ &\mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\ Nb)\ |\!\}\end{aligned} \qquad \square$$

Message 7 ends with another certificate that verifies B's nonce, $\{\!|Nb|\!\}_{\pi_{ab}}$. So, we can prove a theorem, omitted here, that is identical to theorem Auth2 except for the certificate considered and for the assertion existentially quantified over the nonce Na. The same certificate is also output by A's card in message 10. Proving this result (lemma Auth3) also requires B not to be the spy in order to solve case *SR7* (so that the corresponding event does not introduce the certificate on the traffic), and A's card not to be illegally usable to solve case *SR10_Fake*.

Lemma (Auth3). *If B is not the spy, A and B's cards are not illegally usable, and evs is such that*

$$\mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb) \in \mathsf{parts}(\mathsf{knows}\,\mathsf{Spy}\ \ evs)$$

then evs contains

$$\begin{aligned}\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\,&\mathsf{Key}\,(\mathsf{sesK}(Nb,\mathsf{pairK}(A,B))),\\&\mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb)\,|\!\}\end{aligned} \qquad \square$$

The authenticity lemmas serve to prove an authenticity theorem that is applicable by A's card (theorem Auth4). The theorem must include the assumptions on agents and cards required by the lemmas. Upon reception of message 9, the card must inspect the first two certificates, as indicated by theorem Reli10. If the first certificate has the expected form, then theorem Reli9 and lemma Auth1 prove the first event of the assertion (once a message is received, its components appeared on the traffic). Similarly, if the second certificate is as expected, then theorem Reli9 and lemma Auth2 prove the second event. The third certificate does not need to be inspected thanks to its provable authenticity stated by theorem Reli9, which alone justifies the third event of the assertion. We have mechanised this reasoning by one Isabelle command that applies theorem Reli9 only once, and then the appropriate authenticity lemma.

Theorem (Auth4). *If A is not the spy, A's card is not cloned, B's card is not illegally usable, and evs contains*

$$\begin{aligned}\mathsf{Inputs}\ A\ (\mathsf{Card}\,A)\ \{\!|\,&\mathsf{Agent}\,B,\mathsf{Nonce}\,Na,\mathsf{Nonce}\,Nb,\mathsf{Nonce}\,Pk,\\&\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Nonce}\,Pk,\mathsf{Agent}\,B|\!\},\\&\mathsf{Crypt}(\mathsf{pairK}(A,B))\{\!|\mathsf{Nonce}\,Na,\mathsf{Nonce}\,Nb|\!\},\,Cert3\,|\!\}\end{aligned}$$

then evs also contains

$$\begin{aligned}&\mathsf{Says}\ \mathsf{Server}\ A\,\{\!|\mathsf{Nonce}\,Pk,\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Nonce}\,Pk,\mathsf{Agent}\,B|\!\}|\!\}\quad \textit{and}\\&\mathsf{Outputs}\ (\mathsf{Card}\,B)\ B\ \{\!|\,\mathsf{Nonce}\,Nb,\mathsf{Key}\,(\mathsf{sesK}(Nb,\mathsf{pairK}(A,B))),\\&\qquad\mathsf{Crypt}(\mathsf{pairK}(A,B))\{\!|\mathsf{Nonce}\,Na,\mathsf{Nonce}\,Nb|\!\},\\&\qquad\mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb)\,|\!\}\quad \textit{and}\\&\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\mathsf{Nonce}\,Na,\,Cert3|\!\}\end{aligned} \qquad \square$$

The authenticity of the crucial message components can be investigated in the same fashion as that of the certificates. Let us consider the authenticity of pairkeys. Only the server is entitled to issue $\mathsf{Pairkey}(A,B)$, which does not belong to the initial knowledge of the spy if either A or B's card is not cloned. For example, let us suppose that A's card is not cloned. Apparently, this should enforce that, if a pairkey is on the traffic, then it was issued by the server. However, attempting to prove that, if *evs* is such that

$$\mathsf{Pairkey}(A,B) \in \mathsf{parts}(\mathsf{knows}\,\mathsf{Spy}\ \ evs)$$

then, for some *Cert*, *evs* contains

$$\mathsf{Says}\ \mathsf{Server}\ A\,\{\!|\mathsf{Nonce}\,(\mathsf{Pairkey}(A,B)),\,Cert|\!\}$$

leaves the following subgoal arising from case *Base*

$$\begin{array}{l} [\,|\ \ \mathsf{Card}\,A \notin \mathsf{cloned}; \\ \quad \mathsf{Pairkey}(A,B) = \mathsf{Pairkey}(A',B'); \\ \quad \mathsf{Card}\,A' \in \mathsf{cloned};\ \ \mathsf{Card}\,B' \in \mathsf{cloned}\ |\,] \implies \mathsf{False} \end{array}$$

We cannot derive that $A = A'$ because the pairkey is implemented in terms of the exclusive-or operator, which is not collision-free. The subgoal can be in fact falsified because there may exist two pairs of distinct agents A, A' and B, B' who satisfy the premises. This proof attempt signifies that the spy might exploit the collisions suffered by the exclusive-or operator and forge a pairkey even if she does not know its original components. The probability of this happening is influenced by the redundancy introduced by the encryption function and by the length of the cipher-texts.

We now examine the authenticity of the session key. This crucial message component is only sent between cards and agents, never through the network. Despite this, the spy could either forge it (by *Forge*), or obtain it from her own card if she is one of the peers (by *SR7* or *SR10*), or learn it from the illegally usable cards (by *SR7_Fake* or *SR10_Fake*). Let us make the assumptions that prevent all these circumstances. For example, if the responder's card is not illegally usable and therefore not cloned, then the session key cannot be forged by lemma Regu3. Then, if a session key ever appears on the traffic, one of its peers necessarily leaked it by accident, while the trace recorded the corresponding oops event (lemma Auth5). This is in fact a counterguarantee of authenticity because it emphasises the conditions under which a session key that is on the traffic is not authentic: the spy in fact introduced it. Also, it will be fundamental below (§6.5) to assess a form of session key confidentiality.

Lemma (Auth5). *If A and B are not the spy, their cards are not illegally usable, and evs is such that*

$$\mathsf{Key}\,(\mathsf{sesK}(Nb, \mathsf{pairK}(A,B))) \in \mathsf{parts}(\mathsf{knows}\,\mathsf{Spy}\ \ evs)$$

then evs contains

$$\mathsf{Notes}\,\mathsf{Spy}\ \{\!|\mathsf{Key}\,(\mathsf{sesK}(Nb, \mathsf{pairK}(A,B))), \mathsf{Nonce}\,Nb, \mathsf{Agent}\,A, \mathsf{Agent}\,B|\!\}\qquad \square$$

Proving the authenticity lemmas for Shoup-Rubin requires a standard strategy (simpler than Paulson's for the authenticity theorems on traditional protocols [26, §4.7]), which appears to be generalisable to smart card protocols.

1. Apply induction.
2. If the lemma concerns
   - a certificate sealed under a shared key, then simplify case *Fake* by lemma Regu1;
   - a certificate sealed under a card key, then simplify case *Fake* by lemma Regu2;

- a certificate sealed under a pair-k, then simplify case *Fake* by lemma Regu3;
- a session key, then apply "$H \subseteq \mathsf{parts}\, H$" to case *Forge* and simplify it by lemma Regu3.

3. Solve case *Fake* by a standard tactic [26, §4.5].
4. Apply the reliability theorems as follows: theorem Reli5 to case *SR9*, theorem Reli6 to cases *SR8* and *OopsB*, theorem Reli7 to case *SR11*, and a variant of theorem Reli7 — which binds the form of the certificate, given the form of the session key — to case *OopsA*.
5. Simplify remaining cases.

6.4 Unicity

Key distribution protocols issue a fresh key per each session. The freshness assumption enforces that the same session key cannot be issued more than once. Using this argument, the unicity theorems could be designed [26, §4.4] and then made useful to agents [4, §4.1]. The same argument can be applied to smart card protocols but, in addition, a reliability theorem allows us to prove a further unicity guarantee about session keys.

Shoup-Rubin requires B's card to build a fresh session key in message 7. The key is bound uniquely to the remaining components of the message (theorem Unic1). In proving this result, after induction and simplification two subgoals remain, which are about *SR7* and *SR7_Fake*. The latter is easily solvable because it forces the spy to use her own card illegally, which is impossible. The other case is solved by freshness: the session key could not appear before. Message 7 also contains B's fresh nonce, so a variant of the theorem may be proved using the nonce as pivot.

Theorem (Unic1). *If evs contains*

$$\mathsf{Outputs}\,(\mathsf{Card}\, B)\; B\; \{\!|\mathsf{Nonce}\, Nb, \mathsf{Key}\, Kab, Cert1, Cert2|\!\}\quad \textit{and}$$
$$\mathsf{Outputs}\,(\mathsf{Card}\, B')\; B'\; \{\!|\mathsf{Nonce}\, Nb', \mathsf{Key}\, Kab, Cert1', Cert2'|\!\}$$

then

$$B = B' \quad \textit{and} \quad Nb = Nb' \quad \textit{and} \quad Cert1 = Cert1' \quad \textit{and} \quad Cert2 = Cert2' \qquad \square$$

A similar theorem, which is omitted here, holds about the card output in step 4, by exploiting the freshness of A's nonce. More surprisingly, it also holds about the card output in step 10 (theorem Unic2), although the card uses no fresh components on that occasion. Corollary Reli8 supplies. Whenever a specific session key appears, the form of the corresponding certificate can be assessed, so the same key cannot stand by two different certificates. This strategy solves the subgoal about *SR10*, while the subgoal about *SR10_Fake* is terminated routinely.

Theorem (Unic2). *If evs contains*

$$\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\mathsf{Key}\,Kab, Cert|\!\}\quad and$$
$$\mathsf{Outputs}\,(\mathsf{Card}\,A')\,A'\,\{\!|\mathsf{Key}\,Kab, Cert'|\!\}$$

then

$$A = A' \quad and \quad Cert = Cert' \qquad \square$$

The unicity theorems teach agents a lot. For example, if in the real world B receives the same session key within two different instances of message 7, he may suspect that something wrong happened. Having violated theorem Unic1, the scenario is due to problems that lie outside our model, ranging from a malfunction of B's card to a spy's break-in between the agent and his card. Theorem Unic2 provides the equivalent information to A.

However, if B happens to receive the same session key within the same message more than once, theorem Unic1 would not be violated. Still, should that scenario alarm B? The answer is affirmative because in our model, where B's card always outputs a fresh key, we can prove that the mentioned scenario never occurs. We define the predicate

$$\mathsf{Unique}\ ev\ \mathsf{on}\ evs \ \equiv\ ev \notin \mathsf{set}(\mathsf{tl}(\mathsf{dropWhile}(\lambda z.z \neq ev)\ evs))$$

which scans the trace *evs* till the event *ev* is found and then skipped; if *ev* does not belong to the set of events of the remaining list, then *ev* only occurs once on *evs*.

Upon reception of any output commencing with a nonce, B can be assured that the corresponding event is unique (theorem Unic3). After expanding the definition of the predicate, the cases about *SR4* and *SR7* are solved by freshness of the nonce. The result also applies to the output of A's card in step 4. No similar theorem can be established about step 10, which does not involve any fresh components.

Theorem (Unic3). *If evs contains*

$$\mathsf{Outputs}\,(\mathsf{Card}\,B)\,B\,\{\!|\mathsf{Nonce}\,Nb, rest|\!\}$$

then

$$\mathsf{Unique}\ (\mathsf{Outputs}\,(\mathsf{Card}\,B)\,B\,\{\!|\mathsf{Nonce}\,Nb, rest|\!\})\ \mathsf{on}\ evs \qquad \square$$

6.5 Confidentiality

Because of the weakness discovered by the authenticity argument (§6.3), a pairkey cannot be proved confidential even assuming that it has not been sent in the network and that its components cannot be forged.

The regularity lemmas may be interpreted as non-trivial confidentiality guarantees. Moreover, applying $\mathsf{analz}\,H \subseteq \mathsf{parts}\,H$ to the authenticity theorem Auth5, we obtain a guarantee of session key confidentiality: a session key that cannot be forged and that has not been leaked by accident remains confidential (theorem Conf 1). Unfortunately, this theorem is not useful to agents because it requires inspection of the structure of the session key.

Theorem (Conf 1). *If A and B are not the spy, their cards are not illegally usable, and evs does not contain*

$$\mathsf{Notes\,Spy}\ \{\!|\mathsf{Key}\,(\mathsf{sesK}(Nb, \mathsf{pairK}(A,B))), \mathsf{Nonce}\,Nb, \mathsf{Agent}\,A, \mathsf{Agent}\,B|\!\}$$

then

$$\mathsf{Key}\,(\mathsf{sesK}(Nb, \mathsf{pairK}(A,B))) \notin \mathsf{analz}(\mathsf{knows\,Spy}\ \ evs) \qquad \square$$

Now, we initiate Paulson's proof strategy for confidentiality, and find that Shoup-Rubin does not confirm its goals of confidentiality to the peers if the data buses of the cards are unreliable. However, the following guarantees can be applied by the smart cards.

The confidentiality argument for a protocol responder B must develop on an event that B can verify: his card sending a message that contains the session key in step 7 of the protocol. The event formalising such step,

$$\mathsf{Outputs}\,(\mathsf{Card}\,B)\,B\ \{\!|\mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab, Cert1, Cert2|\!\}$$

includes two certificates, *Cert1* and *Cert2*, that B cannot inspect because they are sealed by long-term keys. We have attempted to prove Kab confidential on a trace *evs* that contains the mentioned event but no oops event leaking Kab. Also, B's card must be assumed not to be cloned otherwise the spy would know $\mathsf{pairK}(P,B)$ for any agent P and could so be able to forge the session key by rule *Forge*. The proof leaves two subgoals unsolved, respectively arising from cases *SR10* and *SR10_Fake*. The inspection of the former teaches that B's peer might be the spy, who could so obtain a copy of Kab from her own smart card. The latter subgoal shows that B's peer's card could be illegally usable regardless the identity of the peer; the spy would be able to use this card to compute Kab. While the protocol requires B's card to issue a new session key in step 7, his peer in fact computes a copy of the key from available components in step 10.

Therefore, further assumptions are necessary on B's peer and her card, but the message obtained by B does not state the identity of such peer. This reveals that B does not get the identity of the peer with whom the session key is to be shared, which violates a well-known explicitness principle due to Abadi and Needham: "If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message" [2, §4]. If we inspect either one of the certificates, then B's peer becomes explicit, so the relevant assumptions can be stated and theorem Conf 2 proved.

Theorem (Conf 2). *If A and B are not the spy, A's card is not illegally usable, B's card is not cloned, and evs contains*

$$\mathsf{Outputs}\,(\mathsf{Card}\,B)\,B\ \{\!|\mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab, Cert,\ \mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb)|\!\}$$

but does not contain

$$\mathsf{Notes\,Spy}\ \{\!|\mathsf{Key}\ Kab, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}$$

then

$$\mathsf{Key}\ Kab \notin \mathsf{analz}(\mathsf{knows\,Spy}\ \ evs) \qquad \square$$

From B's viewpoint, trusting that the peer is not malicious and that her card cannot be used by the spy is indispensable — it is part of what we called minimal trust. So is trusting that the key has not been leaked by accident.

However, B cannot verify that the main event of the theorem ever occurs because he cannot inspect the certificate. Therefore, he cannot apply the theorem. Shoup and Rubin's analysis asserts an analogous property requiring that the peers' cards be "unopened" [30, §3.1], which may be interpreted as "not cloned" here. However, their treatment does not investigate whether the property is in fact applicable by the peers.

When reasoning from A's viewpoint, the outcome is similar. Attempting to prove confidentiality on the assumption that the event formalising step 10,

$$\mathsf{Outputs}\,(\mathsf{Card}\ A)\ A\ \{\!|\mathsf{Key}\ Kab, Cert|\!\}$$

occurs, leaves the subgoals arising from *SR7* and *SR7_Fake* unsolved. They highlight that A could be communicating either with the spy or with an agent whose card is illegally usable. Indeed, step 10 fails to express A's peer. Like the previous theorem, also this one can be proved if the form of *Cert* is explicit, resulting in a guarantee that can be applied by A's card but not by A (theorem Conf 3).

Theorem (Conf 3). *If A and B are not the spy, A and B's cards are not illegally usable, and evs contains*

$$\mathsf{Outputs}\,(\mathsf{Card}\ A)\ A\ \{\!|\mathsf{Key}\ Kab, \mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\ Nb)|\!\}$$

but does not contain

$$\mathsf{Notes\,Spy}\ \{\!|\mathsf{Key}\ Kab, \mathsf{Nonce}\ Nb, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}$$

then

$$\mathsf{Key}\ Kab \notin \mathsf{analz}(\mathsf{knows\,Spy}\ \ evs) \qquad \square$$

6.6 Authentication

Phase V terminates B's role in the protocol. Then, B's peer, say A, obtains the session key from message 10, but the identity of B remains unspecified unless the certificate is inspected. If the certificate is not fake, induction proves it to have appeared with the instance of message 7 that concerns B (theorem Auth6). Although A cannot appeal to the theorem, it becomes significant to A's card, which can inspect the certificate. When the card issues A with the session key, it is guaranteed that both B and his card were present on the network and that B's card, which is using the pair-k for A and B, is participating in a session with A. The proof observes that the event of the assumption implies that the certificate $\{\!|Na, Nb|\!\}_{\pi_{ab}}$ appears on the traffic for some Na; then, it applies lemma Auth2.

Theorem (Auth6). *If B's card is not illegally usable, and evs contains*

$$\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\mathsf{Key}\,Kab, \mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb)|\!\}$$

then, for some Na, evs also contains

$$\begin{aligned}\mathsf{Outputs}\,(\mathsf{Card}\,B)\,B\,\{\!|\,&\mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab,\\ &\mathsf{Crypt}(\mathsf{pairK}(A,B))\{\!|\mathsf{Nonce}\,Na, \mathsf{Nonce}\,Nb|\!\},\\ &\mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb)\,|\!\}\end{aligned} \qquad \square$$

A relevant authentication guarantee for B should establish that A is active after B creates the session key. At the end of the protocol, B may receive from the network the certificate for his nonce. Provided that lemma Auth3 is applicable, A's card can be proved to have sent a suitable instance of message 10, which establishes the presence of A with her card, and the intention of A's card to communicate with B (theorem Auth7).

Theorem (Auth7). *If B is not the spy, A and B's cards are not illegally usable, and evs contains*

$$\mathsf{Gets}\,B\,(\mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb))$$

then evs also contains

$$\begin{aligned}\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\,&\mathsf{Key}\,(\mathsf{sesK}(Nb, \mathsf{pairK}(A,B))),\\ &\mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb)\,|\!\}\end{aligned} \qquad \square$$

Is this theorem useful to B? The answer is negative because the agent cannot inspect the encrypted certificate. In practice B obtains no information about the sender of the certificate, and his peer remains unknown. Observing that the certificate was originally created in message 7 does not help because neither that message states the peer (theorem Reli6). A possible solution, which we have verified, is concluding the protocol with two additional steps: B forwarding the certificate to his card, and the card responding with A's identity. The card should use the right pair-k to decrypt the certificate, thus identifying A. While adding explicitness to message 7 is a simpler fix (as demonstrated below, §7), making the guarantee available also to B's card necessarily requires those additional steps.

6.7 Key Distribution

Applying the definition of knows to the conclusion of theorem Auth6 yields that, when A's card computes the session key for A, the key is already known to B (theorem Keyd1). However, A gains nothing from this result because the certificate must be inspected. Notice that the guarantee itself does not prevent B from being the spy.

Theorem (Keyd1). *If B's card is not illegally usable, and evs contains*

$$\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\mathsf{Key}\,Kab, \mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb)|\!\}$$

then

$$\mathsf{Key}\,Kab \in \mathsf{analz}(\mathsf{knows}\,B\ \ evs) \qquad \square$$

Let us attempt to design the corresponding guarantee for B. His session key is obtained via message 7. By theorem Auth7, if B receives the last message of the protocol, he infers that A obtained *some* session key. The two events must be correlated in order to assure that both peers hold the *same* session key. This can only be done by inspecting one of the certificates of message 7, so to make A explicit. Then, theorem Reli6 specifies the form of the session key that is output by B's card (theorem Keyd2).

Theorem (Keyd2). *If B is not the spy, A and B's cards are not illegally usable, and evs contains*

$$\begin{aligned}&\mathsf{Outputs}\,(\mathsf{Card}\,B)\,B\,\{\!|\mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab, Cert,\\ &\qquad\qquad \mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb)|\!\}\quad \textit{and}\\ &\mathsf{Gets}\,B\,(\mathsf{Crypt}(\mathsf{pairK}(A,B))(\mathsf{Nonce}\,Nb))\end{aligned}$$

then

$$\mathsf{Key}\,Kab \in \mathsf{analz}(\mathsf{knows}\,A\ \ evs) \qquad \square$$

No stronger result than this can be envisaged because there exists no protocol message that binds the session key with both of its peers. Can B inspect any of the certificates of message 7 ? Or, can B's card inspect that of the last message? Both answers being negative, theorem Keyd2 turns out to be applicable neither by B nor by his card, which seems a poor achievement for the protocol.

7 Verifying an Upgraded Shoup-Rubin

Omitting B's name from message 2 of the public-key Needham-Schroeder protocol led to Lowe's well-known attack [21] — the spy could interleave two sessions, learn an important nonce and violate the authentication of the initiator to the responder. With Shoup-Rubin, the assumption of secure means prevents this. However, since the data buses of the cards are unreliable (§3.1.1), the agents cannot distinguish which protocol session a single output belongs to, because the card outputs lack explicitness.

Abadi and Needham demonstrate that lack of explicitness may crucially affect the interpretation of a message — "The names relevant for a message can sometimes be deduced from other data and from what encryption keys have been applied. However, when this information cannot be deduced, its omission is a blunder with serious consequences." [2, §4].

As mentioned above, message 7 of Shoup-Rubin cannot inform B of A's identity both because the session key does not state its peers and because the two cipher-texts cannot be decrypted by agents. Nor can message 10 inform A of B's identity. Moreover, while message 7 quotes the nonce Nb that is used to build the session key, message 10 fails to do so. These components cannot be learnt from the underlying transport protocol. Therefore, upon reception of an instance of message 10, agent A cannot derive the complete form of the instance of message 7 sent during that session.

However, messages 6 and 9 do quote the identity of the respective, intended peer. So, should the data buses be reliable, the calling agent could store the identity of the peer until the card returned, and associate the session key just received to that peer. Nevertheless, messages 7 and 10 do violate the explicitness principles mentioned above.

These considerations suggest upgrading messages 7 and 10, which are transmitted over a secure means, with the relevant agent names and, for simmetry, with a nonce. The upgrades are underlined in figure 19, which omits the unaltered rest of the protocol. It is straightforward to upgrade the formal protocol model accordingly. In the new model, many of the theorems discussed above ob-

$$\begin{array}{lll} 7. & C_b \to B: & Nb, \underline{A}, Kab, \{\!|Na, Nb|\!\}_{\pi_{ab}}, \{\!|Nb|\!\}_{\pi_{ab}} \\ 10. & C_a \to A: & \underline{B, Nb}, Kab, \{\!|Nb|\!\}_{\pi_{ab}} \end{array}$$

Figure 19: Upgrading the Shoup-Rubin protocol

tain slightly modified assertions and, crucially, assumptions that never inspect cipher-texts. Therefore, the assumptions become verifiable by agents, signifying that the upgraded protocol makes its guarantees available to them. For example, theorem Reli4 can now be enforced on the event

$$\mathsf{Outputs}\,(\mathsf{Card}\,A)\;A\;\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab, Cert|\!\}$$

The assertion of theorem Reli6 can be stripped of one existential, so B learns the peer for the session key (theorem Reli6′). One existential still constrains the form of one of the certificates, but B's knowledge is not significantly affected.

Theorem (Reli6′). *If evs contains*

$$\mathsf{Outputs}\,(\mathsf{Card}\,B)\;B\;\{\!|\mathsf{Nonce}\,Nb, \mathsf{Agent}\,A, \mathsf{Key}\,Kab, Cert1, Cert2|\!\}$$

then

$$\begin{array}{l} Kab = \mathsf{sesK}(Nb, \mathsf{pairK}(A, B)) \quad \textit{and} \\ Cert2 = \mathsf{Crypt}(\mathsf{pairK}(A, B))(\mathsf{Nonce}\,Nb) \end{array}$$

and there exists Na such that

$$Cert1 = \mathsf{Crypt}(\mathsf{pairK}(A, B))\{\!|\mathsf{Nonce}\,Na, \mathsf{Nonce}\,Nb|\!\} \qquad \square$$

Similarly, proving theorem Reli7 on the event

$$\mathsf{Outputs}\,(\mathsf{Card}\,A)\;A\;\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab, Cert|\!\}$$

avoids the existential quantifiers in the assertion because both B and Nb are already bound. The second event enforced by theorem Reli7 obtains an extra component, while the rest of the authenticity argument remains unaltered.

The unicity results continue to hold. For example, theorem Unic2 must now cope with the additional components (theorem Unic2′).

Theorem (Unic2′). *If evs contains*

$$\mathsf{Outputs}\,(\mathsf{Card}\,A)\;A\;\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab, Cert|\!\}\quad \textit{and}$$
$$\mathsf{Outputs}\,(\mathsf{Card}\,A')\;A'\;\{\!|\mathsf{Agent}\,B', \mathsf{Nonce}\,Nb', \mathsf{Key}\,Kab, Cert'|\!\}$$

then

$$A = A' \quad \textit{and} \quad B = B' \quad \textit{and} \quad Nb = Nb' \quad \textit{and} \quad Cert = Cert' \qquad \square$$

Theorem Conf 2 gets a simpler main assumption

$$\mathsf{Outputs}\,(\mathsf{Card}\,B)\;B\;\{\!|\mathsf{Nonce}\,Nb, \mathsf{Agent}\,A, \mathsf{Key}\,Kab, Cert1, Cert2|\!\}$$

which is verifiable by B, and so does theorem Conf 3, which now rests on

$$\mathsf{Outputs}\,(\mathsf{Card}\,A)\;A\;\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab, Cert|\!\}$$

In consequence, the peers will be able to decide, within the minimal trust, whether the session key they obtain is confidential.

Both authentication theorems are strengthened. Agent A can now be informed that B and his card were present on the network and that B's card intended to communicate with A (theorem Auth6′). Agent A must only verify to receive from her card a message with four components: an agent name, a nonce, a key, and a cipher-text. The theorem could be more specific on the form of *Cert1*, but this would not enrich A's knowledge substantially.

Theorem (Auth6′). *If B's card is not illegally usable, and evs contains*

$$\mathsf{Outputs}\,(\mathsf{Card}\,A)\;A\;\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab, Cert2|\!\}$$

then, for some Cert1, evs also contains

$$\mathsf{Outputs}\,(\mathsf{Card}\,B)\;B\;\{\!|\mathsf{Nonce}\,Nb, \mathsf{Agent}\,A, \mathsf{Key}\,Kab, Cert1, Cert2|\!\} \qquad \square$$

Also theorem Auth7 can be reformulated (theorem Auth7′) as to become applicable by B, who can check the reception from the network of a cipher-text previously obtained from his card. This result is still not applicable by B's card.

Theorem (Auth7′). *If B is not the spy, A and B's cards are not illegally usable, and evs contains*

$\mathsf{Outputs}\,(\mathsf{Card}\,B)\,B\,\{\!|\mathsf{Nonce}\,Nb, \mathsf{Agent}\,A, \mathsf{Key}\,Kab, \mathit{Cert1}, \mathit{Cert2}|\!\}$ *and*
$\mathsf{Gets}\,B\,(\mathit{Cert2})$

then evs also contains

$\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab, \mathit{Cert2}|\!\}$ □

Theorem Keyd1 can be enforced on the same assumptions as those of theorem Auth6′, so becoming applicable by A (theorem Keyd1′).

Theorem (Keyd1′). *If B's card is not illegally usable, and evs contains*

$\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,Nb, \mathsf{Key}\,Kab, \mathit{Cert2}|\!\}$

then,

$\mathsf{Key}\,Kab \in \mathsf{analz}(\mathsf{knows}\,B\ evs)$ □

Similarly, the assertion of theorem Keyd2 can be proved on the assumptions of theorem Auth7′ and become useful to B. The resulting theorem and theorem Keyd1′ signify that the upgraded protocol confirms the goal of key distribution to its peers.

8 Conclusions

Modern networking strongly demands clear guarantees to dissipate a number of concerns [24]. One concern is to verify whether or not smart card protocols take real advantage from the use of the cards to achieve stronger goals. In the light of our work, it seems fair to conclude that this hierarchy of protocols can be verified inductively regardless of whether they assume secure means between agents and smart cards, or whether they employ PIN-operated cards.

Paulson's original machinery for traditional protocols clearly had to be extended with the modelling of the smart cards, but we are pleased to observe that the extensions were quite straightforward. For example, the risks of physical tampering with the cards can be intuitively included, while a basic set of stored secrets can be easily extended according to the protocol to analyse. We remark that our modelling of the cards is entirely reusable for future protocol analyses.

Designing guarantees for smart card protocols requires a meticulous understanding of how the cards influence the protocol goals. Therefore, it was positive to find out that the proof strategies for traditional protocols could be adapted with limited efforts. As we have remarked, all guarantees must be interpreted from the viewpoints of the protocol peers, otherwise they are inapplicable even within the minimal trust [7], but also the viewpoints of the cards may become relevant. The Shoup-Rubin protocol, which consists of 11 steps, is the longest protocol analysed so far using the Inductive Approach. Even encoding it in an inductive definition required a certain effort, which offers an additional justification to research towards a common language for specifying cryptographic protocols [11]. Our proofs establish that the protocol achieves its goals for a pair of peers whose cards are safe from the spy. In particular, the goals are achieved independently from risks to other agents' cards, signifying that local breaches of security only have local consequences. Not only does this support the claim that the protocol makes an intelligent use of the smart cards, but it also encourages research towards the tamper resistance of the cards. However, if the cards give their outputs in an unspecified order, then two of the protocol messages need extra explicitness in order for the goals to be confirmed to the peers.

New protocols based on smart cards are being developed for electronic payment over the Internet. They establish a form of electronic currency whereby a coin is just a unique random number created and signed by a bank. The secrets stored in the cards should help prevent, for example, double spending. We believe that verifying those protocols may help develop the path to pervasive E-commerce. Mechanised inductive proofs now shed some light on that path.

Acknowledgements. Dieter Gollmann clarified the meaning of message authenticity. Peter Honeyman unveiled a few technicalities of the Shoup-Rubin protocol. Markus Kuhn inspired a detailed account for the assumption of secure means. Catherine Meadows gave invaluable suggestions to improve the presentation. The anonymous referees made useful comments. I am indebted to Larry Paulson and to Peter Ryan, for important discussions and large comments on a draft of this paper. The research was funded by the EPSRC grant GR/R01156/01 *Verifying Electronic Commerce Protocols*.

References

[1] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. Authentication and Delegation with Smart-cards. Research Report 125, Digital - Systems Research Center, 1994.

[2] M. Abadi and R. M. Needham. Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[3] R. J. Anderson and M. J. Kuhn. Low Cost Attacks on Tamper Resistant Devices. In M. et al. Lomas, editor, *Proc. of the 5th International Workshop on Security Protocols*, LNCS 1361, pages 125–136. Springer-Verlag, 1997.

[4] G. Bella. Message Reception in the Inductive Approach. Research Report 460, Computer Laboratory, University of Cambridge, 1999. Available on the Internet.

[5] G. Bella. Modelling Agents' Knowledge Inductively. In B. Christianson, B. Crispo, J. A. Malcolm, and R. Michael, editors, *Proc. of the 7th International Workshop on Security Protocols*, LNCS 1796, pages 85–94. Springer-Verlag, 1999.

[6] G. Bella. Modelling Security Protocols Based on Smart Cards. In M. Blum and C. H. Lee, editors, *Proc. of the International Workshop on Cryptographic Techniques & E-Commerce (CrypTEC'99)*, pages 139–146. City University of Hong Kong, 1999.

[7] G. Bella. *Inductive Verification of Cryptographic Protocols.* PhD thesis, Research Report 493, Computer Laboratory, University of Cambridge, 2000. Available on the Internet.

[8] G. Bella. Mechanising a Protocol for Smart Cards. In I. Attali and T. Jensen, editors, *Proc. of International Conference on Research in Smart Cards (e-Smart'01)*, LNCS 2140, pages 19–33. Springer-Verlag, 2001.

[9] 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, *Proc. of the 5th European Symposium on Research in Computer Security (ESORICS'98)*, LNCS 1485, pages 361–375. Springer-Verlag, 1998.

[10] M. Bellare and P. Rogaway. Provably Secure Session Key Distribution — the Three Party Case. In *Proc. of the 27th ACM SIGACT Symposium on Theory of Computing (STOC'95)*, pages 57–66. ACM Press and Addison Wesley, 1995.

[11] S. Brackin, C. Meadows, and J. Millen. CAPSL Interface for the NRL Protocol Analyzer. In *Proc. of 2nd IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'99).* IEEE Press, 1999.

[12] M. Burrows, M. Abadi, and R. M. Needham. A Logic of Authentication. *Proc. of the Royal Society of London*, 426:233–271, 1989.

[13] F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand Spaces: Why is a Security Protocol Correct? In *Proc. of the 17th IEEE Symposium on Security and Privacy.* IEEE Press, 1998.

[14] Forrester Research. `http://www.forrester.com`.

[15] Gary Gaskell and Mark Looi. Integrating Smart Cards Into Authentication Systems. In Ed Dawson and Jovan Colić, editors, *Proc. of the 1st International Conference on Cryptography: Policy and Algorithms*, LNCS 1029, pages 270–281, 1995.

[16] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, 33:210–217, 1986.

[17] N. Itoi and P. Honeyman. Smartcard Integration with Kerberos V5. In *Proc. of the USENIX Workshop on Smartcard Technology*, 1999.

[18] R. Jerdonek, P. Honeyman, K. Coffman, J. Rees, and K. Wheeler. Implementation of a Provably Secure, Smartcard-based Key Distribution Protocol. In J.-J. Quisquater and B. Schneier, editors, *Proc. of the 3rd Smart Card Research and Advanced Application Conference (CARDIS'98)*, 1998.

[19] O. Kömmerling and M. G. Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. In *Proc. of the USENIX Workshop on Smartcard Technology*, 1999.

[20] T. Leighton and S. Micali. Secret-key Agreement without Public-key Cryptography. In D. R. Stinson, editor, *Proc. of Advances in Cryptography — CRYPTO'93*, LNCS 773, pages 456–479. Springer-Verlag, 1993.

[21] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-key Protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Proc of the 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, LNCS 1055, pages 147–166. Springer-Verlag, 1996.

[22] D. P. Maher. Fault Induction Attacks, Tamper Resistance, and Hostile Reverse Engineering in Perspective. In R. Hirschfeld, editor, *Proc. of Financial Cryptography '97*, LNCS 1318, pages 109–121. Springer-Verlag, 1997.

[23] C. Meadows. Private conversations. 2001.

[24] C. Meadows. Open Issues in Formal Methods for Cryptographic Protocol Analysis. In *Proc. of DISCEX 2000*. IEEE Press, 2000.

[25] C. A. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

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

[27] L. C. Paulson. Inductive Analysis of the Internet protocol TLS. *ACM Transactions on Computer and System Security*, 2(3):332–351, 1999.

[28] L. C. Paulson. Proving Protocols Correct. In *Proc. of the 14th International Conference of Logic in Computer Science (LICS'99)*, pages 370–383. IEEE Press, 1999.

[29] P. Y. A. Ryan and S. A. Schneider. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2000.

[30] V. Shoup and A. Rubin. Session Key Distribution using Smart Cards. In U. Maurer, editor, *Advances in Cryptology — Eurocrypt'96*, LNCS 1070, pages 321–331. Springer-Verlag, 1996.