Inductive Verification of Cryptographic Protocols

Giampaolo Bella

Clare College
University of Cambridge

A dissertation submitted for the degree of
Doctor of Philosophy

March 2000

Preface

Except where otherwise stated in the text, this dissertation is the result of my own work and is not the outcome of work done in collaboration.

This dissertation is not substantially the same as any I have submitted for a degree or diploma or any other qualification at any other university.

No part of my dissertation has already been, or is being currently submitted for any such degree, diploma or other qualification.

This dissertation does not exceed sixty thousand words, including footnotes and bibliography.

Publications

Some of the work presented in this dissertation has been published, and may be found in references [11, 12, 13, 15, 17, 18, 19, 20].

For comments or suggestions, email `Giampaolo.Bella@cl.cam.ac.uk`

Acknowledgements

I am extremely grateful to Larry Paulson, my supervisor, for the perfect mixture of knowledge, experience and patience with which he has guided my research. On the human side, he has taught me how to face unsuccessful experiments and learn from them, and how to evaluate the successful ones. On the practical side, he has found a solution for my unexpected financial hardship due to inexplicable problems with my initial awarding body, which would have compromised the completion of my research. I thankfully acknowledge assistance from the EPSRC grants GR/K77051 and GR/K57381.

I am also particularly grateful to my Ph.D. examiners, Mike Gordon and Peter Ryan, who gave me invaluable suggestions on how to make this dissertation more readable and more intelligible.

Special thanks are due to Lewis Tiffany, librarian of the Computer Laboratory, for making the library a friendly environment and for speeding up my access to printed information, and to Margaret Levitt, secretary of the laboratory, for her assistance through the bureaucratic steps of my Ph.D.

My sincere gratitude goes to the professors of the University of Catania for the educational and psychological support they have granted me through my studies: Domenico Cantone, who favoured my security seminars in Catania, Giovanni Gallo, who supervised my first degree, Alfredo Ferro, who encouraged me to undertake a Ph.D. at Cambridge, and Elvinia Riccobene, who discovered my impetus towards research in general, and guided that impetus towards expertise.

I would like to thank the following people: Dieter Gollmann, of Microsoft Research, who clarified for me the relation between integrity and authenticity; Peter Honeyman, of the University of Michigan, who unveiled a few technicalities of the Shoup-Rubin protocol; Gavin Lowe, of the University of Leicester, who facilitated my understanding of authentication; Markus Kuhn, of the Computer Laboratory, who discussed with me some of the risks affecting the use of smart cards. I am indebted to David Richerby, my office-mate, for being an up-to-date and responsive dictionary of English and, especially, for proof-reading the entire dissertation.

I am deeply grateful to Zia Nerina, Zio Tanino and, particularly, to my brother Giuseppe for filling my absence from home with vitality and skill: without them, I could not have accepted to live abroad.

My deepest recognition and my warmest *grazie* go to my father, Carmelo, whose death helped me broaden my views of life, and to my mother, Ada, whose loss of eyesight helped me bring those views to a focus.

To my mother
and to the loving
memory of my father

Abstract

The dissertation aims at tailoring Paulson's Inductive Approach for the analysis of classical cryptographic protocols towards real-world protocols. The aim is pursued by extending the approach with new elements (e.g. timestamps and smart cards), new network events (e.g. message reception) and more expressive functions (e.g. agents' knowledge). Hence, the aim is achieved by analysing large protocols (Kerberos IV and Shoup-Rubin), and by studying how to specify and verify their goals.

More precisely, the modelling of timestamps and of a discrete time are first developed on BAN Kerberos, while comparing the outcomes with those of the BAN logic. The machinery is then applied to Kerberos IV, whose complicated use of session keys requires a dedicated treatment. Three new guarantees limiting the spy's abilities in case of compromise of a specific session key are established. Also, it is discovered that Kerberos IV is subject to an attack due to the weak guarantees of confidentiality for the protocol responder.

We develop general strategies to investigate the goals of *authenticity*, *key distribution* and *non-injective agreement*, which is a strong form of authentication. These strategies require formalising the agents' knowledge of messages. Two approaches are implemented. If an agent creates a message, then he knows all components of the message, including the cryptographic key that encrypts it. Alternatively, a broad definition of agents' knowledge can be developed if a new network event, message reception, is formalised.

The concept of smart card as a secure device that can store long-term secrets and perform easy computations is introduced. The model cards can be stolen and/or cloned by the spy. The kernel of their built-in algorithm works correctly, so they spy cannot acquire unlimited knowledge from their use. However, their functional interface is unreliable, so they send correct outputs in an unspecified order. The provably secure protocol based on smart cards designed by Shoup & Rubin is mechanised. Some design weaknesses (unknown to the authors' treatment by Bellare & Rogaway's approach) are unveiled, while feasible corrections are suggested and verified.

We realise that the evidence that a protocol achieves its goals must be available to the peers. In consequence, we develop a new a principle of prudent protocol design, *goal availability*, which holds of a protocol when suitable guarantees confirming its goals exist on assumptions that both peers can verify. Failure to observe our principle raises the risk of attacks, as is the case, for example, of the attack on Kerberos IV.

Contents

List of Figures

Chapter 1

Introduction

Communicating across modern computer networks should be *secure*. The adjective "secure" typically embodies multiple properties. For example, a peer may wonder whether the message just received has exactly the same form as it was sent. If so, the message is said to enjoy *integrity*. Even if a message that is received quotes a certain peer as its creator, this may not be true. In case it is, then the message is said to be *authentic*. Another important property is whether the message that is received has been intercepted and understood by other peers beside its creator and its receiver. If not, then the message is *confidential*.

While it is a matter of current research to devise a satisfactory list of properties for defining a secure communication in the context of modern computer networks, all of these properties implicitly assume the existence of a malicious entity, *spy* below, who can overhear the communication, create fake messages and introduce them in the traffic. While history tells us that security was already an important issue in ancient times, considerable frauds have still been mounted during the last decade using relatively expensive hardware and software resources.

Cryptography may help. Used massively also during the World Wars [53], it is the art of coding information by a *cryptographic key*. A cleartext message is transformed into a ciphertext using the key (*encryption*). In the best case, the cleartext can be retrieved from the ciphertext (*decryption*) if and only if the key is available. As a consequence, the cleartext is safe from the spy as long as she does not know the key. On the contrary, the intended receiver of the message is assumed to know the key. When the cryptographic key used for encryption is the same as that to be used for decryption, cryptography is said to be *symmetric* or *shared-key* (DES [81], IDEA [61], etc.), otherwise it is *asymmetric* or *public-key* (RSA [93], LUC [99], etc.). In the sequel, we assume a basic familiarity with these techniques.

Steganography [55] may also be used towards secure communications. It is the art of hiding a message inside a larger, intelligible one so that the spy cannot discern the presence of the hidden message from seeing the larger one. For example, the low-order pixel bits of a digital image may be changed to the message bits without the image suffering perceptible variations.

A more recent technique aiming at confidentiality is *chaffing and winnowing* [92], which may be viewed as a form of steganography. It makes use of MACs (a MAC, *message authentication code*, for a message is another message computed out of both the original one and a secret that is shared by the peers). The message sender authenticates the message by adding the correct MAC to it and sends the pair. The sender also sends *chaff*, namely different messages with wrong MACs. Only the intended receiver of the message knows which MAC is correct as he knows the secret key used to compute it. So, he can *winnow* the received messages, namely discard the chaff and select the original message.

The vast majority of *security protocols* for computer networks are based on cryptography, hence the name *cryptographic protocols*. These are sequences of *messages*, possibly encrypted, exchanged between pairs of peers in order to make their subsequent communication secure. Messages include peer names, cryptographic keys, random numbers, timestamps, concatenations of those components and ciphertexts obtained from them. Each protocol attempts to achieve certain *goals* at the time of its completion, namely the set of properties that define security. The peers belong to a larger set of *agents*, who could be interpreted as humans, machines, or processes, as Abadi and Needham point out [4, §2]. We prefer the last interpretation. For example, if a confidentiality guarantee is available to a process running on a workstation, it is not necessarily available to the human that owns the process because there could be a malicious break-in at any level of the workstation architecture. In this dissertation, the word “peer” refers to an agent who is engaged in the execution of a protocol.

Experience shows that cryptographic protocols often fail to enforce the goals claimed by their designers. In fact, establishing whether a protocol lives up its promises is, in general, daunting. While informal reasoning has failed to capture serious protocol failures, formal reasoning can address only a few goals on abstract protocol models. This dissertation extends an existing approach for reasoning on classical protocols towards the analysis of deployed protocols. It is found that the approach scales up and that a number of protocol goals can be formally verified, while unveiling an important principle of prudent protocol design.

1.1 Informal vs. Formal Protocol Analysis

Until the late 1980s, cryptographic protocols were only analysed by informal reasoning. If a protocol claimed to achieve the goal of confidentiality, some researchers studied the protocol in detail and decided whether this was true.

Although the process became more and more scrupulous with the increasing utilisation of cryptographic protocols on local area networks and then over the Internet, it was granted the rigour of formal reasoning only when it was understood that the protocols bore a high potential for flaws. The reason is that each protocol is a concurrent distributed program that can be executed by a large population of agents including the spy. Not only is the spy entitled to participate in the protocol as any other agent, but she can also act illegally, interleaving a number of concurrent protocol sessions. By doing so, she can exploit on a session the messages obtained from others. Moreover, the vulnerability of modern computer networks allows her to overhear the messages exchanged by other agents.

Using cryptography appropriately for enforcing the protocol goals in this setting is not easy. This claim is supported by the large number of flaws that have been reported. Some affect well-known protocols [7, 62], and a subset of them can be classified [100]. Others affect less publicly known banking protocols, whose weaknesses have been exploited by dishonest employees [6]. Some others are due to the specific implementation of the cryptographic primitives [96].

The literature shows that formal approaches can significantly help to detect protocol flaws [56, 74], as well as to yield general principles of secure protocol design. Some approaches lack expressiveness or automation, others are just too complicated to use on realistic protocols. Informal reasoning indeed retains its importance: it is crucial to grasp the semantics of a protocol design beyond its bare representation as a sequence of messages; it may find simple flaws and minor weaknesses of a protocol more quickly than formal reasoning; it is easier to follow for a non-experienced audience; it helps to develop formal approaches. Moreover, the complete understanding of certain flaws or of the guarantees against them is not a trivial matter, as we discuss in the next chapter.

1.2 Induction via Theorem Proving

The simple principle of mathematical induction suffices to model cryptographic protocols in detail: this is Paulson's intuition behind the *Inductive Approach* [88]. We are not aware of other entirely inductive approaches in this field.

The Inductive Approach relies on the concept of *trace* as a list of events occurred on the network while a population of agents is running a protocol. Traces are defined inductively and so is the set of all traces admissible under a specific protocol. This set, which represents the formal protocol model, is unbounded. Proofs may be carried out by induction on a generic trace of the model, establishing trace properties that represent goals of the underlying protocol. More details may be found in chapter 3.

The interactive theorem prover *Isabelle* [85] supports the inductive modelling of the protocol in Higher Order Logic, where nested quantification is permitted over functional symbols, and mechanises the proofs developed by the user.

1.3 Motivations

Our research concerns the verification of realistic protocols using the Inductive Approach. We have three goals: the development of a young approach that shows large potentialities, the verification of protocol goals that have not yet been formally explored in a realistic setting, the investigation of general principles of protocol design.

This section refers to the development state of the Inductive Approach in early 1997 when the present research began.

1.3.1 Developing the Inductive Approach

Further Testing

The approach has only been applied to a few classical cryptographic protocols [86, 87]. Nevertheless, a general theory of messages and an extendible formalisation of the spy are already provided. The great advantage of the entire treatment is that it does not bound the size of the models. Crucially, the population of agents who could participate in the protocol is potentially infinite: the model agents originate from a bijection with the natural numbers. Besides, each agent is allowed to interleave an arbitrary number of protocol sessions.

However, to convince ourselves of the practicality of the approach, further case studies are necessary. In particular, the security community lament that the size of the existing case studies is not realistic. Hence, we decide that the new case studies must be deployed protocols.

Deeper Understanding

Further informal criticism to the approach derives from difficulties in understanding the concept of trace, commented as a “low-level” view of the network traffic, or as a structure “non-existent” in reality. Indeed, it took us some experimentation to develop the view of a trace as a possible *history* of the events occurred on the network. This interpretation has smoothened any subsequent conversation about the foundations of the approach.

All proofs follow the natural inductive style adopted by humans, verifying that a certain property is preserved through the various protocol steps. However, many proofs are widely viewed as “cryptic” and, consequently, their results are only reluctantly accepted. This is imputed to the high level of automation of the released proof scripts. Admittedly, when the Isabelle simplifier drastically reduces the size of a subgoal, it is difficult to track down the rewriting rules that have been applied and in which order. The same concern arises from the use of highly specific, automatic tactics, which may implement several proof steps and aim at tidying up the proof scripts.

A possible solution could be conforming to the following procedure at least in a few, demonstrative proofs: (i) simplifying the subgoals manually to a certain extent by applying the crucial rewriting rule one at a time; (ii) avoiding automatic tactics by a linear application of the steps they implement.

Additional Constituents

Many protocols use timestamps to assure freshness of important components such as session keys. The datatype for messages built in the approach does not comprise timestamps. However, even after the necessary syntactical updates, the main issue would be how to reason about freshness.

No message reception is modelled in the original release of the approach. However, understanding a theorem very often involves some informal reasoning about reception. It would be desirable to make the entire reasoning formal. This raises the issue of how complex updating the existing analyses with new events could be, and whether the outcomes would pay back.

Another important issue is how to account for e-commerce protocols, which often involve smart cards. How could the cards be modelled? How could their functionalities and interaction with agents be represented? How would the protocol analyses be influenced?

1.3.2 Verifying Significant Protocol Goals

Existing Guarantees

A general strategy for proving the goal of confidentiality is provided, but the concept of *viewpoint* is not mentioned in the initial literature [86, 87]. In fact, the guarantees expressed in terms of theorems are useful to their recipients only when they can verify whether the assumptions of the theorems hold. For example, if a proof of session key confidentiality is available on assumptions that the protocol initiator can verify, then the protocol achieves confidentiality from the initiator's viewpoint. Unless the proof can be conducted also on assumptions verifiable by the responder, the protocol does not necessarily attain confidentiality from the responder's viewpoint.

The existing guarantees, which only pertain to confidentiality and authentication often omit these issues. Moreover, they may contain a few assumptions that seem impossible to verify. Is a guarantee with this feature still of any importance?

Also, the goal of authentication is merely formalised in terms of agents' aliveness, whereas it may presuppose far wider requirements, as subsequently discussed using a different approach [65].

New Guarantees

The well-known goals of integrity, authenticity, key distribution and stronger forms of authentication need to be treated. For example, even if the initiator is informed that session key confidentiality holds, it is not obvious that the responder shares the same session key and means to share it with the initiator. It is not clear *a priori* what and how substantial the extensions necessary to formalise these goals might be. Some of the existing guarantees might have to be reinterpreted.

1.3.3 Investigating Principles of Protocol Design

The best collection of principles for designing cryptographic protocols prudently is due to Abadi and Needham [4]. The main principle, supported with various examples, is *explicitness.* The contents of a message should say exactly what the message signifies and, crucially, express the sender and the intended receiver. Other principles include avoiding unnecessary encryption, and synchronising the clocks of the network nodes within a small interval when running a protocol based on timestamps.

It is interesting to investigate whether findings obtained via the Inductive Approach support these principles or clarify how relevant they are towards the goals of a protocol. Besides, the deep study of the protocol goals mentioned in the previous section might unveil additional principles.

1.4 Contributions

Our contributions address all three motivations for the research itself.

1.4.1 On the Inductive Approach

The Inductive Approach scales up to the analysis of real-world protocols thanks to its extensibility.

Timestamps can be modelled using a discrete formalisation of time that exploits the ordering of the elements of each trace. Each history of the network has only a global clock induced by the length of the corresponding trace. Therefore, each trace is equipped with a global clock yielding the current time of the trace. All agents must conform to it, so the model avoids problems of clock synchronisation. A message component is fresh on a trace if the time interval between the creation of the component and the current time of the trace is smaller than or equal to the lifetime allowed for the component. Session keys are considered valid if and only if they are used within their lifetime.

These extensions have allowed us to mechanise the BAN Kerberos protocol [31], which is based on timestamps, and the larger, deployed Kerberos IV [77]. The proof scripts are fairly understandable because, in each proof, the use of automatic tactics has been reduced to the single case concerning the spy's operation.

New events can be modelled. In particular, introducing message reception makes the specifications more readable and the proofs easier to follow, so that the entire treatment becomes more intuitive. Updating the existing scripts can be done pragmatically with minor efforts. Message reception is not forced to occur. This models a network that is entirely controlled by an active spy, who can intercept certain messages and prevent their delivery. Besides, the reception event allows the formalisation of any agent's knowledge rather than just the spy's.

New *objects* can be modelled as new types of the language, as we demonstrate with smart cards. The interaction between the cards and their owners is formalised by additional events. The agents' knowledge, in particular the spy's, must be reviewed for two reasons: (i) all long-term secrets are now stored into the cards; (ii) certain protocols that are based on smart cards assume that the spy cannot listen in between a card and its owner, while others do not assume this. We verify the entire Shoup-Rubin [98] protocol using a faithful model obtained both from the informal specification of the protocol and the description of its implementation. The protocol involves new long-term secrets, which can be easily introduced in the definition of agents' knowledge.

As a result, the Inductive Approach is now equipped with all the necessary features to tackle modern protocols. The experiments carried out with the well-known SET [70, 71, 72] protocol developed by VISA and Mastercard support this claim [16]. In general, the approach is so expressive that the bare statements of the theorems convey most guarantees without additional informal argument. Therefore, the prose that accompanies each theorem can be reduced to the very minimum.

1.4.2 On Protocol Goals

We find that the argument about any protocol goal can (and must, see next section) be interpreted from the viewpoint of each peer. This practice has the double outcome of providing us with a better understanding of each protocol step, and the peers with guarantees that can be practically applied. We have thus realised that one assumption of a theorem proved of the shared-key Needham-Schroeder protocol is in fact superfluous (see §4.6). Moreover, the assumptions that can never be verified by any agent different from the spy, but are still necessary, constitute the *minimal trust* (see §4.8).

Paulson's strategy for proving confidentiality is still effective after the modelling of timestamps. However, several specific lemmas are necessary with Kerberos IV because of its hierarchical distribution of session keys. We have unveiled an important weakness in the protocol management of timestamps and lifetimes, which lets the spy exploit certain session keys within their lifetime. What makes this attack more serious is that the agent to whom the session keys have been legally granted is no longer present on the network, and so will not register any irregularity.

Our treatment supports the claim that the goals of authenticity and integrity are equivalent (see §4.3), while the corresponding guarantees can be derived from some of the existing theorems.

The definition of agents' knowledge, which can be achieved thanks to the modelling of message reception, allows the formal verification of the goal of key distribution. We argue the case that this goal is equivalent to a strong form authentication (§4.7), as we demonstrate on both BAN Kerberos (see §7.3) and Kerberos IV (see §7.4).

Verifying the goals of the protocols that are based on smart cards only requires minor modifications to the existing proof strategies. A set of simplification rules must be proved to deal with the new events and the new definition of agents' knowledge. Also the smart cards require guarantees that the protocol goals are met. Two of the messages of the Shoup-Rubin protocol lack crucial explicitness, so that none of the peers knows which session key is associated with each other. The confidentiality argument is significantly weakened in the realistic setting in which the spy can exploit other agents' smart cards. The proofs suggest a simple fix to the protocol, yielding stronger guarantees.

Our proofs are, admittedly, difficult. When mechanising Kerberos IV and Shoup-Rubin, certain proofs took up to four weeks each to be developed, while each script was up to 50 Isabelle commands long. Polishing the original scripts often shortens them up to one fifth of their original length because the necessary theorems can be installed on the Isabelle automatic tactics. However, this clearly affects the resulting intelligibility, as discussed above. Also, the fact that no other theorem prover has been tailored thus far to protocol analysis in this detail supports the efficiency of our strategies and of Isabelle. Informal conversations with experienced users of other provers confirm this.

1.4.3 On Principles of Protocol Design

Our research confirms the essential importance of explicitness. The messages that are not explicit about their meaning force the peers to heuristic decisions that turn out to be extremely risky. This was known to affect classical protocols such as the public-key Needham-Schroeder, but we find that it also affects protocols that are apparently stronger such as a smart card one, Shoup-Rubin.

It is interesting to note how, if a message lacks explicitness, then carrying out any proofs about it requires quantifying existentially the exact components that are not sufficiently explicit. It could be argued that mere expertise in theorem proving could make up for lack of competency in the area of prudent protocol design.

The verification of Kerberos IV confirms that extra encryption does not necessarily strengthen confidentiality. Despite the double encryption of the responder's session key, the key is vulnerable to the attack mentioned above.

We state a new principle of protocol design, *goal availability*, which holds of a protocol and one of its goals if there exist guarantees from the viewpoints of both protocol peers that the goal is met. We did not encounter this principle in the literature despite its crucial importance. The attack on Kerberos IV is in fact due to a violation of goal availability. The weakness of Shoup-Rubin arise from violations of both our principle and of the one about explicitness. Precisely, the lack of explicitness could only be discovered through the verification of goal availability. In this light, we argue that our principle is more basic and easier to verify.

1.5 Outline

We briefly outline here the following chapters with their contents.

Chapter 2 reviews the main formal approaches towards the analysis of cryptographic protocols, and focuses on interpreting the outcomes of the analyses. The difficulties in obtaining them are not always related to those in interpreting them.

Chapter 3 outlines Paulson's Inductive Approach. The treatment is as much as possible informative. The basic constituents are discussed but the details of their implementations are avoided.

Chapter 4 discusses the desirable goals of cryptographic protocols in addition to the only two, confidentiality and authenticity, that can be found in the literature regarding the Inductive Approach. The strategies for proving them are developed and suitable examples provided. The minimal trust that must be put to obtain the results of most theorems is explained. An important aim of the verification is minimising such trust. The principle of goal availability is stated.

Chapter 5 concerns the extension of the Inductive Approach with timestamps. The BAN Kerberos protocol is mechanised and a suitable formalisation of time provided. The protocol model is then refined by a temporal modelling of session key accidental leaks, which realistically lowers the minimal trust.

Chapter 6 contains the mechanisation of Kerberos IV. The modelling of time and the temporal modelling of accidents on the session keys are inherited from the preceding chapter. The protocol violates our principle of goal availability, and the spy can exploit this to mount an attack. The attack can be prevented by refining the functioning of one of the two trusted servers.

Chapter 7 introduces the modelling of agents' knowledge using two different approaches. These are both demonstrated on the two protocols previously mechanised, verifying a strong goal of authentication and the goal of key distribution, which was impossible before. The argument of their comparison is used to refute a claim of the BAN logic.

Chapter 8 describes the modelling of smart cards. Since some protocols explicitly assume that the communication means between the cards and their owners is reliable, while others do not, our treatment develops around both options. The spy can exploit an unspecified set of smart cards, some through simple theft, others through elaborate tampering.

Chapter 9 contains the mechanisation of the Shoup-Rubin protocol, which is based on smart cards, by the approach extended as in the preceding chapter. The protocol is discovered to violate our principle of goal availability. This reveals that two protocol messages lack explicitness, so that each peer does not know who to associate the received session key with. This affects substantially the goals of confidentiality, authentication and key distribution, but fixing the weakness is reasonably easy and inexpensive.

Chapter 10 concludes the presentation. The research presented throughout the dissertation is summarised and discussed. Statistics for some proofs are given and ideas for future work conclude.

1.6 Notation

In this dissertation, the protocols are presented in standard notation. For each protocol step (which essentially sends a message), the step number, the sender and the intended recipient of the message, and the message itself are indicated. Messages will be indicated by fat braces, using a notation due to Paulson [88, §2.1], external braces being omitted. The ciphertext obtained by encrypting a cleartext message m with a cryptographic key K is indicated as $\{\!|m|\!\}_K$.

$$\begin{array}{lcccl} 1. & A & \rightarrow & B & : & N \\ 2. & B & \rightarrow & A & : & \{\!|N, N'|\!\}_K \end{array}$$

Figure 1.1: An example protocol

The invented example protocol in figure 1.1 serves to demonstrate the notation. That protocol consist of two steps. In the first step, agent A sends agent B the cleartext message N. In the second step, agent B replies to A with a ciphertext obtained by encrypting the concatenation of N and another message N' with a key K.

No further discussion on the meaning of N, N' or K is needed at this stage. All protocols will be studied assuming the worst conditions for the network: the spy can intercept all messages, prevent their delivery and tamper with them. Therefore, because the network is insecure, in general the receiver of a message is not necessarily its intended recipient.

Chapter 2

The Analysis of Cryptographic Protocols

Several formal approaches may help in the task of analysing cryptographic protocols, but the findings may not be straightforward to interpret.

At the beginning of the new millennium, formal methods are commonly accepted as a significant contribution to the analysis of cryptographic protocols. We discuss the main ones (§2.1) and provide a few examples on how to interpret their findings (§2.2).

Belief logics give a formal representation of the beliefs that the peers derive during the execution of a protocol (§2.1.1), but fail to capture several protocol weaknesses. *State enumeration* techniques such as *model checking* verify exhaustively that a protocol model of limited size admits no attacks (§2.1.2). While *provable security* allows for a theoretical study of confidentiality (§2.1.3), other approaches also attempt reasoning on the goal of authentication (§2.1.4) but usually lack mechanised support. The Inductive Approach, which is discussed in the next chapter, seems to embody the most desirable features: it can reason about a variety of protocol goals on models of unbounded size, and is mechanised.

Interpreting the contributions of protocol analysis is not straightforward. We present examples of findings that are: easy to obtain and interpret, on the TMN protocol (§2.2.1); fairly easy to obtain but fairly difficult to interpret, on the Woo-Lam protocol (§2.2.2); difficult to obtain but easy to interpret, on the public-key Needham-Schroeder protocol (§2.2.3); difficult to obtain and interpret, on the shared-key Needham-Schroeder protocol (§2.2.4). The list does not attempt to be exhaustive but merely to highlight that some of the issues concerning cryptographic protocols and their analysis are not necessarily straightforward, although they may, at first, appear to be so.

2.1 Formal Approaches

This section presents the main formal approaches developed throughout the last decade towards analysing cryptographic protocols. The treatment aims at being informative rather than detailed or exhaustive. Appropriate references are pointed out.

2.1.1 Belief Logics

The belief logic due to Burrows et al. [31] is the first attempt to exceed the limits of informal reasoning. The idea of their formal approach is representing the beliefs that the agents running a protocol derive at the various stages of the execution. Each step of the protocol is idealised as an (initial) logical formula, while a set of logical postulates is provided. The formulae that, using the postulates, can be derived from the initial ones formalise the goals of the protocol. We present here three typical formulae and quote their semantics [31, p.236]:

$P \mid\equiv X$: "P would be entitled to believe X. In particular, the principal P may act as though X is true." It is often denoted also as $P\,\mathsf{believes}\,X$.

$P \xleftrightarrow{K} Q$: "P and Q may use the shared key K to communicate. The key K is good, in that it will never be discovered by any principal except P or Q, or a principal trusted by either P or Q."

$\#(X)$: "the formula X is fresh, that is, X has not been sent in a message at any time before the current run of the protocol."

The semantics provided by the authors turned out to be unsatisfactory and several attempts were made to repair to the problem [5, 26]. For example, it is not clear from the first formula if P might believe something that is in fact false. The BAN logic indeed claims correct some protocols that were subsequently found to be flawed, such as a variant of the Otway-Rees protocol that does not encrypt the nonce issued by the protocol responder. An attack is possible, whereby the protocol initiator would share a session key with the spy at completion of the protocol while believing to be sharing it with the intended responder [87, §3.8]. This is both a violation of authentication and confidentiality. Another example is the public-key Needham-Schroeder protocol, which the logic analyses detecting no troubles: Lowe discovered a few years later that the protocol suffers a subtle failure of authentication [62].

The second formula has been considered ambiguous because it embodies the goals of both confidentiality and key distribution. The third formula is crucial to the analyses. For example, it is used to specify that the shared-key Needham-Schroeder protocol does not guarantee the protocol responder that the received session key is fresh. In fact, Denning and Sacco had pointed out that the spy might fool the responder into accepting an expired session key as a fresh one [36].

A major limitation of the approach is reasoning about confidentiality. This is done informally on top of the formulae derived through the calculus of the logic, since no malicious entity is modelled explicitly. A number of extensions have been designed in order to enhance the expressiveness of the logic and account for further protocol goals [1, 44, 69] but they tend to sacrifice the intuitive nature of the logic itself. We believe that confidentiality has never received an adequate treatment in this setting.

Proofs by belief logics are typically short and carried out by hand, but certain logics have been implemented [29, 30] using the theorem prover HOL [45].

2.1.2 State Enumeration via Model Checking

The well known process calculus CSP [49] has had vast applications in the field of formal methods thanks to its intuitive notions of *process* and *channel*. This setting easily scales up to the analysis of cryptographic protocols [97], as pioneered by Ryan [95]. The approach is mainly targeted at detecting possible attacks from the spy.

The peers are modelled as processes who can exchange the messages encompassed by the protocol via specific channels. This idealised specification, which accounts for no malicious entity, is certainly not flawed because no-one tries to mount attacks. Then, another specification is obtained by introducing the spy as a new process which can perform illegal operations. If this specification is *equivalent* to the idealised one, then the protocol is claimed to suffer no attacks.

More precisely, the two specifications are considered equivalent if all the states that are reachable by the second can be also reached by the first. Checking this by pen and paper is long and tedious, so a model checker can be tailored to enumerate the reachable states. Lowe employs FDR [94]. However, the process is then constrained by the intrinsic limitations of model checking: only finite systems of reasonably small size can be tackled. Despite the various techniques existing to loosen this limit as much as possible [73], the model protocols still are very small. They typically account for at most three or four agents, including the spy.

Many attacks have been discovered by model checking techniques [64, 67]. However, if the system of limited size does not suffer any attacks, it is not obvious that neither does the system of arbitrary size. This result has been proved by pen and paper on a specific protocol [66], while another model checker, the NRL Protocol Analyzer [75], also allows mechanised proofs by induction that certain states are out of reach. Nevertheless, no protocol goals have been discussed in detail except confidentiality and, later, authentication [65] (see §4.6). If a protocol cannot be attacked, it does not necessarily mean that all its goals are achieved.

Other model checkers that have achieved results in the field are ASTRAL [35], Murphi [37, 79], SMV [33] and SPIN [10]. More recently, also STEP [104] has been used to analyse the public-key Needham-Schroeder protocol. Some of these checkers are based on temporal logic rather than process calculus.

2.1.3 Provable Security

Provable security is a complexity-theoretic study of confidentiality. Originally developed by Bellare and Rogaway to address the problem of two-agent authenticated key exchange [23], the notion was later extended by the same authors to cope with the three-agent setting where a trusted server helps to achieve the goal of distributing session keys to a pair of peers [24]. They comment that devoting the entire efforts of formal analysis towards establishing that a protocol suffers no attacks is unsatisfactory: “there is finally a general consensus that session key distribution is not a goal adequately addressed by giving a protocol for which the authors can find no attacks” [24, §1.2].

In this setting, Bellare and Rogaway formally define the problem of session key distribution, design a cryptographic protocol and prove it *secure* assuming the existence of a family of pseudo-random functions (PRFs). To show that this assumption is minimal, they prove that if a secure session key distribution protocol exists, then a one-way function exists; then they apply the existing result stating that a PRF family exists if a one-way function exists [48].

The protocol is claimed secure in terms of two properties. The first is key distribution, signifying that both peers share the same session key at completion of the session. The second is the spy's negligible advantage on the discovery of the key, namely confidentiality of the key. All relevant proofs are carried out by hand and involve substantial formal complications. However, it is unquestionable that the treatment is a fine piece of theory.

Shoup and Rubin later extend the approach with an account of smart cards and design a new protocol based on smart cards that they prove secure. Their treatment suggests that provable security cannot express detailed but crucial requisites of the goal of confidentiality (see chapter 9).

To our knowledge, the approach has been tested only on very few protocol designs. However, it already shows limited flexibility in the sense that the actual protocol design must be adapted for the analysis, raising the risk of verifying a different design. Shoup and Rubin comment on this, while verifying someone else's protocol, and state 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" [98, §2.2]. However, the implementors of the Shoup-Rubin protocol still point out that "the details of Shoup-Rubin are fairly intricate, in part to satisfy the requirements of an underlying complexity-theoretic framework" [52].

2.1.4 Others

More recent approaches use specific strategies to verify the goals of confidentiality and authentication. The main ones, which are presented below, lack mechanised support.

Spi Calculus

The Spi Calculus [2, 3] is an extension of the more popular Π-Calculus [78] with primitives representing the cryptographic operations of encryption and decryption. Like other process calculi, it is based on processes that communicate through channels. Channels may be *restricted* in the sense that only certain processes may communicate on them. The Π-Calculus and the derived Spi Calculus allow the scope of each such restriction to dynamically change during the computation. A process may decide to send a message on a restricted channel to a process outside the scope of the restriction. When this happens, the scope is said to *extrude*.

It is intuitive to use the restricted channels to model confidentiality and the scope extrusion to allow for communication of secrets. Once the spy is modelled as an arbitrary environment for the protocol, confidentiality of a message X is proved as an equivalence of two protocol specifications, one featuring X, another an arbitrary X'. The idea of the proof is that the presence of X does not influence the reactions of the environment. However, the details may be difficult to grasp as admitted by the authors: "If you get lost in the formal passages of the paper, the cleartext nearby may help — hopefully the informal explanations convey the gist of what is being accomplished" [2].

Strand Spaces

A recent approach [40] rests on the notion of *strand*, which records a protocol history from the viewpoint of a single peer. Therefore, a strand is the sequence of events (message sending or receiving) concerning a peer of a protocol. This differs from Paulson's notion of *trace* (see next chapter), which records a protocol history from the viewpoint of an observer who can see the entire network.

A *strand space* is an unspecified set of strands, some for the agents of the network, some for the spy. Expectedly, while the strands formalising the spy's illegal behaviour are independent from the protocol being specified, those for the protocol peers are not. More important for the sake of the verification is the notion of *bundle*, a set of traces that are sufficiently expressive to formalise a protocol session. Therefore, certain strands of a bundle send messages, other strands receive them. It is not clear to us whether the spy's strands might interfere with this, as would be desirable to model real-world scenarios in which the spy can prevent the delivery of messages.

The notions of fresh or unguessable components are elegantly modelled as constraints on the construction of strands. For example, there exists no strand in which the spy sends an unguessable component prior to its reception. Proofs are carried out by induction on a bundle and their philosophy is reasonably easy to grasp. The treatment, entirely carried out by pen and paper, is only applied to the three classical protocols Needham-Schroeder [40], Otway-Rees and Yahalom [39]. Applications to further protocols are expected.

Abstract State Machines

Gurevich's *Abstract State Machines*, ASMs in brief [28], (formerly known as *Evolving Algebras* [46]), are born as a general-purpose formalism that should be more flexible than a Turing machine but retain the same power. Therefore, the ASM thesis is that any computable program can be represented by a suitable ASM.

An ASM is essentially a first-order signature equipped with a program that is a set of if-then-else rules. Once an interpretation of the signature is provided, a static algebra originates, the *initial state* of the ASM. Such state can be updated by the rules of the program, which, updating certain elements of the signature, produce a new algebra, namely a new *state*. The admissible functions include *oracles*, whose interpretation is not influenced by the ASM program, but is provided by the environment at each state. As a consequence, the resulting ASM computational model is not linear but has a graph structure. The approach, which has been benchmarked on a large variety of real-world applications [27], also has a distributed variant where each agent can independently run his own program. We have tailored this variant to the analysis of cryptographic protocols.

Initially, we modelled the Kerberos IV protocol using stepwise refinements in the presence of the spy. Confidentiality at the more detailed level was investigated by simulation [21]. This was the first formal specification of the semantical aspects of the protocol, obtained from the substantial informal documentation provided by its designers. It has significantly simplified the modelling phase of our mechanisation with theorem proving (see chapter 6).

Then, we developed a general theory of messages and demonstrated it on the public-key Needham-Schroeder protocol. Proofs were conducted by induction but still by pen and paper [22]. However, integration with mechanised tools, a model checker [102] or a theorem prover [42], has now reached an advanced level of development and may be used to investigate further protocol goals.

2.2 Interpreting the Findings

We present a few examples of findings obtained from the analysis of well-known protocols. Those concerning the Woo-Lam protocol arise from informal reasoning, which confirms the importance of that approach. All others originate from formal approaches based either on model-checking or on theorem-proving. Some of the corresponding notation is quoted without explanation for the sake of demonstration.

2.2.1 On TMN

Let us consider the TMN protocol [101], which aims at distributing session keys for mobile communications (figure 2.1).

$$\begin{array}{llcll}
1. & A & \rightarrow & \mathsf{S} & : A, \mathsf{S}, B, e(K_A) \\
2. & \mathsf{S} & \rightarrow & B & : \mathsf{S}, B, A \\
3. & B & \rightarrow & \mathsf{S} & : B, \mathsf{S}, A, e(K_B) \\
4. & \mathsf{S} & \rightarrow & A & : \mathsf{S}, A, B, v(K_A, K_B)
\end{array}$$

Figure 2.1: The TMN protocol

The identifiers A, B, K_A, K_B, S respectively represent the protocol initiator, the responder, the keys they choose and the trusted server; the unary constant $e()$ denotes a standard encryption function that only the server is able to invert, and the binary constant $v()$ stands for bit-wise exclusive-or (Vernam encryption).

Nothing protects the messages so, while intercepting them, the spy can read and modify any components. The protocol does not enforce agent authentication. The spy may send the server a fake instance of the first message using a key chosen by herself rather than by A, and complete the protocol with B. Even if A has not taken part in the session with B, both B and the server will believe that the opposite is true at completion of the session. Precisely, B will rely on the key K_B for communicating with A.

$$\begin{array}{l}
\langle \mathit{fake.Msg1.A.S.B.Encrypt.}K_C, \\
\ \mathit{comm.Msg2.S.B.A}, \\
\ \mathit{comm.Msg3.B.S.A.Encrypt.}K_B, \\
\ \mathit{intercept.Msg4.S.B.A.Vernam.}K_C.K_B, \\
\ \mathit{resp\_fake\_session.A.B.}K_B \rangle
\end{array}$$

Figure 2.2: An attack on the TMN protocol (CSP notation)

Therefore, this scenario [67, Attack 4.1], which can be spotted also while reasoning informally, is fairly easy to interpret. Its formalisation as a list of CSP events (figure 2.2) encompasses that the spy sends the first message on the channel *fake*, while the second and third messages are issued legally and travel on the channel *comm*. Finally, the spy prevents the delivery of the fourth message to A on the channel *intercept*, while B erroneously concludes that he shares the key K_B with A.

Similarly, the spy may intercept the second message, replace B's identity with hers in the third, and include in it a key chosen by herself rather than by B [67, Attack 4.2]. As a result, both A and the server would believe that B took part to the session, and A would erroneously think that the key invented by the spy is shared with B.

2.2.2 On Woo-Lam

The Woo-Lam protocol [103] aims at authenticating the initiator to the responder (figure 2.3). According to a standard notation, Ka in general indicates A's long-term key shared with the server, while Nb stands for the nonce issued by B. Authentication here means mere presence on the network.

$$\begin{array}{llcll}
1. & A & \rightarrow & B & : & A \\
2. & B & \rightarrow & A & : & Nb \\
3. & A & \rightarrow & B & : & \{\!|Nb|\!\}_{Ka} \\
4. & B & \rightarrow & \mathsf{S} & : & \{\!|A, \{\!|Nb|\!\}_{Ka}|\!\}_{Kb} \\
5. & \mathsf{S} & \rightarrow & B & : & \{\!|Nb|\!\}_{Kb}
\end{array}$$

Figure 2.3: The Woo-Lam protocol

After B receives, encrypted, the nonce that he issued for A, she forwards it to the server quoting A's identity. The server extracts the nonce, ultimately using A's shared key, and returns it to B. This signifies for B that the nonce was encrypted using A's key, and hence implies A's presence. The protocol does not attempt to convince B that A indeed meant to communicate with him, which would be a stronger guarantee (see §4.6).

$$\begin{array}{llcll}
1. & C & \rightarrow & B & : & A \\
1'. & C & \rightarrow & B & : & C \\
2. & B & \rightarrow & A & : & Nb \\
2'. & B & \rightarrow & C & : & Nb' \\
3. & C & \rightarrow & B & : & \{\!|Nb|\!\}_{Kc} \\
3'. & C & \rightarrow & B & : & \{\!|Nb|\!\}_{Kc} \\
4. & B & \rightarrow & \mathsf{S} & : & \{\!|A, \{\!|Nb|\!\}_{Kc}|\!\}_{Kb} \\
4'. & B & \rightarrow & \mathsf{S} & : & \{\!|C, \{\!|Nb|\!\}_{Kc}|\!\}_{Kb} \\
5. & \mathsf{S} & \rightarrow & B & : & \{\!|Nb''|\!\}_{Kb} \\
5'. & \mathsf{S} & \rightarrow & B & : & \{\!|Nb|\!\}_{Kb}
\end{array}$$

Figure 2.4: An attack on the Woo-Lam protocol

Abadi and Needham point out an attack whereby the spy impersonates

A when communicating with B [4, §4], so B's reasoning, seen above, is flawed. The spy (indicated by C in figure 2.4) may interleave two sessions with B. She uses A's identity in the first, but acts on her own behalf during the second session (distinguished by the primes). During this session, the spy cheats in the third step using the nonce Nb that B addressed to A. Therefore, the server's reply for the first session contains a new nonce Nb'', while the reply for the second session must contain Nb. At the end, B is misled into believing she has communicated with A.

However, one may wonder why B is not puzzled by receiving on a session the nonce that she issued on another session. It must be stressed that no agent can distinguish to which sessions the received messages belong, unless the contents of the messages state this. Therefore, receiving the nonce previously issued for A gives B the required "evidence" of A's presence although this is a wrong conclusion.

2.2.3 On Public-Key Needham-Schroeder

Lowe's "middle-person attack" [63] on the public-key Needham-Schroeder protocol [83] is a rather subtle one. It took one and a half decades to be discovered. The protocol (figure 2.5) exchanges the nonces Na and Nb in order to mutually authenticate A and B. Since A receives in the second step

$$\begin{array}{lcccl}
1. & A & \rightarrow & B & : \{\!|A, Na|\!\}_{Kb} \\
2. & B & \rightarrow & A & : \{\!|Nb, Na|\!\}_{Ka} \\
3. & A & \rightarrow & B & : \{\!|Nb|\!\}_{Kb}
\end{array}$$

Figure 2.5: The public-key Needham-Schroeder protocol

the nonce that she encrypted using B's public key, then B was alive. Upon reception of his own nonce in the third step, B draws the same conclusion about A. Lowe employs model-checking techniques to show how the spy can exploit two interleaved runs and interpose between the peers so that B believes that his peer is A when it is, in fact, the spy. The details of the attack are omitted here as they are well known. It does not violate authentication in terms of aliveness (it is true that A is alive, as B believes), but a stronger form of authentication, "weak-agreement" (see §4.6), whereby B should be assured that A intends to communicate with him. It appears that formal methods may be the way to detect the subtle consequences of session interleaving. Moreover, by showing that the attack can be avoided simply by including B's identity in the second step, Lowe points out the importance of explicitness, later investigated more deeply by Abadi and Needham [4].

Paulson formally verifies [88, §5.4] that, if B sends the second message to A and someone sends an instance of the third, then B is assured that A has sent an instance of the first message to someone. This conveys A's aliveness from B's viewpoint, but does not convey weak agreement of A with B. The peers' viewpoints become crucial when investigating *positive* guarantees (see §2.2.4), which establish the beliefs on which the peers rely.

2.2.4 On Shared-Key Needham-Schroeder

Needham and Schroeder also proposed a key distribution protocol [83]. Based on symmetric encryption, the protocol presupposes that each agent shares a long-term key with the trusted server (figure 2.6).

$$\begin{array}{rcccl}
1. & A & \rightarrow & \mathsf{S} & : & A, B, Na \\
2. & \mathsf{S} & \rightarrow & A & : & \{\!|Na, B, Kab, \{\!|Kab, A|\!\}_{Kb}|\!\}_{Ka} \\
3. & A & \rightarrow & B & : & \{\!|Kab, A|\!\}_{Kb} \\
4. & B & \rightarrow & A & : & \{\!|Nb|\!\}_{Kab} \\
5. & A & \rightarrow & B & : & \{\!|Nb-1|\!\}_{Kab}
\end{array}$$

Figure 2.6: The shared-key Needham-Schroeder protocol

A replay attack on the protocol is well-known. The spy may intercept the cipher sent in the third step and replay it to B without understanding its contents. As a result, B may be fooled into accepting an old session key as fresh.

Paulson verifies by induction that the protocol guarantees the confidentiality of the session key issued by the server, provided that both peers' shared keys are safe from the spy [84]. More formally, the main assumption of this theorem requires that the event

$$\mathsf{Says\ Server}\ A\,(\mathsf{Crypt}(\mathsf{shrK}\ A)\{\!|NA, \mathsf{Agent}\,B, \mathsf{Key}\,Kab, X|\!\}) \qquad (2.1)$$

occurs. However, the theorem must be interpreted. Can the peers who are intended to use the session key take advantage of it? In general, no agent can verify any events concerning other agents because they take place at other points in the network. Therefore, the theorem is only applicable by the server or some super-agent who sees all occurring events. It is not applicable either by A or by B, who cannot inspect the server's activity unless we establish some suitable lemmas on assumptions that they can verify. In particular, we can prove that, if A's shared key is safe from the spy and the certificate

$$\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|NA, \mathsf{Agent}\,B, \mathsf{Key}\,Kab, X|\!\}$$

appears in the traffic, then the event (2.1) occurred. Then, Kab is confidential by application of the previous theorem. The resulting theorem can be applied by A because, if she ever receives an instance of the second message, the message was in the traffic.

Similarly, the confidentiality argument can be made useful to B by the following lemma. If B's shared key is safe from the spy and the certificate

$$\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Key}\,Kab, \mathsf{Agent}\,A|\!\}$$

appears in the traffic, then the event (2.1) occurred for some NA. Consequently, when B receives an instance of the third message, he considers the session key found inside it to be confidential.

Following these considerations, we emphasise that it is the peers running the protocol who need guarantees that the protocol goals are met. This leads to the principle of goal availability (see §4.8). Any formal reasoning must be devoted to finding suitable evidence for the peers, in terms of assumptions that they can verify, that enforces these goals. Model-checking techniques typically exhibit a protocol version that suffers an attack, and a strengthened one that does not: it is not clear to us whether the latter version makes the guarantee available to the peers, or whether the reasoning can account for this.

Chapter 3

The Inductive Approach

A detailed analysis of security protocols can be performed by means of the Inductive Approach and the support of the interactive theorem prover Isabelle.

The informal way of establishing a protocol property is to verify that it is preserved by all protocol steps. If not complex *per se*, the process tends to reach unmanageable size because a protocol is a distributed concurrent program that is executed by an indefinitely large population of agents. The aim of preserving a property through various steps inspired Paulson's idea of proving the property formally via structural induction on an unbounded protocol model.

Paulson considers a security protocol, $\mathcal{P}$, and an unlimited population of agents. In particular, the agents include a 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 actions performed by the agents while they are executing $\mathcal{P}$. Each agent can interleave an unlimited number of protocol sessions. A history of the network traffic may be represented by the list of the events occurred, which is a *trace*. Paulson proposes the set P of all possible (finite or infinite) traces as an operational model for the network where $\mathcal{P}$ is executed. Generally, by abuse of terminology, P is referred to as the *formal protocol model for* $\mathcal{P}$. The set P is defined inductively by specific rules drawn from $\mathcal{P}$. The events occur via the firing of these rules. Since no rule is forced to fire, no event is forced to occur.

The use of induction in analysing security protocols features in the work of Meadows [75], who combines state-enumeration on a system of limited size with inductive proofs that the system never reaches infinite sets of states. However, Paulson's Inductive Approach [88] is entirely inductive: induction is employed to define the protocol model and all operators, and provides the strategy to prove any properties of the model. The approach is mechanised by the interactive theorem prover *Isabelle* [85], which supports inductively defined sets, provides simplification by conditional rewrite rules and performs automatic case splits for if-then-else expressions

This chapter introduces the main features of the Inductive Approach prior to our extensions: the basic formal structures (§3.1), the formalisation of the spy's knowledge (§3.2), the operators for the traces (§3.3) and the principles for constructing the formal protocol model (§3.4).

3.1 Basics

The free type key is introduced to represent cryptographic keys. The Isabelle datatype definition provides a compact and easily extensible way to define type constructors that are injective and have disjoint ranges. Paulson uses it to define types agent (§3.1.1), msg (§3.1.2) and event (§3.1.3), providing the basic structures to model most security protocols.

In the symmetric-key setting, each agent is endowed with a long-term key that is shared with the server

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

while two functions are defined respectively for the private and the public keys of each agent (omitted here) when encryption is asymmetric. In the former setting, a cryptographic key is either a shared key or, if it does not belong to the range of the function shrK, a session key. Also, an unspecified set bad of *compromised* agents have revealed their respective shared keys to the spy since the beginning of the protocol. They have also agreed to reveal the notes taken throughout the protocol sessions.

The remainder of this section will show the actual Isabelle syntax for the basic types.

3.1.1 Agents

Although finite, the population of agents running a security protocol in the real world is indefinitely large. Therefore, modelling any limited population would prevent the protocol model from capturing some potentially realistic scenarios.

An infinite population can be easily modelled by establishing a bijection with the set of natural numbers. Given a number i, $\mathsf{Friend}\,i$ is the corresponding agent [88, §3.1]. The malicious agent is modelled by the nullary constructor Spy, which will serve to investigate the protocol robustness towards illegal activity. Most symmetric-key protocols rely on a trusted third party, Server (often abbreviated as S), which has access to all agents' long-term secrets.

```
datatype agent = Server | Friend nat | Spy
```

In particular, Kerberos IV [77] has two trusted parties, so the datatype must be extended accordingly (see §6.2). By abuse of terminology, all agents except the spy will be often addressed as "friendly". Recall that we view agents as processes.

3.1.2 Messages

The original datatype for messages [87, §3.1] only included six constructors, which were subsequently extended by our introduction of guessable numbers to model timestamps (see §5.1) [88, §3.1]. The basic constructors allow agent names, nonces and cryptographic keys.

```
datatype msg = Agent agent
             | Nonce nat
             | Key   key
             | Mpair msg msg
             | Hash  msg
             | Crypt key msg
```

The recursive ones introduce compound messages, hashed messages and ciphers; $\mathsf{MPair}\,X_1 \ldots (\mathsf{MPair}\,X_{n-1}\,X_n)$ is abbreviated as $\{\!|X_1, \ldots X_{n-1}, X_n|\!\}$.

Encryption is assumed to be perfect: there is no rule to extract X from $\mathsf{Crypt}\,KX$ unless K is available. Also, encryption is assumed to introduce enough redundancy to be collision-free. This obviously does not hold of certain encryption schemes such as exclusive-or, so, attempts to allow for them have required alternative formalisations to the datatype [38].

3.1.3 Events

The events of the model allow message sending and message noting. The latter is used when agents need to note down portions of messages they receive (see the analysis of TLS [88]), or when the spy learns a session key by some agent's inaccuracy.

```
datatype event = Says  agent agent msg
               | Notes agent       msg
```

We will introduce a third event to model message reception and derive a formalisation of agents' knowledge (§7.2). Additional events will be necessary for modelling protocols based on smart cards (§8.2).

A list of events, namely a trace, records all those events occurred during a certain history of the network. Traces will be the parameter for the construction of the protocol model and for the verification of its properties.

3.2 Spy's Knowledge

Modelling the spy's knowledge inductively requires devising her initial knowledge. For this purpose, Paulson introduces the function [88, §3.5]

$$\mathsf{initState} \ : \ \mathsf{agent} \longrightarrow \mathsf{msg\ set}$$

which also models agents different from the spy's initial knowledge, although this is not useful to the rest of the treatment.
The server's initial knowledge consists of all agents shared keys (namely, all long-term secrets).

$$\mathsf{initState\ Server} \triangleq \{\mathsf{Key}\,(\mathsf{shrK}\,A)\}$$

Friendly agents' initial knowledge consists of their respective shared keys.

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

The spy's initial knowledge consists of all compromised agents' shared keys.

$$\mathsf{initState\,Spy} \triangleq \{\mathsf{Key}\,(\mathsf{shrK}\,A) \mid A \in \mathsf{bad}\}$$

The knowledge that the spy obtains from traces, expressed by the function

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

is defined as follows.

0. The spy knows her initial state.

   $\mathsf{spies}\,[\,] \ \triangleq \ \mathsf{initState\,Spy}$

1. The spy knows all messages ever sent on a trace.

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

2. The spy knows all compromised agents' notes.

   $$\mathsf{spies}\,((\mathsf{Notes}\,A\,X) \# \mathit{evs}) \ \triangleq \ \begin{cases} \{X\} \cup \mathsf{spies}\ \mathit{evs} & \text{if } A \in \mathsf{bad} \\ \mathsf{spies}\ \mathit{evs} & \text{otherwise} \end{cases}$$

Note that spies *evs* contains the entire network traffic occurred during the history recorded by *evs*. As a consequence, by abuse of terminology, spies *evs* is often referred to as *traffic on evs*.

3.3 Operators

Three operators are introduced to manipulate sets of messages

$$\mathsf{parts}, \mathsf{analz}, \mathsf{synth} : \mathsf{msg\ set} \longrightarrow \mathsf{msg\ set}$$

Their formal definition [88, §3.2] is omitted here. Given a set H of messages, $\mathsf{parts}\,H$ contains H and all messages recursively extracted from messages in H by projection and decryption. A similar definition builds $\mathsf{analz}\,H$, but decryption is performed only when the corresponding key is recursively available. In brief, while the former operator assumes readable ciphers, the latter does not. Thus, given a message M and a trace evs,

$$M \in \mathsf{parts}(\mathsf{spies}\ evs)$$

signifies that M appears in the traffic on evs possibly as component of a larger message, while

$$M \notin \mathsf{analz}(\mathsf{spies}\ evs)$$

expresses that the spy is not able to extract M, using the keys that she knows, from the messages on evs. The latter means that M is confidential on evs.

The definition of analz is somewhat more restrictive than that of parts. Indeed, it follows that

$$\mathsf{analz}\,H \subseteq \mathsf{parts}\,H$$

When H is the traffic on a trace, the theorem signifies that the components that are available to the spy are a subset of the entire traffic.

If H is the set of messages known to the spy, she can either forward its elements or use them to build compound messages and ciphers sealed by keys in H while adding agent names (which are publicly known). These messages constitute the set $\mathsf{synth}\,H$.

Another operator conveys the notion of freshness on a trace

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

A message M is considered fresh on a trace if it never appears either as a component of any agent's initial state or of a message recorded by the trace. So, used is defined as follows

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

– $\mathsf{used}((\mathsf{Says}\,A\,B\,X)\ \#\ evs) \quad \triangleq \quad \mathsf{parts}\{X\} \cup \mathsf{used}\ evs$

– $\mathsf{used}((\mathsf{Notes}\,A\,X)\ \#\ evs) \quad \triangleq \quad \mathsf{parts}\{X\} \cup \mathsf{used}\ evs$

3.4 Protocol Model

The formal protocol model is a set of lists of events, namely a set of traces. It is defined inductively and is unbounded because so are Isabelle lists.

The base case of the definition states that the empty trace belongs to the set. All other rules formalise the inductive steps. Each of them states how to extend a given trace of the set by all possible events whereby the protocol sends a new message in the traffic. For example, if a protocol has four steps, we will have four inductive rules (see §5.3).

Another rule, *Fake* lets the spy send all messages that she can fake. So, if *evs* is a trace of the set, then also the concatenation of $\mathsf{Says}\,\mathsf{Spy}\,B\,X$ with *evs* must be a trace of the set, X being drawn from $\mathsf{synth}(\mathsf{analz}(\mathsf{spies}\ evs))$.

When modelling key-distribution protocols, rule *Oops* is typically used to allow for the accidental loss of session keys to the spy. The rule introduces an event, referred to as *oops event* below, whereby the spy notes a session key. This makes it possible to investigate how local breaches of security can affect the global security.

Properties of an inductively defined set can be established by the corresponding induction principle. The property must be verified against all rules that define the set. This generates long case analyses, which Isabelle mechanises efficiently.

Chapter 4

Verifying the Protocol Goals

The main goals of cryptographic protocols, which are only part of the overall security of a system, are discussed along with the strategies for proving them within the Inductive Approach.

Security protocols cover a large spectrum of applications ranging from banking services to e-mail and, ultimately, e-commerce, and are intended to achieve a number of goals, depending on the specific application. Thus, "security is not a simple boolean predicate" [6]. Bolstering this analogy, we may think of security as a *conjunctive-normal-form* formula. Some of its conjuncts represent protocol goals, while the remaining ones embody properties of the entire system where the security protocol runs. Devising the "security formula" for a specific application is matter of open research. It is not obvious how many and which conjuncts the security formula should have. For example, many LANs have suffered breaches of security because of badly-configured firewalls and some users saving important keys on their workstations, despite the fact that the keys were always protected by well-established security protocols whenever officially exposed to the external world. External attackers have merely overcome the firewalls and then read the workstation memories. This demonstrates that security is a concept that spreads across multiple vertices of a communication architecture and multiple levels of each vertex.

Furthermore, verifying whether a conjunct of a given security formula holds may be a challenging task. In particular, those representing the goals of security protocols have been studied by formal methods only during the last decade. For example, an agent may need evidence that a received message is reliable in all its components (*integrity*), or that the same session key cannot be received within two different messages (*unicity*), or that his peer is indeed meaning to communicate with him (*authentication*). The Inductive Approach allows a scrupulous study of these properties.

This chapter comments on how to verify the reliability of the protocol model by means of suitable theorems (§4.1) and introduces the regularity lemmas (§4.2), which were pioneered by Paulson [88, §4.3]. Then, it discusses the goals of authenticity, stressing its correlation with integrity (§4.3), unicity (§4.4), confidentiality (§4.5), authentication (§4.6) and key distribution (§4.7), while explaining the strategies for proving them by Isabelle and reporting a few limitations of the original approach. Finally, the *minimal trust* required by most of the proved guarantees is explained, along with the principle of *goal availability* (§4.8).

4.1 The Reliability of the Protocol Model

The problem of how close a model is to the system it should represent intrinsically limits any formal verification. When a model is based on theorem proving, suitable *reliability* theorems can highlight whether it suffers any discrepancies from the real system or behaves as it is expected. Their major outcome is increasing the significance that the subsequent theorems proved of the model have in the real world. We develop new reliability theorems establishing properties that hold until a certain event takes place on a trace. For this purpose, we define the function before in the sequel of this section. When verifying security protocols, the reliability theorems do not address any protocol goals directly.

The theorems that fall under this category cannot be enumerated exhaustively, as new ones may arise from specific protocols to analyse. Most of the results proved by Paulson on the theories of messages, events, and shared-key protocols must be regarded as reliability theorems. For example, if the spy has obtained a set H of messages from the observation of the traffic, she may extract message components from them by decomposing compound messages and decrypting the ciphers sealed under known keys. This process can be iterated until there are no more compound messages and no more intelligible ciphers. In the real world, at this stage, the spy cannot acquire new knowledge by repeating the previous process. The model conforms to this, as the analz operator, which performs the message analysis, can be proved to be idempotent

$$\mathsf{analz}(\mathsf{analz}\, H) = \mathsf{analz}\, H$$

Another important reliability result states that a cryptographic key that is fresh on a trace certainly is not a long-term (shared) key. This assures that when a fresh key is generated, it cannot clash with any agent's shared key. Indeed, the probability of this happening in the real world is negligible.

The "possibility property" [88, §4.1] states that the protocol model contains traces on which the event formalising the last step of the real protocol occurs. This must be considered a reliability theorem because it signifies that the model allows completion of the protocol.

Other theorems of this class state that if a friendly agent (including the server) sends a certain message of the protocol, then the components of the message can be specified. As a matter of fact, in the real world all agents other than the spy act according to known legal rules. For example, Paulson proves that the server of the shared-key Needham-Schroeder protocol only sends well-formed messages [84]. If *evs* is a trace of the protocol model containing

$$\mathsf{Says\ Server}\ A\,(\mathsf{Crypt}\,K'\{\!|NA, \mathsf{Agent}\,B, \mathsf{Key}\,Kab, X|\!\})$$

then

$$K' = \mathsf{shrK}\,A \quad \text{and} \quad Kab \notin \mathsf{range}\,\mathsf{shrK} \quad \text{and}$$
$$X = \mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Key}\,Kab, \mathsf{Agent}\,A|\!\}$$

The full proof consists of three Isabelle commands: the first makes the event of the assumption a premise of the inductive formula, the second applies induction, the third simplifies all subgoals. However, this theorem does not guarantee that the server's operation is entirely reliable. For example, it is not clear whether the session key is fresh, as it should be, when it is sent. To investigate this, we declare the function

$$\mathsf{before} : [\mathsf{event}, \mathsf{event\ list}] \longrightarrow \mathsf{event\ list}$$

so that before *ev evs* yields the subtrace of events that occur on a trace *evs* before the introduction of the event *ev*. Since all traces are extended in reverse, the head of a trace contains the most recent events. So, we scan the reversed *evs* and collect its elements until *ev* is found

$$\mathsf{before}\ ev\ \mathsf{on}\ evs \ \ \triangleq\ \ \mathsf{takeWhile}(\lambda z. z \neq ev)(\mathsf{rev}\ evs)$$

On the assumptions of the previous theorem, we have proved that the session key is fresh when the server issues it,

$$\begin{aligned}\mathsf{Key}\,Kab \notin \mathsf{used}(&\mathsf{before}\\ &(\mathsf{Says\ Server}\ A\,(\mathsf{Crypt}\,K'\{\!|NA, \mathsf{Agent}\,B, \mathsf{Key}\,Kab, X|\!\}))\\ &\mathsf{on}\ evs)\end{aligned}$$

which completes the argument about the reliability of the model server. The proof requires two general subsidiary lemmas, one stating that the set of elements used on a trace is the same as the set of elements used on the reversed trace, the other stating that, if an element is used on a subtrace, then it is also used on the trace from which the subtrace is derived.

When verifying protocols based on smart cards (§9.3.1), we will prove that both friendly agents and the server can only use their own smart card, provided that it has not been stolen by the spy. By contrast, the spy can use both her own card and a set of compromised cards. These are also reliability theorems.

Proving a theorem of this class is not difficult. The idempotence of analz is easily derived from its definition, while the possibility property is proved by "joining up the protocol rules in order and showing that all their preconditions can be met" [88, §4.1]. All remaining theorems, strictly depending on the specific protocol being verified, necessitate induction over the protocol rules. Then, the simplifier either terminates all subgoals or, alternatively, highlights the structure of the remaining ones. On these occasions, the Isabelle automatic tactic, which combines simplification and classical reasoning, concludes the proof.

4.2 Regularity

Paulson regards as *regularity* lemmas all laws that can be proved on the assumption that some message appears in the traffic [88, §4.3]. More concretely, if a trace *evs* of the protocol model is such that

$$X \in \mathsf{parts}(\mathsf{spies}\ evs)$$

then some conditions can be proved about X on *evs*. By such a broad definition, many of the theorems proved of the protocol model must be regarded as regularity lemmas. In particular, the class includes those theorems assessing the originator of a certificate that is on the traffic, and those that formalise the goals of authenticity and authentication (see below).

A *basic* regularity lemma can be stated for each kind of long-term key. If a shared-key protocol never requires the agents to send their shared keys in the traffic, then, given a trace *evs* of the protocol model, an agent's key appearing in the traffic on *evs* implies that the agent is compromised. The proof applies induction and shows that the key could appear in the traffic only in the *Fake* case, namely when it is the spy who uses it, because the key owner is compromised. By definition of initState, spies and parts, the same result holds in the opposite direction regardless of the protocol being analysed. Combining the two implications, the basic regularity lemma for shared keys is obtained

$$\mathsf{Key}(\mathsf{shrK}\,A) \in \mathsf{parts}(\mathsf{spies}\ evs) \iff A \in \mathsf{bad}$$

However, the major relevance of the theorem arises when it is expressed in terms of analz

$$\mathsf{Key}(\mathsf{shrK}\,A) \in \mathsf{analz}(\mathsf{spies}\ evs) \iff A \in \mathsf{bad}$$

The left-to-right implication holds because of $\mathsf{analz}\,H \subseteq \mathsf{parts}\,H$ and the regularity lemma; the opposite holds by definition of initState, spies and analz. By abuse of terminology, we still address this result as regularity lemma in the sequel. Its importance lies in translating a condition that an agent is certainly *not* able to verify — the spy learning his key from the analysis of the traffic — into one that he *could* be able to verify — his being compromised (see §4.8). In other words, the lemma says that the spy knows a shared key from analysing the traffic induced by the protocol if and only if she knows it initially. For example, let us consider a certificate $\{\!|X|\!\}_{Ka}$, meant for A and sealed by her shared key. If we want to prove any properties about the certificate, then it must be tamperproof against the spy. This, in turn, requires A's key not to be available to the spy, which, by the regularity lemma, is equivalent to A not being compromised.

The regularity lemma concerns the private keys in case *evs* is a trace of the model for a public-key protocol.

4.3 Authenticity

If a message that appears to have originated with a certain agent did indeed originate with the agent, then the message enjoys *authenticity*. The ISO Security Architecture framework [50] distinguishes authenticity from *integrity*, which holds of a message that is proved to be received in the same form as it was generated.

However, many researchers consider the source of a message as essential part of the message. Therefore, verifying that the message is unaltered when it is received (integrity) confirms its originator (authenticity). Conversely, discovering the originator of a message that is received also confirms that the message is unaltered. To this extent, the two properties may be considered equivalent.[1]

[1]Private conversation with Dieter Gollmann.

Our proofs support this viewpoint. For example, let us consider the ticket of the shared-key Needham-Schroeder protocol (§2.2.4), $\{\!|Kab, A|\!\}_{Kb}$, which is created by the server. The integrity of the ticket is equivalent to preventing the spy from knowing Kb and thus, via the regularity lemma, to B not being compromised. Under this assumption, Paulson proves that the ticket originated with the server [84], and hence its authenticity. So, the ticket integrity implies its authenticity.

The converse also holds. Trying to prove the ticket authenticity, permitting the spy to know Kb, leaves the following subgoal, which arises from the case *Fake*

```
[| evsF ∈ ns_shared;
   Crypt (shrK B) {|Key Kab, Agent A|} ∉ parts (spies evsF);
   Crypt (shrK B) {|Key Kab, Agent A|} ∈ synth (analz (spies evsF)) |]
⟹ ∃ NA. Says Server A (Crypt (shrK A) {|NA, Agent B, Key Kab,
                                  Crypt (shrK B) {|Key Kab, Agent A|}|})
           ∈ set evsF
```

The second and third assumptions signify that, although the ticket does not appear on the traffic on $evsF$, the spy can synthesise it from the analysis of that traffic. The symbolic evaluation of synth states that two cases are, in general, possible. Either the spy merely forwards the ticket that she obtains from the analysis of the traffic, namely the ticket belongs to $\mathsf{analz}(\mathsf{spies}\, evsF)$; or the spy can handle all components necessary to fake the ticket. The former is impossible because it contradicts the second assumption of the subgoal (since $\mathsf{analz}\, H \subseteq \mathsf{parts}\, H$). According to the latter case, simplification and the regularity lemma transform the third assumption in the following pair

$$\mathsf{Key}\, Kab \in \mathsf{analz}(\mathsf{spies}\, evsF) \quad \text{and} \quad B \in \mathsf{bad}$$

The resulting subgoal can be falsified because it assumes a non-contradictory scenario in which an agent and a session key are compromised to the spy. So, she is able to forge the corresponding ticket $\{\!|Kab, A|\!\}_{Kb}$ before the server issues it legally. In this scenario, when the ticket appears in the traffic, it is not necessarily authentic.

In the light of these considerations, we will regard authenticity and integrity as a single concept denoted by the former term. Also, when proving message authenticity, we will in general make the assumptions that appear to prevent the spy from faking the message, and will attempt to enforce the event corresponding to the protocol step that creates the message. If we are dealing with a certificate sealed by a long-term key, then, by application of the corresponding regularity lemma, it will suffice to assume that the key owner is not compromised. However, further assumptions may be required to investigate the authenticity of specific message components such as the "pairkey" used by the Shoup-Rubin protocol (see §9.3.3).

The introduction of message reception (§7.2) will allow a more realistic formalisation of the authenticity theorems from the agents' viewpoints.

4.4 Unicity

Security protocols often involve the creation of fresh components such as nonces or session keys. Since the same component cannot be created as fresh more than once, it is uniquely bound to its message of origin. These observations inspired the *unicity* theorems [88, §4.4], establishing that, if two events containing a certain component occur, then the events are identical. We investigate the argument in deeper detail, proving that certain events cannot occur more than once. We achieve this aim below, defining the predicate unique.

The Yahalom protocol, for example, requires the server to issue a fresh session key Kab for two peers A and B [90]. So, if evs is a trace of the Yahalom model containing the events

$$\begin{aligned}&\textsf{Says Server } A\ \{\!|\textsf{Crypt}(\textsf{shrK}\,A)\{\!|\textsf{Agent}\,B, \textsf{Key}\,Kab, Na, Nb|\!\}, X|\!\} \quad \text{and}\\ &\textsf{Says Server } A'\ \{\!|\textsf{Crypt}(\textsf{shrK}\,A')\{\!|\textsf{Agent}\,B', \textsf{Key}\,Kab, Na', Nb'|\!\}, X'|\!\}\end{aligned}$$

then

$$A = A' \quad \text{and} \quad B = B' \quad \text{and} \quad Na = Na' \quad \text{and} \quad Nb = Nb'$$

An initiator A of the public-key Needham-Schroeder protocol (§2.2.3) has to issue a fresh nonce Na and include it in the first message. Therefore, if the nonce is not available to the spy and evs is a trace of the protocol model such that

$$\begin{aligned}&\textsf{Crypt}(\textsf{pubK}\,B)\{\!|\textsf{Nonce}\,Na, \textsf{Agent}\,A|\!\} \in \textsf{parts}(\textsf{spies}\,evs) \quad \text{and}\\ &\textsf{Crypt}(\textsf{pubK}\,B')\{\!|\textsf{Nonce}\,Na, \textsf{Agent}\,A'|\!\} \in \textsf{parts}(\textsf{spies}\,evs)\end{aligned}$$

then

$$A = A' \quad \text{and} \quad B = B'$$

Should Na be available to the spy, the certificates of the theorem could have been created by the spy. Alternatively, both B and B' should be assumed not to be compromised in order to apply the corresponding regularity lemma.

Proving these theorems merely requires an inductive analysis of the protocol steps and a simplification of the arising subgoals. The critical case is the step where the fresh component is created. If a different component is introduced, then simplification terminates the subgoal; if the same component is introduced, then the freshness assumption concludes the proof.

However, while the former unicity theorem is only useful to the server, the latter may become useful to a non-compromised friendly agent B only upon reception of the certificates $\{\!|Na, A|\!\}_{Kb}$ and $\{\!|Na, A'|\!\}_{Kb}$. Modelling this scenario formally requires the introduction of message reception in the model (see §7.2). Once B has received the two certificates, should A differ from A', the theorem would be violated. Thus, B would suspect that something that lies outside the model happened, ranging from some failure of the underlying transport protocol to brute-force codebreaking.

We observe that the first form of unicity theorem still permits the server to send two identical messages. This would, all the same, violate the freshness assumption on the session key, so it should be impossible. To investigate the matter, we declare a predicate that takes as parameter an event and a trace,

$$\mathsf{unique} : [\mathsf{event}, \mathsf{event\ list}]$$

scans the trace until the event is found and skipped, and finally checks that the event does not occur on the remaining part of the trace

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

The predicate holds on an event ev and a trace evs when ev occurs only once on evs. As expected, we can prove that a trace evs of the Yahalom model is such that

$$\begin{aligned}&\mathsf{unique}\ (\mathsf{Says\ Server}\ A\ \{\!|\mathsf{Crypt}(\mathsf{shrK}\ A)\{\!|\mathsf{Agent}\ B, \mathsf{Key}\ Kab, Na, Nb|\!\}, X|\!\})\\&\mathsf{on}\ \ evs\end{aligned}$$

An equivalent result can be routinely proved by induction and simplification for all protocols analysed so far. The definition of the predicate must be used as a rewrite rule for the simplifier.

The analysis of protocols that are based on smart cards (see §9.3.4) will gain by the new theorem in case the protocols assume a secure means between agents and cards. Agents will receive further assurances about the cards' functioning.

4.5 Confidentiality

A protocol enforces *confidentiality* of M if it does not disclose M to the spy.

Paulson's *session key secrecy theorem* formalises session key confidentiality. For example, if A and B are not compromised and *evs* belongs to the Otway-Rees model and contains

$$\begin{aligned}&\mathsf{Says\ Server}\ B\,\{\!|Na, \mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|Na, \mathsf{Key}\,Kab|\!\},\\ &\qquad\qquad \mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|Nb, \mathsf{Key}\,Kab|\!\}|\!\}\end{aligned}$$

but does not contain an oops event on Kab involving the same nonces

$$\mathsf{Notes\,Spy}\ \{\!|Na, Nb, \mathsf{Key}\,Kab|\!\}$$

then

$$\mathsf{Key}\,Kab \notin \mathsf{analz}(\mathsf{spies}\ evs)$$

Proving this theorem [88, §4.6] requires evaluating the assertion for all the possible extensions of *evs* according to the protocol model. In each of these cases, spies extracts the new message, say X, leaving expressions of the form

$$\mathsf{Key}\,Kab \notin \mathsf{analz}(\{X\} \cup (\mathsf{spies}\ evs))$$

The symbolic evaluation rules for analz inspect X and pull out all components except the keys, leaving expressions of the form

$$\mathsf{Key}\,Kab \notin \mathsf{analz}(\{\mathsf{Key}\,K\} \cup (\mathsf{spies}\ evs))$$

for some session key K in the subgoal corresponding to the oops case, where a message containing a non-fresh session key is introduced. Symbolic evaluation cannot proceed any further because in general K may be used to encrypt Kab in some message of *evs*. However, perhaps the protocol prevents this. Proving such result, the *session key compromise theorem* [88, §4.5], provides the necessary rewriting rule

$$\begin{aligned}&\mathsf{Key}\,K' \in \mathsf{analz}(\{\mathsf{Key}\,K\} \cup (\mathsf{spies}\ evs))\\ &\qquad\qquad\qquad \Longleftrightarrow K' = K \vee \mathsf{Key}\,K' \in \mathsf{analz}(\mathsf{spies}\ evs)\end{aligned}$$

K being a session key. The result confirms that Otway-Rees never uses session keys to encrypt other keys, so the spy cannot exploit a stolen session key to learn others.

After simplification by the compromise theorem, the oops subgoal of the secrecy theorem terminates via the unicity theorem enforcing that the server issued *Kab* only with the nonces *Na* and *Nb*. The subgoal corresponding to the third step of the protocol, where the server issues the session key, is solved by freshness of the session key, and the remaining ones routinely.

Although the session key secrecy theorem constitutes the main confidentiality result, it is still not directly applicable by the agents. Further lemmas are necessary to this purpose (§2.2.4).

Confidentiality is often crucial also on nonces. For example, in the TLS protocol [89], the pre-master secret, *PMS*, is a nonce of fundamental importance because it is used to compute other nonces, session keys and MACs. Confidentiality of the *PMS* can be proved conventionally with the addition of a new rewriting rule for analz. If A and B are not compromised and *evs* is a trace of the TLS model containing

$$\mathsf{Notes}\,A\,\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,PMS|\!\}$$

then

$$\mathsf{Nonce}\,PMS \notin \mathsf{analz}(\mathsf{spies}\ evs)$$

The theorem signifies that once a non-compromised agent notes the *PMS*, it remains secure from the spy. As explained on the preceding result, the proof requires a suitable rewriting rule for analz

$$\mathsf{Nonce}\,N \in \mathsf{analz}(\{\mathsf{Key}\,K\} \cup (\mathsf{spies}\ evs)) \Longleftrightarrow \mathsf{Nonce}\,N \in \mathsf{analz}(\mathsf{spies}\ evs)$$

K being a session key, stating that session keys cannot be exploited to learn nonces.

The strategies presented in this section do not vary with the introduction of message reception or of smart cards into the model.

4.6 Authentication

Despite the fact that agent *authentication* is the main, claimed goal of many security protocols, there exists significant potential for confusion about the interpretation of this term [43]. A taxonomy due to Lowe may elucidate the matter identifying four levels of authentication. Let us suppose that an initiator A completes a protocol session with a responder B.

1. **Aliveness** of B signifies that B has been running the protocol.
2. **Weak agreement** of B with A signifies that B has been running the protocol with A.

3. **Non-injective agreement** of B with A on H signifies weak agreement of B with A and that the two agents agreed on the set H of message components.

4. **Injective agreement** of B with A on H signifies non-injective agreement of B with A on H and that B did not respond more than once on the session with A.

Note that each level subsumes the previous one. In particular, the injective agreement of B with A establishes an injective relation between B's runs of the protocol with A and A's runs with B. The existing authentication arguments carried out using the Inductive Approach do not set their findings within this taxonomy. Although authentication is typically interpreted as aliveness, we find that many guarantees also convey weak-agreement. However, investigating non-injective agreement requires extending the approach, as we explain below.

The Inductive Approach allows agents to respond more than once to a received message (provided that it is of the expected form) because the protocol models are meant to be as permissive as possible. Therefore, the strongest form of authentication that we can wish to prove is non-injective agreement. A generic rule of the protocol model takes a trace *evs* of the model, insists that some events occur on it and others do not, and establishes that the concatenation of a specific event *ev* with *evs* is still a trace of the model. Constraining agents to a single reply can be done by adding to the generic rule an extra assumption stating that *ev* does not occur on *evs*. While leaving the existing proofs unaltered, this would make it possible to investigate injective agreement. Lowe simply says that such property "may be important in, for example, financial protocols" [65, §2.4], but we are not aware of real-world protocols that have claimed it explicitly as a goal.

However, we find that the original Inductive Approach requires some extensions even to formalise non-injective agreement. With key-distribution protocols, for example, non-injective agreement on a session key is the relevant form of authentication. Paulson proves that, if A and B are not compromised and *evs* is a trace of the model for the shared-key Needham-Schroeder protocol (§2.2.4) containing

$$\mathsf{Says}\,B\,A\,(\mathsf{Crypt}\,Kab(\mathsf{Nonce}\,Nb))$$

and such that

$$\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Key}\,Kab, \mathsf{Agent}\,A|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs) \text{ and}$$
$$\mathsf{Crypt}\,Kab\{\!|\mathsf{Nonce}\,Nb, \mathsf{Nonce}\,Nb|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs)$$

but not containing an oops event on Kab,

$$\mathsf{Notes\,Spy}\,\{\!|N, N', \mathsf{Key}\,Kab|\!\}$$

for any N, N', then evs contains

$$\mathsf{Says}\,A\,B\,(\mathsf{Crypt}\,Kab\{\!|\mathsf{Nonce}\,Nb, \mathsf{Nonce}\,Nb|\!\})$$

The message $\{\!|Nb - 1|\!\}_{Kab}$ is formalised as $\mathsf{Crypt}\,Kab\{\!|\mathsf{Nonce}\,Nb, \mathsf{Nonce}\,Nb|\!\}$ (if the model spy could add or subtract 1, then she could spoof all nonces), so the assertion means that the last step of the protocol has taken place. Reviewing the theorem, we have discovered that the first assumption is superfluous. The general strategy for all theorems of this form appeals to authenticity (§4.3) and then derives the confidentiality of the session key (§4.5). Hence, if the certificate $\mathsf{Crypt}\,Kab\{\!|\mathsf{Nonce}\,Nb, \mathsf{Nonce}\,Nb|\!\}$ appears in the traffic, then it is integral, so induction proves it to have originated with A. This proof has not required assuming that B sent the other certificate, which can in fact be proved as a corollary.

Does the theorem establish non-injective agreement of A with B on Kab? Upon reception of the two mentioned certificates, and with the assumptions that A is not compromised and that Kab has not been leaked by accident (which is a *minimal trust*, see §4.8), B is informed that A sent him a cipher sealed by Kab. While B learns Kab from one of the certificates, A merely sending the cipher does not express A's knowledge of the key that seals it, so B is not informed of whether A agrees on Kab. In general, A might be just forwarding an unintelligible message previously received. By Lowe's definitions, the theorem only establishes weak agreement of A with B. However, the protocol inspection highlights that A is in fact the true creator of the cipher and therefore knows the key to seal it. In chapter 7, we will prove such insight formally by means of two different strategies.

4.7 Key Distribution

Key distribution is the main goal for many protocols. It is met when any two peers who complete a protocol session gain evidence that they share a session key with each other. There are also weaker interpretations of this goal, according to which a protocol meets key distribution when it just distributes the keys without giving the corresponding evidence to the peers. However, in the latter case, the goal is certainly not *available* to the peers according to the definition that we give in the next section (§4.8). Bellare and Rogaway state that key distribution is "very different from" agent authentication.

"The reason is that entity authentication is rarely useful in the absence of an associated key distribution, while key distribution, all by itself, is not only useful, but it is not appreciably more so when an entity authentication occurs along side. Most of the time the entity authentication is irrelevant" [24, §1.6].

We believe that key distribution and agent authentication are strictly related. We observe that mutual non-injective agreement on a session key is, in fact, a stronger goal than key distribution because the former implies the latter. Indeed, our proofs will establish key distribution via the authentication argument. Also, this is the only possible strategy towards key distribution on all protocols analysed so far, which supports the claim that the two goals are equivalent. For example, let us recall the second and third steps of the shared-key Needham-Schroeder protocol

$$\begin{array}{llcll} 2. & \mathsf{S} & \rightarrow & A & : \{\!|Na, B, Kab, \{\!|Kab, A|\!\}_{Kb}|\!\}_{Ka} \\ 3. & A & \rightarrow & B & : \{\!|Kab, A|\!\}_{Kb} \end{array}$$

Upon B's reception of the certificate $\{\!|Kab, A|\!\}_{Kb}$, the only strategy to show B that also A knows Kab is that of deriving that the certificate originated with A upon reception of the second message of the protocol, which delivered Kab to her. This also establishes non-injective agreement of A with B upon the session key. Formalising message reception is, once more, required to express this reasoning formally.

4.8 Minimal Trust and Goal Availability

The previous sections (and also §2.2.4) have stressed that formal guarantees can be applied by an agent only when the agent can verify their assumptions. This concept necessitates further consideration.

Let us recall the session key secrecy theorem (§4.5). Both peers must not be compromised in order to prevent the spy from knowing their shared keys and decrypting the ciphers that deliver the session key. Besides, the spy must not have learned the key by exploiting either peer's incaution via an oops event. These assumptions form the *minimal trust* because they cannot be verified by any agent, although they are indispensable in our highly permissive modelling. All guarantees that appeal to the confidentiality argument will be constrained by the minimal trust.

Reinterpreting this concept from a specific agent's viewpoint, we observe that an agent's minimal trust includes that the agent's peer is not compromised and that neither of them has leaked the key. Moreover, if the agent's long-term key is stored on some workstation, the spy may be able to get hold of it even before the agent participates in the protocol. Therefore, depending on the static protection of the workstations (e.g. in terms of firewalls), an agent's minimal trust may also have to include himself not being compromised.

Modelling timestamps and restricting the protocol model to only leak session keys that have expired will weaken the assumption on the oops event and therefore lower the minimal trust (see §5.5). Modelling smart cards will modify the minimal trust so to include that certain cards are not compromised (see §8.1.2).

We say that a guarantee is *applicable* by an agent, when it is established by a theorem whose assumptions, excepted those in the minimal trust, can be verified by the agent. If there exist guarantees about a certain goal g of a protocol $\mathcal{P}$ that are applicable, respectively, by both protocol peers, then $\mathcal{P}$ *makes* g *available to its peers*. We state the following principle of *goal availability*: a protocol must make its goals available to its peers. Investigating this principle reveals an attack on Kerberos IV (see §6.3.5) and important weaknesses of Shoup-Rubin (see §9.3.5).

Chapter 5

Modelling Timestamps

The Inductive Approach is extended with the treatment of timestamps and a timestamp-based protocol, BAN Kerberos, is modelled and verified.

The original Inductive Approach does not include timestamps among the formalised message components [87, §3.1] and is in fact only benchmarked on nonce-based protocols such as Otway-Rees and Yahalom.

At the beginning of the 1980s, Denning and Sacco pioneered the use of timestamps in the field of security protocols [36] to avoid replay attacks. Timestamps, which are numbers marking a specific instance of time, have been employed since then in many protocols such as BAN Kerberos [31, §6] and Kerberos IV [77]. We extend the Inductive Approach with the treatment of timestamps in order to analyse this new class of protocols.

The single operational difference between nonces and timestamps is that the latter can be guessed by the spy. So, we model timestamps as guessable numbers to include in the allowed message components. The price paid is limited to introducing the new message constructor, proving a few technical lemmas, and doing minor updates to some of the existing ones. The approach becomes significantly more general. The new message component is also used to model any extra information that the real-world protocols pass inside their messages, which is in general available to the spy. This is the case, for example, with the session identifier and other fields of the model for the TLS protocol [89].

The first benchmark we choose for the extended approach is the BAN Kerberos protocol, as its well-known analysis by BAN logic [31, §6] provides a significant opportunity for comparison. The BAN analysis concludes that, if A has completed a session of the protocol with B, then A is aware that

B meant to communicate with A using the session key Kab; the equivalent guarantee is offered to B

$$A \models B \models A \overset{Kab}{\longleftrightarrow} B \quad \text{and} \quad B \models A \models A \overset{Kab}{\longleftrightarrow} B$$

Since the logic omits investigating whether the agreement is injective, its conclusions may be viewed as mutual non-injective agreement on the session key. Our findings confirm this goal (via some extensions to the approach, see §7.3) and strengthen it with a deeper investigation of others such as authenticity and confidentiality [20]. The proofs, partially adapted from the existing ones on the shared-key Needham-Schroeder protocol, also suggest refining the treatment of the accidental losses of session keys. Our protocol model accounts for the temporal checks performed by the agents at each step, which helps in understanding how the protocol functions beyond the mere sequencing of messages.

The Kerberos project started at MIT during the mid 1980s [77] and, over a decade, generated several variants of the same protocol design [58]. BAN Kerberos is the natural modification of the shared-key Needham-Schroeder with the addition of timestamps. Therefore, it is interesting to compare the temporal requisites that the two protocols add to the goal of authentication. This is achieved later (§7.6) because the Inductive Approach must be suitably extended.

This chapter presents the formalisation of guessable numbers (§5.1) and introduces the BAN Kerberos protocol (§5.2), which is then modelled (§5.3) and verified (§5.4).

5.1 Modelling Guessable Numbers

The process of modelling guessable numbers consists of three phases. First, the Isabelle datatype for msg is extended by a new constructor Number that takes as its parameter a natural number. A timestamp T will be represented in the model by the message component $\mathsf{Number}\,T$.

Then, the inductive definition of $\mathsf{synth}\,H$ is updated by a rule that allows the spy to synthesise any number

$$\mathsf{Number}\,N \in \mathsf{synth}\,H$$

Finally, the main operators need suitable rewriting rules for the symbolic evaluations that involve the new component

$$\begin{aligned} \mathsf{parts}(\{\mathsf{Number}\,N\} \cup H) &= \{\mathsf{Number}\,N\} \cup (\mathsf{parts}\,H) \\ \mathsf{analz}(\{\mathsf{Number}\,N\} \cup H) &= \{\mathsf{Number}\,N\} \cup (\mathsf{analz}\,H) \end{aligned}$$

Minor updates necessary to a few existing lemmas are omitted here.

5.2 The BAN Kerberos protocol

BAN Kerberos [31, §6] is a key distribution protocol, namely it aims at distributing session keys to its peers (figure 5.1).

$$\begin{array}{lllll}
1. & A & \rightarrow & \mathsf{S} & : A, B \\
2. & \mathsf{S} & \rightarrow & A & : \{\!|Tk, B, Kab, \underbrace{\{\!|Tk, A, Kab|\!\}_{Kb}}_{ticket}|\!\}_{Ka} \\
3. & A & \rightarrow & B & : \underbrace{\{\!|Tk, A, Kab|\!\}_{Kb}}_{ticket}, \underbrace{\{\!|A, Ta|\!\}_{Kab}}_{authenticator} \\
4. & B & \rightarrow & A & : \{\!|Ta + 1|\!\}_{Kab}
\end{array}$$

Figure 5.1: The BAN Kerberos protocol

The design closely resembles that of the shared-key Needham-Schroeder protocol (§2.2.4) but rests on a different procedure of mutual authentication (see in particular the last two steps). The protocol associates a lifetime to session keys and another to authenticators. Lifetimes represent the time intervals within which the corresponding components should be considered valid. The lifetimes are passed in the messages but we omit them from the presentation, assuming they are known to all.

After A's initial request to establish a session with B, the server issues a fresh session key and includes it, along with the timestamp that marks its time of issue, inside a message sealed by A's shared key. The message also contains a ticket sealed by B's shared key, which in turn contains a duplicate of the session key and its timestamp. The message is sent to A, who removes the external encryption and learns that the session key Kab issued at time Tk is indeed meant for the session with B. Then, A checks that Tk has not expired to establish whether the session key is still valid. If so, A builds an authenticator with a new timestamp Ta and sends it with the ticket to B. Upon reception of this message, B decrypts the ticket, learns the session key for the session with A and its timestamp Tk. If Tk has not expired, B uses the session key to decipher the authenticator. This should give him evidence that A was alive and able to use the session key at time Ta. The same guarantee should be given to A by the certificate that B sends her in the final step of the protocol.

5.3 Modelling BAN Kerberos

The protocol uses the notion of time in two ways: it issues timestamps as the current time and checks timestamps against the current time. Only specific instants are considered, so a discrete sampling of the continuous time suffices for the modelling. We assume that there exists a global, secure clock that all agents rely on. Hence, we will equip each trace with a clock yielding the current time of the network history modelled by that trace.

Traces only grow linearly. If the event ev is located after the event ev' on a trace, then ev' in the real world occurred at some later, unspecified, time after ev did.[1] Hence, there exists an injective function between the set of all traces and the set of all samplings of time. A sampling corresponds to a trace if the first value of the sampling represents the time when the first event of the trace took place, the second value represents the time for the second event, and so on. The function is clearly not a bijection because there may exist different traces corresponding to the same sampling. The empty trace corresponds to the empty sampling.

A time-sampling can be *normalised* in terms of natural numbers: there exists an injective function between the set of all discrete samplings of time and the set of segments of natural numbers including zero. The empty sampling corresponds to the set $\{0\}$. Any sampling containing only one value corresponds to the set $\{1\}$; any sampling containing only two values corresponds to the set $\{1, 2\}$, and so on. Since the current time of a trace is the highest value of the corresponding time sampling, after the normalisation process the current time of a trace of length n is n. We declare the function

$$\mathsf{ct} : \mathsf{event\ list} \longrightarrow \mathsf{nat}$$

and define

$$\mathsf{ct}\, evs \;\triangleq\; \mathsf{length}\, evs$$

Recall that a trace of length n represents a history of the network during which n events have taken place. We believe it is intuitive to think that the current time of the trace is n.

We declare two natural numbers, $\mathsf{seskLife}$ and $\mathsf{authLife}$, to formalise respectively the lifetime of session keys and of authenticators. Agents check the timestamps against them and discard the messages containing expired timestamps. To formalise those checks, we declare two binary predicates

$$\mathsf{expiredS},\ \mathsf{expiredA} : [\mathsf{nat}, \mathsf{event\ list}]$$

[1]The trace model assumes that no two events occur simultaneously. This could be relaxed by defining a trace as a list of sets of events.

and define them as

$$\begin{aligned}\mathsf{expiredS}\ Tk\ \ evs &\equiv (\mathsf{ct}\ evs) - Tk > \mathsf{seskLife}\\ \mathsf{expiredA}\ Ta\ \ evs &\equiv (\mathsf{ct}\ evs) - Ta > \mathsf{authLife}\end{aligned}$$

When, for example, $\mathsf{expiredS}\ Tk\ evs$ holds, a longer time than $\mathsf{seskLife}$ has elapsed since Tk at the moment when the history registered by evs is examined. Therefore, the session key associated with Tk is no longer valid on evs (which in practice has become too long). The association between Tk and its session key is established by the structure of the second protocol message, so it does not need explicit formalisation.

The constant $\mathsf{bankerberos}$, declared as a set of lists of events, represents the formal protocol model and is defined by induction in figure 5.2. 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. Rule *Fake* models the spy's illegal activity, which includes forging any timestamps. Rule *BK1* lets any agent begin a protocol session at any time. Rule *BK2* models the server's operation, which is subordinate to some other agent having sent the first message of the protocol. Since the first message is a cleartext, the spy may easily fake it many times and overload the server. The session key has not been used before and is accompanied with a timestamp drawn from the current time. Rule *BK3* states that an agent who initiated a protocol session may proceed with the third step of the protocol if she has been sent a message with a non-expired session-key timestamp. By rule *BK4*, an agent completes the protocol if he has been sent a specific intelligible message containing two non-expired timestamps. Note that the model does not need to increment the timestamp in the last protocol step as the message is already structurally different from all others. Finally, rule *Oops* allows accidental leaks of session keys at any time.

Note that rules *BK1* to *BK4* model the agents' behaviour that is legal according to BAN Kerberos. For example, it is visible that only the correct timestamps are inserted.

5.4 Verifying BAN Kerberos

The main guarantees that we have proved about BAN Kerberos are presented in this section, where evs is always a generic trace of the set $\mathsf{bankerberos}$. The authentication goals (§5.4.6) will be subsequently strengthened thanks to an extension to the approach (§7.3).

```
Base
[] ∈ bankerberos

Fake
[| evs ∈ bankerberos; X ∈ synth (analz (spies evs)) |]
⟹ Says Spy B X # evs ∈ bankerberos

BK1
[| evs ∈ bankerberos |]
⟹ Says A Server {|Agent A, Agent B|} # evs ∈ bankerberos

BK2
[| evs ∈ bankerberos; Key K ∉ used evs;
   Says A' Server {|Agent A, Agent B|} ∈ set evs |]
⟹ Says Server A (Crypt (shrK A) {|Number (ct evs), Agent B, Key K,
              Crypt (shrK B) {|Number (ct evs), Agent A, Key K|}|})
      # evs ∈ bankerberos

BK3
[| evs ∈ bankerberos;
   Says A Server {|Agent A, Agent B|} ∈ set evs;
   Says S A (Crypt (shrK A) {|Number Ts, Agent B, Key K, Ticket|})
     ∈ set evs;
   ¬ expiredS Ts evs |]
⟹ Says A B {|Ticket, Crypt K {|Agent A, Number (ct evs)|}|}
      # evs ∈ bankerberos

BK4
[| evs ∈ bankerberos;
   Says A' B {|Crypt (shrK B) {|Number Ts, Agent B, Key K|},
             Crypt K {|Agent A, Number Ta|}|} ∈ set evs;
   ¬ expiredS Ts evs; ¬ expiredA Ta evs |]
⟹ Says B A (Crypt K (Number Ta))
      # evs ∈ bankerberos

Oops
[| evs ∈ bankerberos;
   Says Server A (Crypt (shrK A) {|Number Ts, Agent B, Key K,
                                 Ticket|}) ∈ set evs |]
⟹ Notes Spy {|Number Ts, Key K|} # evs ∈ bankerberos
```

Figure 5.2: Formalising BAN Kerberos inductively

5.4.1 Reliability of the BAN Kerberos Model

The only relevant reliability theorem states that the model server is reliable (theorem 5.1). If the cipher is addressed to A, then the server encrypts it with A's shared key, and inserts a session key and the ticket meant for A's peer, B. Since cryptographic keys can be either long-term keys or session keys, Kab is merely shown not to be a long-term one, while it is never used before the server issues it, so it is fresh. The timestamp is chosen as the current time of the subtrace where the key is being issued.

Theorem 5.1. *If evs contains*

$$ev = \mathsf{Says\ Server}\ A\,(\mathsf{Crypt}\,K\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,B, \mathsf{Key}\,Kab,\ Ticket|\!\})$$

then

$$\begin{aligned}&K = \mathsf{shrK}\,A \quad \textit{and} \quad Kab \notin \mathsf{range\,shrK} \quad \textit{and}\\ &Ticket = (\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,A, \mathsf{Key}\,Kab|\!\}) \quad \textit{and}\\ &\mathsf{Key}\,Kab \notin \mathsf{used}(\mathsf{before}\,ev\,\mathsf{on}\,evs) \quad \textit{and}\\ &Tk = \mathsf{ct}(\mathsf{before}\,ev\,\mathsf{on}\,evs)\end{aligned}$$

Proving the last conjunct of the assertion requires extending the existing strategy (§4.1) by two lemmas for the symbolic evaluation of the length of a subtrace.

5.4.2 Regularity

The protocol employs a single kind of long-term key and never sends it in the traffic, so a basic regularity lemma is provable. An agent's shared key appears in the traffic if and only if the agent is compromised (lemma 5.2).

Lemma 5.2. $\mathsf{Key}(\mathsf{shrK}\,A) \in \mathsf{analz}(\mathsf{spies}\ evs) \iff A \in \mathsf{bad}.$

All subsequent guarantees about certificates sealed by a shared key will appeal to this lemma to guarantee their integrity.

5.4.3 Authenticity

The second message and the ticket contained inside it represent the crucial certificates of the protocol because they are meant to deliver the session key to A and B respectively. They are encrypted under shared keys. Applying the regularity lemma assures that the spy cannot handle the keys that encrypt the certificates and so cannot spoof them.

Let us consider the certificate for A and assume the agent not to be compromised in order to apply the regularity lemma. When A receives the certificate and inspects it, she realises that it is the four-component certificate delivering a session key. The theorem proves that the certificate was created by the only entity that is legally entitled to issue session keys, the server, so it is authentic (theorem 5.3).

Theorem 5.3. *If A is not compromised and evs is such that*

$$\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,B, \mathsf{Key}\,Kab,\ Ticket|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs)$$

then evs contains

$$\mathsf{Says\ Server}\ A\,(\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,B, \mathsf{Key}\,Kab,\ Ticket|\!\})$$

Since A is not compromised, the certificate is, via the regularity lemma, integral. Therefore, A learns that the session key was issued at time Tk. Checking this timestamp against the current time prevents her from accepting an old key as fresh. A similar guarantee holds about the certificate $\{\!|Na, B, Kab, X|\!\}$ of the shared-key Needham-Schroeder protocol [84]. Since the nonce Na is previously issued by A and then received along with the session key inside the integral certificate, A infers that the session key is more recent than her nonce.

An analogous guarantee can be established about the certificate for B (theorem 5.4).

Theorem 5.4. *If B is not compromised and evs is such that*

$$\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,A, \mathsf{Key}\,Kab|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs)$$

then evs contains

$$\begin{aligned}\mathsf{Says\ Server}\ A\,(\mathsf{Crypt}(\mathsf{shrK}A)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,B, \mathsf{Key}\,Kab,\\ \mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,A, \mathsf{Key}\,Kab|\!\}|\!\})\end{aligned}$$

Agent B is guaranteed that the session key was issued at time Tk and so can verify its freshness. The corresponding guarantee on the shared-key Needham-Schroeder protocol [18, §5.1] relies on the certificate $\{\!|K, A|\!\}_{Kb}$. This time B cannot decide whether he is accepting an old session key as fresh, which raises the known chance of a replay attack.

5.4.4 Unicity

The server issues fresh session keys, so the same key cannot be issued twice. Using Paulson's strategy [88, §4.4], we can enforce that, if a session key appears within two message contexts, then the contexts must be the same (theorem 5.5).

Theorem 5.5. *If evs contains*

$$\mathsf{Says\ Server}\ A\,(\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,B, \mathsf{Key}\,Kab,\ Ticket|\!\})\ \textit{and}$$
$$\mathsf{Says\ Server}\ A'\,(\mathsf{Crypt}(\mathsf{shrK}\,A')\{\!|\mathsf{Number}\ Tk', \mathsf{Agent}\,B', \mathsf{Key}\,Kab,\ Ticket'|\!\})$$

then

$$A = A' \quad \textit{and} \quad Tk = Tk' \quad \textit{and} \quad B = B' \quad \textit{and} \quad Ticket = Ticket'$$

This result is often used when proving theorems that assume the event that issues the session key. The *Oops* rule of the protocol model introduces another event of the same form, but they can be derived to be identical in case they contain the same session key.

A similar result can be enforced if two tickets containing the same, confidential, session key appear in the traffic.

While theorem 5.5 allows the server to send two identical messages, we can prove that this is impossible (theorem 5.6). This is a stronger guarantee because it states that the server never issues the same session key more than once for the same peers using the same timestamp and the same ticket.

Theorem 5.6. *If evs contains*

$$\mathsf{Says\ Server}\ A\,(\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,B, \mathsf{Key}\,Kab,\ Ticket|\!\})$$

then

$$\mathsf{Unique}\ (\mathsf{Says\ Server}\ A\,(\mathsf{Crypt}(\mathsf{shrK}\,A)$$
$$\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,B, \mathsf{Key}\,Kab,\ Ticket|\!\}))\ \mathsf{on}\ \ evs$$

5.4.5 Confidentiality

The proof of the session key compromise theorem (§4.5), which provides a crucial rewriting rule for the analz operator, is conventional, since BAN Kerberos does not use session keys to encrypt other keys.

If the peers' shared keys are not compromised, then no protocol step reveals to the spy the session key that the server issues. Further, if the key is not leaked by accident, then it can be proved to be confidential (theorem 5.7).

Theorem 5.7. *If A and B are not compromised, and evs contains*

$$\mathsf{Says\ Server}\ A\,(\mathsf{Crypt}(\mathsf{shrK}\ A)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\ B, \mathsf{Key}\ Kab,\ Ticket|\!\})$$

but does not contain $\mathsf{Notes\,Spy}\ \{\!|\mathsf{Number}\,T, \mathsf{Key}\,Kab|\!\}$ *for any T, then*

$$\mathsf{Key}\,Kab \notin \mathsf{analz}(\mathsf{spies}\ evs)$$

The application of the authenticity theorem 5.3 makes this result useful to A within the minimal trust still imposed by the assumptions on B and on the oops event (theorem 5.8).

Theorem 5.8. *If A and B are not compromised, and evs is such that*

$$\mathsf{Crypt}(\mathsf{shrK}\ A)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\ B, \mathsf{Key}\ Kab,\ Ticket|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs)$$

but does not contain $\mathsf{Notes\,Spy}\ \{\!|\mathsf{Number}\,T, \mathsf{Key}\,Kab|\!\}$ *for any T, then*

$$\mathsf{Key}\,Kab \notin \mathsf{analz}(\mathsf{spies}\ evs)$$

Likewise, session key confidentiality can be proved from B's viewpoint (theorem 5.9) by application of the authenticity theorem 5.4 to the confidentiality theorem 5.7.

Theorem 5.9. *If A and B are not compromised, and evs is such that*

$$\mathsf{Crypt}(\mathsf{shrK}\ B)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\ A, \mathsf{Key}\ Kab|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs)$$

but does not contain $\mathsf{Notes\,Spy}\ \{\!|\mathsf{Number}\,T, \mathsf{Key}\,Kab|\!\}$ *for any T, then*

$$\mathsf{Key}\,Kab \notin \mathsf{analz}(\mathsf{spies}\ evs)$$

5.4.6 Authentication

One of the aims of BAN Kerberos is to enforce mutual agent authentication. The authenticator of the third message should authenticate A to B, and the fourth message as a whole should authenticate B to A. Our proofs suggest that the goal is met within the minimal trust. This section shows that the protocol establishes mutual weak agreement, while a few extensions to the approach will show that it also establishes mutual non-injective agreement on the session key and on an important timestamp (see §7.3).

Tracing back the originator of the authenticator requires the session key that seals it to be confidential. Thus, the assumptions of the confidentiality theorem 5.9 must be allowed, since we are reasoning from B's viewpoint. In these circumstances, the authenticator can be proved to have originated with A during the third step of the protocol (theorem 5.10).

Theorem 5.10. *If B is not compromised, and evs is such that*

$$\mathsf{Crypt}(\mathsf{shrK}\ B)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\ A, \mathsf{Key}\ Kab|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs) \quad \textit{and}$$
$$\mathsf{Crypt}\ Kab\{\!|\mathsf{Agent}\ A, \mathsf{Number}\ Ta|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs)$$

and does not contain $\mathsf{Notes}\ \mathsf{Spy}\ \{\!|\mathsf{Number}\ T, \mathsf{Key}\ Kab|\!\}$ *for any T, then it contains*

$$\mathsf{Says}\ A\ B\ \{\!|\mathsf{Crypt}(\mathsf{shrK}\ B)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\ A, \mathsf{Key}\ Kab|\!\},$$
$$\mathsf{Crypt}\ Kab\{\!|\mathsf{Agent}\ A, \mathsf{Number}\ Ta|\!\}|\!\}$$

If B receives the ticket, extracts the session key, and then receives the authenticator that is sealed by it, he infers that A was alive and meant to communicate with him. The theorem does not express A's knowledge of Kab or Ta.

The same strategy proves that a cipher that has the form of the fourth message of the protocol was indeed created during the fourth step (theorem 5.11). The necessary condition that the cipher be sealed by a confidential session key is conveyed via an appeal to the confidentiality theorem 5.8.

Theorem 5.11. *If A is not compromised, and evs is such that*

$$\mathsf{Crypt}(\mathsf{shrK}\ A)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\ B, \mathsf{Key}\ Kab, Ticket|\!\}$$
$$\in \mathsf{parts}(\mathsf{spies}\ evs) \quad \textit{and}$$
$$\mathsf{Crypt}\ Kab(\mathsf{Number}\ Ta) \in \mathsf{parts}(\mathsf{spies}\ evs)$$

and does not contain $\mathsf{Notes}\ \mathsf{Spy}\ \{\!|\mathsf{Number}\ T, \mathsf{Key}\ Kab|\!\}$ *for any T, then it contains*

$$\mathsf{Says}\ B\ A\ (\mathsf{Crypt}\ Kab(\mathsf{Number}\ Ta))$$

When A gets hold of the two suitable certificates, she infers that B was alive and meant to communicate with her. The theorem does not express B's knowledge of Kab or Ta.

5.4.7 Key Distribution

We want to establish whether, at the end of a protocol session, the peers have evidence that they share a session key, which is a major goal of the protocol.

In order to decrypt the authenticator $\{\!|A, Ta|\!\}_{Kab}$, B must learn Kab from the ticket $\{\!|Tk, A, Kab|\!\}_{Kb}$. Under the minimal trust, B considers Kab

confidential. Therefore, the authenticator cannot have been faked and, so, must have been sent by A in the third step of the protocol. The authentication theorem 5.10 establishes this formally. Also, since A is the true creator of the authenticator, she must know the session key to seal it. This notion cannot be captured formally in the current approach: in general, when an event $\mathsf{Says}\,A\,B\,X$ occurs, A may be just forwarding X and therefore having no knowledge about its contents.

To overcome this problem, we extend the Inductive Approach in chapter 7 to capture the notion of an agent being the true creator of a message. Modelling message reception will provide an alternative reasoning: when B receives the ticket and the authenticator, A must have previously received an instance of the second message and so must have learnt the session key.

The authentication theorem 5.11 allows the same considerations from A's viewpoint. Although the certificate $\{\!|Ta|\!\}_{Kab}$ is proved to have been sent by B, proving B's knowledge of the session key requires, as mentioned above, further modelling.

5.5 A Temporal Modelling of Accidents

The minimal trust of the confidentiality and authentication theorems is also raised by the necessity of checking that no oops event occurred. This is in general not possible from any friendly agent's viewpoint (see §2.2.4), which raises concern that perhaps allowing the leaking of any session key at any time makes the model too permissive.

We observe that the longer a message component is in the traffic, the higher is the risk that the spy may get hold of it. In particular, the probability that a session key becomes compromised increases over time.

In the light of these considerations, we assume for this protocol that a session key cannot be leaked as long as its lifetime has not expired, namely the key cannot be leaked as long as it is still valid. Restricting the model accordingly requires adding the condition

$$\mathsf{expiredS}\ Tk\ evs$$

to the *Oops* rule. Hence, session-key leaks may only happen to histories that have evolved for longer than $\mathsf{seskLife}$ after the time of issue of the session key. In the new model, all confidentiality and authentication theorems (5.7 to 5.11) rest on the assumption $\neg\mathsf{expiredS}\ Tk\ evs$, instead of "$evs$ does not contain $\mathsf{Notes}\,\mathsf{Spy}\,\{\!|\mathsf{Number}\,T, \mathsf{Key}\,Kab|\!\}$ for any T". Any agent can check the new temporal assumption by verifying that the current time does not differ from the timestamp by more than the allowed lifetime. Therefore, the minimal trust required by the theorem becomes lower.

The temporal modelling of accidents will also be used in the analysis of Kerberos IV (see next chapter).

Chapter 6

Verifying a Deployed Protocol

The Kerberos IV protocol is verified using the extended Inductive Approach and a weakness is discovered in the protocol management of timestamps.

The most commonly deployed variant of the Kerberos protocol is Version IV [77]. Kerberos IV is a password-based system for authentication and authorisation over local area networks. It was developed during the late 1980s with the following aim: once a user authenticates himself to a network machine, the process of obtaining authorisation to access another network service should be completely transparent to him. Hence, the user certainly must not be prompted with a password request during the authorisation phase. He only should have to enter his password once during the authentication phase.

Kerberos IV pursues its aim by delivering certain credentials to the login process of each user during the authentication phase. These credentials are to be used with a suitable trusted server during any subsequent authorisation phase. Receiving, storing and using the credentials are invisible to the user.

We model both the trusted server of the authentication phase and that of the authorisation phase [17]. Each of them issues a session key with a specific lifetime. The session key issued by the first server is used to encrypt the one issued by the second server. This feature greatly complicates the verification of the confidentiality goals [19], but a technique for modelling the association between two message components can be reused from the analysis of the Yahalom protocol [90]. Using the temporal modelling of accidents (§5.5), we also investigate the authenticity, unicity, confidentiality, authentication and key distribution goals. During the verification of our principle of goal availability, we discover a weakness in the management of timestamps, which leads to an attack: the spy may access a network service using a session key that belongs to a user who is no longer present on the network. Our proofs suggest a simple fix, which can be formally verified to be effective.

To our knowledge, this is the first mechanised modelling of the complete protocol. We have previously analysed the protocol by ASMs (§2.1.4) formalising the actions of an unbounded population of agents by means of a detailed algebraic model [21]. Using that formal specification as a starting point rather than the informal one [77] significantly simplifies our work towards the inductive definition of the protocol model.

Mitchell et al. [79] model check a highly simplified version of the protocol, which derives from Kohl et al. [57]. Neither are timestamps included in their model, nor are multiple runs allowed. They find no attacks on a system of size three — consisting of an initiator, the Kerberos servers and a responder — and a "redirection" attack on a system of size four, including two responders, which the full Kerberos IV prevents by explicitness (see next section).

The chapter describes the three phases of Kerberos IV (§6.1), its modelling (§6.2) and its verification (§6.3).

6.1 The Kerberos IV Protocol

Kerberos IV is essentially a key distribution protocol. It relies on the Kerberos System (figure 6.1), which comprises two trusted servers and a database containing all users' passwords sealed using the standard Unix one-way encryption algorithm.

6.1.1 Overview

Once a user types in his identifier and his password, his login process encrypts the password using the Unix algorithm. This process is the agent who initiates the protocol (recall we view agents as processes, §3.1.1). The full protocol consists of three phases, only the first being compulsory. The first phase, AUTHENTICATION, comprises the first two protocol steps and serves to authenticate the initiator to the Kerberos System, specifically to the Kerberos Authentication Server, Kas in short. If the user is a registered one, the two agents share the sealed password as a long-term secret, which constitutes the initiator's shared key: the initiator has computed it, while Kas has looked it up in the database. Using this secret, Kas issues some authorisation

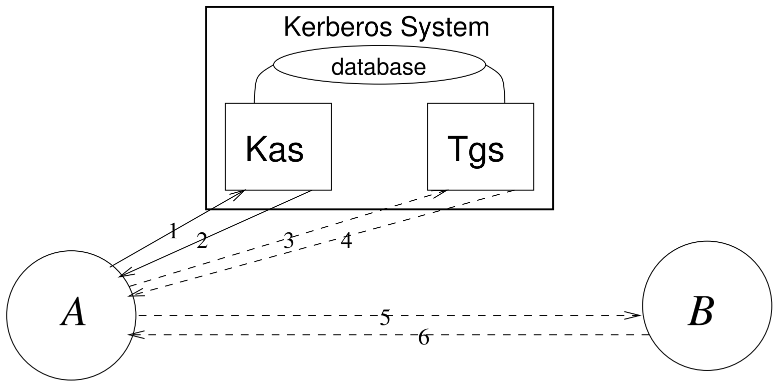

Figure 6.1: The Kerberos IV layout

credentials that the initiator will use in the second phase, AUTHORISATION, which comprises the third and fourth protocol steps. This phase only occurs when the initiator, who is currently running on a workstation, requires a network service. Using the previously obtained authorisation credentials, the initiator contacts the Kerberos System, specifically the Ticket Granting Server, Tgs in brief. The initiator obtains some service credentials to use in the final phase, SERVICE, in order to access the requested service.

Note that, while Kas does not have a long-term key of its own, Tgs, like all other agents, does have one that is shared with Kas.

6.1.2 Details

The complete protocol is presented in figure 6.2.

During the AUTHENTICATION phase, the initiator A queries Kas with her identity, Tgs and a timestamp $T1$; Kas issues a session key and looks up A's shared key in the database. It replies with a message sealed by A's shared key containing the session key, its timestamp Ta, Tgs and a ticket. The session key and the ticket are the credentials to use in the subsequent authorisation phase, so we address them as *authkey* and *authticket* respectively. The authticket is sealed by the Tgs shared key and contains a copy of the authkey, its timestamp and its peers. The lifetime of an authkey is several hours.

If $T1$ is not much older than Ta with respect to a given lifetime, then A is assured that the Kas reply was prompt. If this check is affirmative, A may start the AUTHORISATION phase. She sends Tgs a three-component message including the authticket, an authenticator sealed by the authkey containing

Authentication

$$\begin{array}{llcll}
1. & A & \rightarrow & \mathsf{Kas} & : A, \mathsf{Tgs}, \mathit{T1} \\
2. & \mathsf{Kas} & \rightarrow & A & : \{\!|\mathit{authK}, \mathsf{Tgs}, \mathit{Ta}, \underbrace{\{\!|A, \mathsf{Tgs}, \mathit{authK}, \mathit{Ta}|\!\}_{\mathit{Ktgs}}}_{\mathit{authTicket}}|\!\}_{\mathit{Ka}}
\end{array}$$

Authorisation

$$\begin{array}{llcll}
3. & A & \rightarrow & \mathsf{Tgs} & : \underbrace{\{\!|A, \mathsf{Tgs}, \mathit{authK}, \mathit{Ta}|\!\}_{\mathit{Ktgs}}}_{\mathit{authTicket}}, \underbrace{\{\!|A, \mathit{T2}|\!\}_{\mathit{authK}}}_{\mathit{authenticator}}, B \\
4. & \mathsf{Tgs} & \rightarrow & A & : \{\!|\mathit{servK}, B, \mathit{Ts}, \underbrace{\{\!|A, B, \mathit{servK}, \mathit{Ts}|\!\}_{\mathit{Kb}}}_{\mathit{servTicket}}|\!\}_{\mathit{authK}}
\end{array}$$

Service

$$\begin{array}{llcll}
5. & A & \rightarrow & B & : \underbrace{\{\!|A, B, \mathit{servK}, \mathit{Ts}|\!\}_{\mathit{Kb}}}_{\mathit{servTicket}}, \underbrace{\{\!|A, \mathit{T3}|\!\}_{\mathit{servK}}}_{\mathit{authenticator}} \\
6. & B & \rightarrow & A & : \{\!|\mathit{T3} + 1|\!\}_{\mathit{servK}}
\end{array}$$

Figure 6.2: The Kerberos IV protocol

her identity and a new timestamp *T2*, and *B*'s identity. The lifetime of an authenticator is a few minutes. Upon reception of the message, Tgs decrypts the authticket, extracts the authkey and checks the validity of its timestamp *Ta*, namely that *Ta* is not too old with respect to the lifetime of authkeys. Then, Tgs decrypts the authenticator using the authkey and checks the validity of *T2* with respect to the lifetime of authenticators. Finally, Tgs issues a new session key and looks up *B*'s shared key in the database. It replies with a message sealed by the authkey containing the new session key, its timestamp *Ts*, *B* and a ticket. The session key and the ticket are the credentials to use in the subsequent service phase, so we address them as *servkey* and *servticket* respectively. The servticket is sealed by *B*'s shared key and contains a copy of the servkey, its timestamp and its peers. The lifetime of a servkey is a few minutes.

If *T2* is not much older than *Ts* with respect to a given lifetime, then *A* is assured that the Tgs reply was prompt. If so, *A* may start the service phase. She sends *B* a two-component message including the servticket and an authenticator sealed by the servkey containing her identity and a new timestamp *T3*. Upon reception of the message, *B* decrypts the servticket, extracts the servkey and checks the validity of its timestamp Ts. Then, B decrypts the authenticator using the servkey and checks the validity of $T3$. Finally, B increments $T3$, seals it by the servkey and sends it back to A.

In a simplified version of the protocol [57], the fourth message does not include either of the two occurrences of B's identity. The spy can then mount the mentioned redirection attack [79]: she changes B's identity to some compromised C's (or her own) in the third message; she intercepts the fifth message, decrypts the servticket, extracts the servkey, decrypts the authenticator and extracts the timestamp $T3$; she forges the sixth message. Hence, A believes to have completed a protocol session with B when in fact she has been redirected to someone else and B never participated. The complete Kerberos IV clearly does not suffer this attack.

6.2 Modelling Kerberos IV

Modelling the two trusted servers requires modifying the Isabelle datatype for agents (§3.1.1) as

```
datatype agent = Kas | Tgs | Friend i | Spy
```

The definition of initState must be updated so to allow both trusted servers to know all agents' shared keys. Both Kas and Tgs are assumed not to be compromised.

The protocol reliance on time is identical to that of BAN Kerberos, so we adopt the same discrete formalisation for the current time of a trace (§5.3).

We define three natural numbers, authkLife, servkLife, authLife, formalising respectively the lifetimes of an authkey, of a servkey, and of an authenticator. Three intuitive binary predicates

$$\mathsf{expiredAK},\ \mathsf{expiredSK},\ \mathsf{expiredA} : [\mathsf{nat}, \mathsf{event\ list}]$$

check their respective validity, namely the validity of the timestamps associated with them

$$\begin{array}{rcl} \mathsf{expiredAK}\ Ta\ \ evs & \equiv & (\mathsf{ct}\, evs) - Ta > \mathsf{authkLife} \\ \mathsf{expiredSK}\ Ts\ \ evs & \equiv & (\mathsf{ct}\, evs) - Ts > \mathsf{servkLife} \\ \mathsf{expiredA}\, T\ \ evs & \equiv & (\mathsf{ct}\, evs) - T > \mathsf{authLife} \end{array}$$

When any of these predicates holds for some timestamp and some trace, we say that the timestamp has expired or that the corresponding key, or authenticator, has expired on that trace.

A further lifetime, replyLife, indicates the validity interval of any trusted server's replies. A suitable binary predicate checks that a timestamp is issued within such lifetime after another timestamp

$$\mathsf{valid}\, T\, \mathsf{wrt}\, T' \quad \equiv \quad T \leq T' + \mathsf{replyLife}$$

Agents discard late servers replies, which may be either due to servers' temporary malfunction or to network latency, by operating this check. Those replies arriving within the lifetime are more highly reliable.

The constant kerberos, declared as set of lists of events, represents the formal protocol model and is defined by induction below.

6.2.1 Basics

The inductive definition of kerberos contains two basic rules (figure 6.3). The first sets the base of the induction (*Base*) introducing the empty trace in the protocol model. The second models the spy's illegal activity (*Fake*) allowing the spy to send any fake message, obtained from the active analysis of the traffic, to any agent.

Base
```
[] ∈ kerberos
```

Fake
```
[| evs ∈ kerberos; X ∈ synth (analz (spies evs)) |]
⟹ Says Spy B X # evs ∈ kerberos
```

Figure 6.3: Modelling Kerberos IV: basics

6.2.2 Authentication Phase

The protocol initiator A must go through the authentication phase with Kas (figure 6.4). Any trace, including the empty one, can be extended by the event formalising the first message of the protocol (*K1*). This is faithful to the real world, where any agent, including the spy, may decide to initiate a protocol session. In the model, the Kas reply is subject to the previous occurrence of a suitable request from some agent (*K2*). In the real world, Kas must receive the request before operating, but the approach does not yet include message reception (see §7.2).

```
K1
evs ∈ kerberos
⟹ Says A Kas {|Agent A, Agent Tgs, Number (ct evs)|} # evs ∈ kerberos


K2
[| evs ∈ kerberos; Key authK ∉ used evs;
   Says A' Kas {|Agent A, Agent B, Number T1|} ∈ set evs |]
⟹ Says Kas A (Crypt (shrK A) {|Key authK, Agent Tgs, Number (ct evs),
                              Crypt (shrK Tgs) {|Agent A, Agent Tgs,
                                        Key authK, Number (ct evs)|}|})
      # evs ∈ kerberos
```

Figure 6.4: Modelling Kerberos IV: authentication phase

6.2.3 Authorisation Phase

The initiator A may require authorisation to a network service (figure 6.5). Before contacting Tgs, A checks that some agent has issued a message con-

```
K3
[| evs ∈ kerberos;
   Says A Kas {|Agent A, Agent Tgs, Number T1|} ∈ set evs;
   Says Kas' A (Crypt (shrK A) {|Key authK, Agent Tgs, Number Ta,
                              authTicket|}) ∈ set evs;
   ¬ expiredAK Ta evs; valid Ta wrt T1 |]
⟹ Says A Tgs {|authTicket, Crypt authK {|Agent A, Number (ct evs)|},
               Agent B|}
      # evs ∈ kerberos


K4
[| evs ∈ kerberos; Key servK ∉ used evs; B ≠ Tgs;
   Says A' Tgs {|Crypt (shrK Tgs) {|Agent A, Agent Tgs,
                                Key authK, Number Ta|},
               Crypt authK {|Agent A, Number T2|}, Agent B|}
     ∈ set evs;
   ¬ expiredAK Ta evs; ¬ expiredA T2 evs |]
⟹ Says Tgs A (Crypt authK {|Key servK, Agent B, Number (ct evs),
                           Crypt (shrK B) {|Agent A, Agent B,
                                    Key servK, Number (ct evs)|}|})
      # evs ∈ kerberos
```

Figure 6.5: Modelling Kerberos IV: authorisation phase

taining the authorisation credentials for her. While the message must have a specific form, its sender can be anyone, even the spy. The reception of such message is, as above, implicit (*K3*). Note that A checks that the authkey has not expired and that it was issued within the validity interval from her request. Once a trace records a query of the expected form, Tgs may issue its reply provided that neither the authkey nor the authenticator has expired (*K4*).

6.2.4 Service Phase

This phase can be modelled (figure 6.6) by the same layout used for the authorisation one. The initiator A can contact the requested service B only if the trace records the issue of a servkey within the validity interval from A's request. Also, the servkey must not have expired (*K5*). Before acknowledging A's request, B checks that neither the servkey nor the authenticator has expired (*K6*). We do not need to model the incremented timestamp in the last step because this does not contribute to the goals of the protocol (see §6.3.6, §7.4). The message is still different from all others that are sent in the protocol.

K5
```
[| evs ∈ kerberos;
   Says A Tgs {|authTicket, Crypt authK {|Agent A, Number T2|}, Agent B|}
     ∈ set evs;
   Says Tgs' A (Crypt authK {|Key servK, Agent B, Number Ts, servTicket|})
     ∈ set evs;
   ¬ expiredSK Ts evs; valid Ts wrt T2 |]
⟹ Says A B {|servTicket, Crypt servK {|Agent A, Number (ct evs5)|}|}
      # evs ∈ kerberos
```

K6
```
[| evs ∈ kerberos;
   Says A' B {|Crypt (shrK B) {|Agent A, Agent B, Key servK, Number Ts|},
               Crypt servK {|Agent A, Number T3|}|} ∈ set evs;
   ¬ expiredSK Ts evs; ¬ expiredA T3 evs |]
⟹ Says B A (Crypt servK (Number T3)) # evs ∈ kerberos
```

Figure 6.6: Modelling Kerberos IV: service phase

6.2.5 Accidents

The authkeys are operationally different from servkeys, although they are all session keys. Chiefly, the two types of keys are associated with timestamps that are in turn associated with different lifetimes. Also, while authkeys are issued by Kas and are meant to be used with Tgs, servkeys are issued by Tgs and are meant to be used with the network services. Furthermore, when an authkey expires, the corresponding user is logged out from the workstation and all his processes are killed. On the contrary, when a servkey expires, the intended network service will not accept it, so the initiator must undertake a new authorisation phase.

We allow for accidental leaks of session keys (figure 6.7) adopting the temporal modelling of accidents seen above (§5.5). An authkey can be noted by the spy together with its timestamp and its peers provided that it has expired (*OopsA*), and so can a servkey (*OopsS*). It is interesting to investigate whether the spy can exploit expired session keys to get hold of non-expired ones. This would be a major success for her, since the model allows the reuse of a session key within its lifetime, like the real world does.

OopsA
```
[| evsOa ∈ kerberos;
   Says Kas A (Crypt (shrK A) {|Key authK, Agent Tgs, Number Ta,
                              authTicket|}) ∈ set evsOa;
   expiredAK Ta evsOa |]
⟹ Notes Spy {|Agent A, Agent Tgs, Number Ta, Key authK|}
      # evsOa ∈ kerberos
```

OopsS
```
[| evsOs ∈ kerberos;
   Says Tgs A (Crypt authK {|Key servK, Agent B, Number Ts, servTicket|})
     ∈ set evsOs;
   expiredSK Ts evsOs |]
 ⟹ Notes Spy {|Agent A, Agent B, Number Ts, Key servK|}
       # evsOs ∈ kerberos
```

Figure 6.7: Modelling Kerberos IV: accidents

6.3 Verifying Kerberos IV

Kerberos IV makes use of session keys in a rather peculiar way: Tgs employs an authkey to encrypt a servkey in the fourth message. This feature greatly complicates the verification of the confidentiality goals. Our proofs show that synchronising the issue of the two types of keys is crucial. If no synchronisation is implemented, then the protocol responder (the network service) gets a weak confidentiality guarantee because the spy may exploit some servkeys within their validity interval.

While introducing suitable abbreviations to distinguish the two types of session keys or express the association between authkeys and servkeys, this section presents all goals verified of Kerberos IV.

In the following, *evs* is a generic trace of the set kerberos.

6.3.1 Reliability of the Kerberos IV Model

To address the authkeys formally, we define the function

$$\mathsf{AuthKeys} : \mathsf{event\ list} \longrightarrow \mathsf{key\ set}$$

so that AuthKeys *evs* yields all session keys that Kas issues on the trace *evs*, ignoring any eventual repetitions

$$\begin{aligned}
&\mathsf{AuthKeys}\ evs \ \triangleq \\
&\quad \{authK \mid \exists\ A\ Ts \mid \\
&\qquad \mathsf{Says\ Kas}\ A\,(\mathsf{Crypt(shrK}\,A)\{\!|\mathsf{Key}\ authK, \mathsf{Agent\,Tgs}, \mathsf{Number}\ Ts, \\
&\qquad\qquad\qquad \mathsf{Crypt(shrK\,Tgs)}\{\!|\mathsf{Agent}\ A, \mathsf{Agent\,Tgs}, \\
&\qquad\qquad\qquad\qquad \mathsf{Key}\ authK, \mathsf{Number}\ Ts|\!\}|\!\}) \\
&\qquad \in \mathsf{set}\ evs\}
\end{aligned}$$

Several lemmas are needed for the symbolic evaluation of this function. For example, one states formally that no authkeys appear on an empty trace (lemma 6.1), and another signifies that Kas does introduce an authkey (lemma 6.2).

Lemma 6.1. $\mathsf{AuthKeys}\,[\,] = \{\}$.

Lemma 6.2.

$$\begin{aligned}
&\mathsf{AuthKeys(Says\ Kas}\ A\,(\mathsf{Crypt(shrK}\,A)\{\!|\mathsf{Key}\ authK, \mathsf{Agent\,Tgs}, \mathsf{Number}\ Ta, \\
&\qquad\qquad authTicket|\!\})\#\ evs) \\
&\quad = \{authK\} \cup (\mathsf{AuthKeys}\ evs)
\end{aligned}$$

The servkeys on a trace *evs* are formalised as those keys that do not belong either to the range of shrK (so they are session keys) or to AuthKeys *evs* (so they are not authkeys).

Our model Kas can be proved to be reliable (theorem 6.3). When addressing a message to an agent A, Kas seals it by A's shared key and includes in it a session key that is an authkey and a well-formed authticket. Using the function before (defined in §4.1), we establish that the authkey is fresh when it is issued and its timestamp is chosen as the current time of the moment.

Theorem 6.3. *If evs contains*

$$ev = \mathsf{Says\ Kas}\ A\,(\mathsf{Crypt}\,K\{\!|\mathsf{Key}\ authK, \mathsf{Agent\,Tgs}, \mathsf{Number}\ Ta, authTicket|\!\})$$

then

$$\begin{aligned}
&K = \mathsf{shrK}\,A \quad \textit{and}\\
&authK \notin \mathsf{range\,shrK} \quad \textit{and} \quad authK \in \mathsf{AuthKeys}\ evs \quad \textit{and}\\
&authTicket = (\mathsf{Crypt}(\mathsf{shrK\,Tgs})\{\!|\mathsf{Agent}\,A, \mathsf{Agent\,Tgs},\\
&\qquad\qquad\qquad\qquad \mathsf{Key}\ authK, \mathsf{Number}\ Ta|\!\}) \quad \textit{and}\\
&\mathsf{Key}\ authK \notin \mathsf{used}(\mathsf{before}\ ev\ \mathsf{on}\ evs) \quad \textit{and}\\
&Ta = \mathsf{ct}(\mathsf{before}\ ev\ \mathsf{on}\ evs)
\end{aligned}$$

An analogous guarantee enforces the reliability of the model Tgs (theorem 6.4). An authkey is used to seal the message, which includes a fresh servkey and the servticket that quotes it.

Theorem 6.4. *If evs contains*

$$\begin{aligned}
&ev =\\
&\mathsf{Says\ Tgs}\ A\,(\mathsf{Crypt}\ authK\{\!|\mathsf{Key}\ servK, \mathsf{Agent}\,B, \mathsf{Number}\ Ts, servTicket|\!\})
\end{aligned}$$

then

$$\begin{aligned}
&authK \notin \mathsf{range\,shrK} \quad \textit{and} \quad authK \in \mathsf{AuthKeys}\ evs \quad \textit{and}\\
&servK \notin \mathsf{range\,shrK} \quad \textit{and} \quad servK \notin \mathsf{AuthKeys}\ evs \quad \textit{and}\\
&servTicket = (\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Agent}\,A, \mathsf{Agent}\,B,\\
&\qquad\qquad\qquad\qquad \mathsf{Key}\ servK, \mathsf{Number}\ Ts|\!\}) \quad \textit{and}\\
&\mathsf{Key}\ servK \notin \mathsf{used}(\mathsf{before}\ ev\ \mathsf{on}\ evs) \quad \textit{and}\\
&Ts = \mathsf{ct}(\mathsf{before}\ ev\ \mathsf{on}\ evs)
\end{aligned}$$

As expected, to obtain these results, the general strategy for proving the reliability theorems (§4.1) must be upgraded with frequent appeals to lemmas 6.1 and 6.2.

6.3.2 Regularity

Kerberos IV never sends shared keys on the network, so a shared key is available to the spy if and only if its owner is compromised. The corresponding regularity lemma has the usual formulation (lemma 6.5).

Lemma 6.5. $\mathsf{Key}(\mathsf{shrK}\, A) \in \mathsf{analz}(\mathsf{spies}\ evs) \quad \Longleftrightarrow \quad A \in \mathsf{bad}.$

6.3.3 Authenticity

The protocol intends to deliver an authkey to A and Tgs and a servkey to A and B. Determining the originator of a certificate that contains a session key will confirm the authenticity of both the certificate and the key. As usual, the analysis will be performed from each agent's viewpoint, while all proofs are carried out according to the strategies described above (§4.3).

The instance of the second message that is sealed by A's shared key carries the authkey meant for A. An appeal to the regularity lemma guarantees that the certificate is tamper-proof, so induction proves it to have originated with Kas (theorem 6.6). Recall that the theorem becomes useful to A only upon reception of the certificate.

Theorem 6.6. *If A is not compromised and evs is such that*

$$\begin{aligned}&\mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|\mathsf{Key}\ authK, \mathsf{Agent\ Tgs}, \mathsf{Number}\ Ta, authTicket|\!\}\\ &\quad \in \mathsf{parts}(\mathsf{spies}\ evs)\end{aligned}$$

then evs contains

$$\begin{aligned}&\mathsf{Says\ Kas}\ A\, (\mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|\mathsf{Key}\ authK, \mathsf{Agent\ Tgs},\\ &\qquad\qquad\qquad\qquad \mathsf{Number}\ Ta, authTicket|\!\})\end{aligned}$$

The same guarantee can be enforced on the certificate that delivers the authkey to Tgs, the authticket (theorem 6.7). Recall that Tgs is not compromised.

Theorem 6.7. *If evs is such that*

$$\begin{aligned}&\mathsf{Crypt}(\mathsf{shrK\ Tgs})\{\!|\mathsf{Agent}\ A, \mathsf{Agent\ Tgs}, \mathsf{Key}\ authK, \mathsf{Number}\ Ta|\!\}\\ &\quad \in \mathsf{parts}(\mathsf{spies}\ evs)\end{aligned}$$

then evs contains

$$\begin{aligned}&\mathsf{Says\ Kas}\ A\ (\mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|\mathsf{Key}\ authK, \mathsf{Agent\ Tgs}, \mathsf{Number}\ Ta,\\ &\qquad \mathsf{Crypt}(\mathsf{shrK\ Tgs})\{\!|\mathsf{Agent}\ A, \mathsf{Agent\ Tgs}, \mathsf{Key}\ authK, \mathsf{Number}\ Ta|\!\}|\!\})\end{aligned}$$

We now investigate the authenticity of the servkey. The fourth message, which delivers this to A, is sealed by the authkey. Clearly, this must be A's authkey in order for the message to be intelligible to A. Since the authkey is not a shared key, the regularity lemma cannot help, so the authkey must be explicitly assumed to be confidential. The second and the fourth messages have the same structure. Theorem 6.6 pinpoints the second message by explicitly assuming that it quotes Tgs. The fourth message quotes a different agent from Tgs, so this assumption must be made to investigate its authenticity (theorem 6.8). This version of the theorem is not useful to A because of the confidentiality assumption on the authkey, which can be relaxed by the confidentiality argument (see §6.3.5).

Theorem 6.8. *If B is not* Tgs *and evs is such that*

$$\begin{array}{l}\mathsf{Crypt}\ \mathit{authK}\,\{\!|\mathsf{Key}\ \mathit{servK}, \mathsf{Agent}\ B, \mathsf{Number}\ \mathit{Ts}, \mathit{servTicket}|\!\}\\ \quad\in \mathsf{parts}(\mathsf{spies}\ \mathit{evs}) \quad \textit{and}\\ \mathsf{Key}\ \mathit{authK} \notin \mathsf{analz}(\mathsf{spies}\ \mathit{evs})\end{array}$$

then there exists A such that evs contains

$$\mathsf{Says}\ \mathsf{Tgs}\ A\,(\mathsf{Crypt}\ \mathit{authK}\,\{\!|\mathsf{Key}\ \mathit{servK}, \mathsf{Agent}\ B, \mathsf{Number}\ \mathit{Ts}, \mathit{servTicket}|\!\})$$

Although the servticket has the same structure as the authticket, the former is sealed by the shared key belonging to an agent different from Tgs. The regularity lemma must be applied to investigate the authenticity of the servticket. We prove (theorem 6.9) that Tgs indeed sent the servticket encrypted under a session key that originated with Kas.

Theorem 6.9. *If B is not compromised and is not* Tgs *and evs is such that*

$$\begin{array}{l}\mathsf{Crypt}(\mathsf{shrK}\ B)\{\!|\mathsf{Agent}\ A, \mathsf{Agent}\ B, \mathsf{Key}\ \mathit{servK}, \mathsf{Number}\ \mathit{Ts}|\!\}\\ \quad\in \mathsf{parts}(\mathsf{spies}\ \mathit{evs})\end{array}$$

then there exist authK and Ta such that evs contains

$$\begin{array}{l}\mathsf{Says}\ \mathsf{Tgs}\ A\ \ (\mathsf{Crypt}\ \mathit{authK}\,\{\!|\mathsf{Key}\ \mathit{servK}, \mathsf{Agent}\ B, \mathsf{Number}\ \mathit{Ts},\\ \qquad\qquad \mathsf{Crypt}(\mathsf{shrK}\ B)\{\!|\mathsf{Agent}\ A, \mathsf{Agent}\ B, \mathsf{Key}\ \mathit{servK}, \mathsf{Number}\ \mathit{Ts}|\!\}|\!\})\ \textit{and}\\ \mathsf{Says}\ \mathsf{Kas}\ A\ \ (\mathsf{Crypt}(\mathsf{shrK}\ A)\{\!|\mathsf{Key}\ \mathit{authK}, \mathsf{Agent}\ \mathsf{Tgs}, \mathsf{Number}\ \mathit{Ta},\\ \qquad\qquad \mathsf{Crypt}(\mathsf{shrK}\ \mathsf{Tgs})\{\!|\mathsf{Agent}\ A, \mathsf{Agent}\ \mathsf{Tgs}, \mathsf{Key}\ \mathit{authK}, \mathsf{Number}\ \mathit{Ta}|\!\}|\!\})\end{array}$$

6.3.4 Unicity

Since Kas issues fresh authkeys, the classical unicity theorem can be used to state that an authkey cannot appear into two different messages. Another theorem rests on the unique predicate (§4.4) to state that Kas only sends each authkey one. The same guarantees hold of the fresh servkeys issued by Tgs.

Both the authticket and the servticket have the same structure and contain a fresh session key, so we can establish a useful unicity result that applies to either ticket under the assumption that the session key is confidential (theorem 6.10). Relaxing this assumption by the confidentiality argument (see §6.3.5) makes the theorem useful to either Tgs or B.

Theorem 6.10. *If evs is such that* $K \notin \mathsf{analz}(\mathsf{spies}\ evs)$ *and*

$$\mathsf{Crypt}(\mathsf{shrK}\,P)\{\!|\mathsf{Agent}\,Q, \mathsf{Agent}\,P, \mathsf{Key}\,K, T|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs) \quad \textit{and}$$
$$\mathsf{Crypt}(\mathsf{shrK}\,P')\{\!|\mathsf{Agent}\,Q', \mathsf{Agent}\,P', \mathsf{Key}\,K, T'|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs)$$

then

$$P = P' \quad \textit{and} \quad Q = Q' \quad \textit{and} \quad T = T'$$

6.3.5 Confidentiality

Each authkey can be used to encrypt several servkeys because an agent, once authenticated, can request more than once service. Therefore, we expect that the compromise of an authkey may cascade on to several servkeys. Conversely, servkeys are never used to encrypt any keys, so the compromise of a servkey should not affect other keys.

These observations can be proved formally, providing three significant session key compromise theorems. They also become fundamental rewrite rules for the analz operator when proving the session key secrecy theorems, the actual confidentiality goals of the protocol.

Session Key Compromise

The proof scripts for the theorems discussed in this section may be found in Appendix A. The association of authkeys with servkeys in Kerberos IV resembles the association of session keys with nonces in the Yahalom protocol [90]. These relations can be formalised in a similar fashion. We define the predicate

$$\mathsf{AKcryptSK} : [\,\mathsf{key}, \mathsf{key}, \mathsf{event\ list}]$$

which holds on a trace whenever it contains an event that associates an authkey with a servkey

$$\begin{aligned}
&\mathsf{AKcryptSK}\ authK\ servK\ evs \ \equiv\\
&\quad \exists\ A\ B\ Ts\ |\\
&\qquad \mathsf{Says\ Tgs}\ A\ (\mathsf{Crypt}\ authK \{\!|\mathsf{Key}\ servK, \mathsf{Agent}\ B, \mathsf{Number}\ Ts,\\
&\qquad\qquad \mathsf{Crypt}(\mathsf{shrK}\ B)\{\!|\mathsf{Agent}\ A, \mathsf{Agent}\ B,\\
&\qquad\qquad\qquad \mathsf{Key}\ servK, \mathsf{Number}\ Ts|\!\}|\!\})\\
&\qquad\quad \in \mathsf{set}\ evs
\end{aligned}$$

Several lemmas must be proved about the predicate. Chiefly, no keys encrypt an authkey or a shared key (lemma 6.11); only a single authkey encrypts a servkey (lemma 6.12); a servkey does not encrypt any keys (lemma 6.13).

Lemma 6.11. *If evs is such that* $K \in \mathsf{AuthKeys}\ evs$*, or* $K \in \mathsf{range\ shrK}$*, then* $\neg\mathsf{AKcryptSK}\ K'\ K\ evs$*.*

Lemma 6.12. *If evs is such that* $\mathsf{AKcryptSK}\ authK\ servK\ evs$*, and* K *is different from* $authK$*, then* $\neg\mathsf{AKcryptSK}\ K\ servK\ evs$*.*

Lemma 6.13. *If evs is such that* $servK \notin \mathsf{AuthKeys}\ evs$*, and* $servK \notin \mathsf{range\ shrK}$*, then* $\neg\mathsf{AKcryptSK}\ servK\ K\ evs$*.*

If a session key K is not associated with a key K' by the predicate $\mathsf{AKcryptSK}$, then the key K is never used to encrypt K', so the compromise of K should not increase the spy's chances to discover K'. Expressing this formally provides an essential lemma (lemma 6.14). The proof is rather long and complicated: simplification takes three quarters of the total computational time, and several case analyses are required afterwards. The assumptions of the theorem let, for example, K be an authkey and K' be a servkey, provided that K has not been used to encrypt K'.

Lemma 6.14. *If* K *is such that* $K \notin \mathsf{range\ shrK}$ *and evs is such that*

$$\neg\mathsf{AKcryptSK}\ K\ K'\ evs$$

then

$$\mathsf{Key}\ K' \in \mathsf{analz}(\{\mathsf{Key}\ K\} \cup (\mathsf{spies}\ evs)) \iff K' = K \lor \mathsf{Key}\ K' \in \mathsf{analz}(\mathsf{spies}\ evs)$$

The first session key compromise theorem concerns authkeys and shared keys: they can be proved unaffected by the accidental loss of any session key (theorem 6.15). The proof follows from applying lemma 6.11 to lemma 6.14. Recall that authkeys are particularly valuable secrets because of their long lifetime.

Theorem 6.15. *If evs is such that*

$$K \in \mathsf{AuthKeys}\ evs \quad or \quad K \in \mathsf{range\,shrK}$$

and $K' \notin \mathsf{range\,shrK}$*, then*

$$\mathsf{Key}\,K \in \mathsf{analz}(\{\mathsf{Key}\,K'\} \cup (\mathsf{spies}\ evs)) \\ \iff K = K' \vee \mathsf{Key}\,K \in \mathsf{analz}(\mathsf{spies}\ evs)$$

Another theorem concerns the servkeys: given the authkey that is associated with a servkey, no other authkey would help the spy to discover the servkey (theorem 6.16). The proof follows from applying lemma 6.12 to lemma 6.14.

Theorem 6.16. *If evs is such that*

$$\mathsf{AKcryptSK}\ authK\ servK\ evs$$

and K is different from $authK$, then

$$\mathsf{Key}\,servK \in \mathsf{analz}(\{\mathsf{Key}\,K\} \cup (\mathsf{spies}\ evs)) \\ \iff servK = K \vee \mathsf{Key}\,servK \in \mathsf{analz}(\mathsf{spies}\ evs)$$

No cryptographic key is affected by the loss of a servkey (theorem 6.17). Note that no assumptions bind K. The proof follows from the application of lemma 6.13 to lemma 6.14.

Theorem 6.17. *If evs is such that*

$$servK \notin \mathsf{AuthKeys}\ evs \quad and \quad servK \notin \mathsf{range\,shrK}$$

then

$$\mathsf{Key}\,K \in \mathsf{analz}(\{\mathsf{Key}\,servK\} \cup (\mathsf{spies}\ evs)) \\ \iff K = servK \vee \mathsf{Key}\,K \in \mathsf{analz}(\mathsf{spies}\ evs)$$

Session Key Secrecy

Session key secrecy theorems express the confidentiality of the session keys. Following the general strategy, we verify the property from the viewpoints of the servers and then refine the findings via the authenticity theorems.

If an authkey has not expired, then it cannot be lost by accident via the *OopsA* rule. Adding that its peer is not compromised assures that the key travels safely inside the second message of the protocol. Under these assumptions, the key confidentiality can be enforced from the viewpoint of Kas (theorem 6.18) by frequent appeals to theorems 6.15 and 6.16.

Theorem 6.18. *If A is not compromised and evs contains*

$$\mathsf{Says}\ \mathsf{Kas}\ A\,(\mathsf{Crypt}(\mathsf{shrK}\ A)\{\!|\mathsf{Key}\ authK, \mathsf{Agent}\ \mathsf{Tgs},\\ \mathsf{Number}\ Ta, authTicket|\!\})$$

and is such that $\neg\mathsf{expiredAK}\ Ta\, evs$*, then*

$$\mathsf{Key}\ authK \notin \mathsf{analz}(\mathsf{spies}\ evs)$$

Refining this result by the authenticity theorem 6.6 produces a confidentiality guarantee for A over the authkey.

An analogous result enforces the confidentiality of the servkey from the Tgs viewpoint under the assumption that the key has not expired and that the authkey associated to it is confidential (theorem 6.19). Frequent appeals to theorems 6.15-6.17 are necessary to the proof. The assumption of authkey confidentiality is clearly indispensable, otherwise the fourth message of the protocol would falsify the conclusion of the theorem. That assumption can be relaxed by theorem 6.18 producing a guarantee that is applicable by Tgs, since Kas and Tgs can inspect each other's functioning. Also, if B were compromised, then the servticket, which the spy observes from the fifth message, would reveal the servkey.

Theorem 6.19. *If B is not compromised and evs contains*

$$\mathsf{Says}\ \mathsf{Tgs}\ A\,(\mathsf{Crypt}\ authK\{\!|\mathsf{Key}\ servK, \mathsf{Agent}\ B, \mathsf{Number}\ Ts, servTicket|\!\})$$

and is such that

$$\mathsf{Key}\ authK \notin \mathsf{analz}(\mathsf{spies}\ evs) \quad \textit{and} \quad \neg\mathsf{expiredSK}\ Ts\ evs$$

then

$$\mathsf{Key}\ servK \notin \mathsf{analz}(\mathsf{spies}\ evs)$$

The servkey confidentiality can be made useful to A (theorem 6.20) using existing authenticity and confidentiality theorems. The first step is applying theorem 6.6 to theorem 6.18 to derive the authkey confidentiality. Then, theorem 6.8 gives that the servkey originated with Tgs, and theorem 6.19 concludes.

Theorem 6.20. *If A and B are not compromised and evs is such that*

$$\begin{aligned}&\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Key}\;\mathit{authK}, \mathsf{Agent}\;\mathsf{Tgs}, \mathsf{Number}\;\mathit{Ta}, \mathit{authTicket}|\!\}\\ &\quad\in \mathsf{parts}(\mathsf{spies}\;\mathit{evs})\quad \mathit{and}\\ &\mathsf{Crypt}\;\mathit{authK}\{\!|\mathsf{Key}\;\mathit{servK}, \mathsf{Agent}\;B, \mathsf{Number}\;\mathit{Ts}, \mathit{servTicket}|\!\}\\ &\quad\in \mathsf{parts}(\mathsf{spies}\;\mathit{evs})\quad \mathit{and}\\ &\neg\mathsf{expiredAK}\;\mathit{Ta}\;\mathit{evs}\quad \mathit{and}\quad \neg\mathsf{expiredSK}\;\mathit{Ts}\;\mathit{evs}\end{aligned}$$

then

$$\mathsf{Key}\;\mathit{servK} \notin \mathsf{analz}(\mathsf{spies}\;\mathit{evs})$$

Investigating the servkey confidentiality from B's viewpoint provides the opportunity for a deeper insight into the protocol (theorem 6.21), revealing a violation of our principle of goal availability (§4.8). The proof develops in a forward style. First, it elaborates on the given history of *servK* deriving the confidentiality of *authK* by theorems 6.6 and 6.18. Then, theorem 6.3 states that *authK* is a session key (it does not belong to the range of shrK), which is necessary to apply theorem 6.8 and derive the origin of *servK*. Certainly B is different from Tgs, otherwise theorems 6.7 and 6.3 would derive that *servK* is an authkey while theorem 6.4 states that it is not. At this stage, theorem 6.9 introduces another possible history of *servK* but the unicity argument for Tgs unifies the two histories. An appeal to theorem 6.19 concludes.

Theorem 6.21. *If A and B are not compromised, and evs is such that*

$$\begin{aligned}&\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Agent}\;A, \mathsf{Agent}\;B, \mathsf{Key}\;\mathit{servK}, \mathsf{Number}\;\mathit{Ts}|\!\}\\ &\quad\in \mathsf{parts}(\mathsf{spies}\;\mathit{evs})\quad \mathit{and}\\ &\mathsf{Crypt}\;\mathit{authK}\{\!|\mathsf{Key}\;\mathit{servK}, \mathsf{Agent}\;B, \mathsf{Number}\;\mathit{Ts}, \mathit{servTicket}|\!\}\\ &\quad\in \mathsf{parts}(\mathsf{spies}\;\mathit{evs})\quad \mathit{and}\\ &\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Key}\;\mathit{authK}, \mathsf{Agent}\;\mathsf{Tgs}, \mathsf{Number}\;\mathit{Ta}, \mathit{authTicket}|\!\}\\ &\quad\in \mathsf{parts}(\mathsf{spies}\;\mathit{evs})\quad \mathit{and}\\ &\neg\mathsf{expiredAK}\;\mathit{Ta}\;\mathit{evs}\quad \mathit{and}\quad \neg\mathsf{expiredSK}\;\mathit{Ts}\;\mathit{evs}\end{aligned}$$

then

$\mathsf{Key}\ servK \notin \mathsf{analz}(\mathsf{spies}\ evs)$

A closer look at the proof just described shows that, after applying theorems 6.6, 6.18 and 6.8, $servTicket = \{\!|A, B, servK, Ts|\!\}_{Kb}$ could be derived by theorem 6.4. Since parts is closed under message decomposition, the first assumption of theorem 6.21 is technically unnecessary. However, the present formulation highlights what B can verify and what he must trust.

In the real world, B can only witness the reception of the servticket, thus verifying the first and fifth assumptions of the theorem. He is certainly not able to verify the second, third and fourth because they pertain to the authentication and authorisation phases, to which he does not participate. Are these assumptions necessary? They signify that the authkey that encrypts the servkey whose confidentiality is being studied has not expired. This is indispensable for the application of theorems 6.6 and 6.18 and enforces the authkey confidentiality, which is itself indispensable to derive the servkey confidentiality due to the form of the fourth message.

An Attack

Because of the assumptions that B cannot verify, theorem 6.21 cannot be applied by B. The theorem reveals that, upon reception of the servticket, B must trust that the preceding phases have not compromised the authkey and consequently the servkey. Therefore, B obtains a weak guarantee of servkey confidentiality. Note that these observations arise from checking whether the protocol conforms to the principle of goal availability.

As a matter of fact, admitting that session keys can only be leaked by accident when they have expired, as does our model, the following attack is possible. The authkey belonging to A expires and the spy gets hold of it, while A is killed and her owner logged out from the workstation; the spy extracts all servkeys associated to that authkey from messages she has previously intercepted; she exploits each of those that have not expired to spoof the fifth message (she only needs to update the timestamp of the authenticator) and access the corresponding network service B.

While B believes to be communicating with A, in fact A does not exist anymore. Moreover, A's owner, who is no longer connected, cannot register any irregularities. So, within the lifetime of servkeys, the spy can obtain access to all services that were in fact granted to the protocol initiator.

Fixing the Attack

It is unsafe to allow servkeys to remain valid when the authkey to which they are associated expires. Preventing this fixes the attack. The aim can be achieved by constraining the Tgs operation with a suitable temporal check to add to rule *K4*

$$(\mathsf{ct}\ evs) + \mathsf{servkLife} \leq Ta + \mathsf{authkLife} \tag{6.1}$$

Note that *Ta* is the timestamp that marks the issue of the authkey, so the check prescribes that the expiry time of the servkey at latest equals that of the authkey.

In the strengthened protocol model, theorem 6.21 no longer needs the second, third and fourth assumptions, so becoming applicable by B. Therefore, B obtains a strong confidentiality guarantee. Although shorter than the old proof, the new one requires some arithmetic. Moreover, B must be explicitly assumed to be different from Tgs otherwise the servticket $\{\!|A, B, servK, Ts|\!\}_{Kb}$ could be misinterpreted as an authticket. Theorem 6.9, updated to enforce also condition 6.1, derives a history of *servK*. From $\neg\mathsf{expiredSK}\ Ts\ evs$ and condition 6.1, we derive that $\neg\mathsf{expiredAK}\ Ta\ evs$. Theorems 6.18 and then 6.19 conclude the proof. Note that the unicity argument is not required because the reasoning only develops along a single history of *servK*.

6.3.6 Authentication

Authentication is one of the main goals of Kerberos IV. Not only does the protocol aim at mutual authentication between the initiator and the network service, but also between the initiator and Tgs. For the sake of completeness, we observe that the first message, which is a cleartext, does not authenticate the initiator to Kas. Thus, the spy may overload the server with fake requests in the same way as with BAN Kerberos (§5.3). The authenticity theorem 6.6 also authenticates Kas to the initiator A providing a guarantee of weak agreement.

The authenticator $\{\!|A, T2|\!\}_{authK}$ of the third message aims at authenticating A to Tgs. However, Tgs merely receiving the authenticator does not enforce the goal: Tgs must also receive the corresponding authticket $\{\!|A, \mathsf{Tgs}, authK, Ta|\!\}_{Ktgs}$ to learn *authK* and be able to decipher the authenticator. If *authK* is confidential in this scenario, then A sent those certificates in an instance of the fourth message (theorem 6.22). Relaxing the last assumption by the confidentiality theorem 6.18 results in a guarantee for Tgs. Note that the regularity lemma guarantees the integrity of the authticket because Tgs is not compromised, while the confidentiality of *authK* establishes that of the authenticator.

The theorem formalises weak agreement of A with Tgs but does not express that A is the true creator of the authenticator, which would give evidence to Tgs that A knows *authK* and was alive at time *T2*. We will prove formally that this stronger goal is met (see §7.4).

Theorem 6.22. *If evs is such that*

$$\mathsf{Crypt}\ authK\ \{\!|\mathsf{Agent}\ A, \mathsf{Number}\ T2|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs) \quad and$$
$$\mathsf{Crypt}(\mathsf{shrK\ Tgs})\{\!|\mathsf{Agent}\ A, \mathsf{Agent\ Tgs}, \mathsf{Key}\ authK, \mathsf{Number}\ Ta|\!\}$$
$$\in \mathsf{parts}(\mathsf{spies}\ evs) \quad and$$
$$\mathsf{Key}\ authK \notin \mathsf{analz}(\mathsf{spies}\ evs)$$

then there exists B such that evs contains

$$\mathsf{Says}\ A\ \mathsf{Tgs}\ \{\!|\mathsf{Crypt}(\mathsf{shrK\ Tgs})\{\!|\mathsf{Agent}\ A, \mathsf{Agent\ Tgs}, \mathsf{Key}\ authK, \mathsf{Number}\ Ta|\!\}, \mathsf{Crypt}\ authK\ \{\!|\mathsf{Agent}\ A, \mathsf{Number}\ T2|\!\}, \mathsf{Agent}\ B|\!\}$$

The proof, which relies on the integrity of the two certificates, requires a substantial simplification to deal with the long inductive formula. The most significant subgoal arises from rule *K3*, which introduces an event of the same form as that asserted by the theorem. If the two events contain different authkeys, the inductive formula terminates the proof, otherwise an appeal to the unicity theorem 6.10 is required.

The authenticity theorem 6.8 guarantees weak agreement of Tgs with A.

The same strategy proves weak agreement of A with B (theorem 6.23). The theorem relies on certificates that have the same form as those of the previous one. However, assuming B not to be Tgs establishes that they contain a servticket. As expected, B must be assumed not to be compromised in order for the servticket to be integral. The theorem will be strengthened to state formally that A knows *servK* and was alive at time *T3* (see §7.4).

Theorem 6.23. *If B is not* Tgs *and is not compromised, and evs is such that*

$$\mathsf{Crypt}\ servK\ \{\!|\mathsf{Agent}\ A, \mathsf{Number}\ T3|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs) \quad and$$
$$\mathsf{Crypt}(\mathsf{shrK}\ B)\{\!|\mathsf{Agent}\ A, \mathsf{Agent}\ B, \mathsf{Key}\ servK, \mathsf{Number}\ Ts|\!\}$$
$$\in \mathsf{parts}(\mathsf{spies}\ evs) \quad and$$
$$\mathsf{Key}\ servK \notin \mathsf{analz}(\mathsf{spies}\ evs)$$

then evs contains

$$\begin{aligned}&\mathsf{Says}\,A\,B\,\{\!|\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Agent}\,A, \mathsf{Agent}\,B, \mathsf{Key}\,servK, \mathsf{Number}\,Ts|\!\},\\&\qquad\qquad \mathsf{Crypt}\,servK\{\!|\mathsf{Agent}\,A, \mathsf{Number}\,T3|\!\}|\!\}\end{aligned}$$

Proving weak agreement of B with A (theorem 6.24) is a little more complicated because the last message of the protocol fails to state B's identity. Fortunately, this lack of explicitness can be overcome by additional checks on A's side. The theorem does not state formally that B knows *servK* or when B was alive (see §7.4).

Theorem 6.24. *If A is not compromised and evs is such that*

$$\begin{aligned}&\mathsf{Crypt}\,servK(\mathsf{Number}\,T3) \in \mathsf{parts}(\mathsf{spies}\,evs) \quad and\\&\mathsf{Crypt}\,authK\{\!|\mathsf{Key}\,servK, \mathsf{Agent}\,B, \mathsf{Number}\,Ts, servTicket|\!\}\\&\quad\in \mathsf{parts}(\mathsf{spies}\,evs) \quad and\\&\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Key}\,authK, \mathsf{Agent}\,\mathsf{Tgs}, \mathsf{Number}\,Ta, authTicket|\!\}\\&\quad\in \mathsf{parts}(\mathsf{spies}\,evs)\\&\mathsf{Key}\,authK \notin \mathsf{analz}(\mathsf{spies}\,evs) \quad and \quad \mathsf{Key}\,servK \notin \mathsf{analz}(\mathsf{spies}\,evs)\end{aligned}$$

then evs contains

$$\mathsf{Says}\,B\,A\,(\mathsf{Crypt}\,servK(\mathsf{Number}\,T3))$$

To understand the theorem, let us recall that the last message of the protocol is a certificate sealed by a servkey. That key must be confidential for the certificate not to be spoof. Upon reception of the certificate, A can derive that the servkey is meant for B by recalling the association established by an instance of the fourth message. This must be sealed by a confidential authkey meant for A in order for the association to be reliable, namely not invented by the spy.

6.3.7 Key Distribution

The current potential of the Inductive Approach is inadequate to reason about key distribution (§4.7), as observed during the analysis of BAN Kerberos (§5.4.7). In the next chapter, we will introduce the necessary extensions to verify this goal (§7.4).

Chapter 7

Modelling Agents' Knowledge of Messages

Agents' knowledge of messages is defined via possession of the messages. Two approaches to formalise the notion within the Inductive Approach are demonstrated and compared.

Reasoning formally about security protocols invites confusion between belief and knowledge. Abstracting from the context may help clarify the difference. If Alice *knows* something in everyday life, then she has sufficient evidence that it is true. If she *believes* something, then she does not necessarily have evidence about it. The BAN logic [31] only captures the notion of belief, though this is often misinterpreted as knowledge. The predicate $P\ \mathsf{believes}\ \phi$ signifies that "P would be entitled to believe X" and that "P may act as though X is true" [31, p.236]. Subsequent research extends the logic with a proper concept of knowledge: Bleeker and Meertens [26] introduce the predicate $P\ \mathsf{rightly\_believes}\ \phi$, which holds when P believes ϕ and ϕ in fact holds.

As demonstrated in the previous chapters, the Inductive Approach provides a meta formalisation of agents' beliefs and knowledge by means of theorems. Those stated from an agent's viewpoint express the agent's knowledge in case the agent can verify all their assumptions. Conversely, if the agent cannot obtain evidence of the truth values of some assumptions, then the theorems express the agent's beliefs. When some assumptions constitute a minimal trust (§4.8), the beliefs cannot become knowledge.

The present chapter provides a formal definition of the agents' knowledge of message components and messages, rather than knowledge that certain events have occurred. This is crucial to verify the main goal of most of the protocols already formalised: key distribution. Each of the peers of a session key should receive guarantees that the other peer knows the same key. Moreover, the definition prepares the Inductive Approach for analysing new hierarchies of protocols. For example, non-repudiation protocols (e.g. [105, 106]) aim at non-repudiation of reception, which requires proving agents' knowledge of specific components upon their reception. E-commerce protocols (e.g. [70]) would not authorise delivery of goods until the merchant obtains assurances that his bank knows the components that correspond to the exact payment. Protocols for group key agreement aim at distributing a key to all agents of the group, providing each agent with evidence that the goal is met (e.g. [9, 91]).

We express knowledge of a component via possession of the component [12]. This notion can be formalised via the ability to actively use the component, which is in turn verifiable by inspecting the history of the network (§7.1). Alternatively, reception of the component also expresses its possession but requires introducing the corresponding event (§7.2). This also improves the readability of the analyses. The outcomes of both approaches are presented in detail on BAN Kerberos (§7.3) and on Kerberos IV (§7.4). The chapter continues with a comparison of the two approaches (§7.5), and finally discusses the outcomes of using timestamps or nonces on the same protocol design (§7.6).

7.1 Agents' Knowledge via Trace Inspection

The agent who creates a message certainly knows all components of the message, including the cryptographic key that eventually seals the message. Unfortunately, a Says event is inadequate to express creation (as observed above, §4.6, §5.4.6, §6.3.6) because it may occur when an agent is merely forwarding an unintelligible message.

However, if we require that a message X has never appeared, even within a larger message, before the event $\mathsf{Says}\,A\,B\,X$ occurs, then A is the true creator of X. This can be established by inspecting the history that precedes the event. Since each history is recorded by a trace, enforcing the property simply requires a trace inspection. We declare the predicate

$$\mathsf{Issues} : [\mathsf{agent}, \mathsf{agent}, \mathsf{msg}, \mathsf{event\ list}]$$

so that A Issues B with X on evs holds when A creates message X for B on trace evs. Events may occur more than once and recent events are appended to the head of traces. Therefore, detecting the first occurrence through time of an event *ev* on a trace *evs* requires scanning *evs* in reverse, namely from tail to head. Traces are lists, and Isabelle provides numerous functions for reasoning about lists. The unary rev reverses a list, while the binary takeWhile takes a predicate and a list, scans the list and returns all elements of the list until these verify the predicate. All other functions for defining Issues are known.

$$\begin{aligned}&A \ \mathsf{Issues}\ B \ \mathsf{with}\ X \ \mathsf{on}\ evs \equiv\\ &\quad \exists Y.\ \mathsf{Says}\, A\, B\, Y \in \mathsf{set}\, evs \quad \text{and} \quad X \in \mathsf{parts}\{Y\} \quad \text{and}\\ &\quad X \notin \mathsf{parts}(\mathsf{spies}\,(\mathsf{takeWhile}(\lambda x.\, x \neq \mathsf{Says}\, A\, B\, Y)(\mathsf{rev}\ evs)))\end{aligned}$$

It can be seen that the predicate requires that A has sent X, possibly inside a larger message, and that X never appears in the traffic preceding such event.

7.1.1 Basic Lemmas

A few technical lemmas are necessary to reason about reversed lists. All of them are provable by induction on the relevant trace. For example, the traffic on a trace whose first element is a Says event amounts to the traffic on the subtrace without the event plus the message introduced by that event (lemma 7.1). Recall that @ is the Isabelle symbol for the list concatenation operator.

Lemma 7.1. $\mathsf{spies}\,(evs\ @\ [\mathsf{Says}\, A\, B\, X]) = \{X\} \cup (\mathsf{spies}\ evs)$.

Together with the analogous law concerning the Notes event (omitted here), lemma 7.1 serves to establish an obvious result formally: the traffic on a reversed trace is the same as that on the original trace (lemma 7.2).

Lemma 7.2. $\mathsf{spies}\ evs = \mathsf{spies}\,(\mathsf{rev}\, evs)$.

Another technically important result states that the traffic on a subtrace obtained via the takeWhile function is a subset of the traffic on the whole set (lemma 7.3).

Lemma 7.3. $\mathsf{spies}\,(\mathsf{takeWhile} P\ evs) \subseteq (\mathsf{spies}\ evs)$.

7.1.2 Proving Knowledge

A general strategy can be devised to prove significant guarantees in terms of the Issues predicate.

Let us suppose that a message X is a component of a larger message Y, and that the event $\mathsf{Says}\,A\,B\,Y$ occurs on a trace *evs*. If A is the true creator of X, then (and only then) we can prove that A Issues B with X on *evs* holds. Some assumptions are necessary to ensure that the spy cannot forge X before A actually creates it: if X is a compound message, then we must assume that the spy cannot analyse its components from the traffic or synthesise them; if X is a ciphertext, then we must assume the confidentiality of the key that seals it. Also, A must be assumed not to be the spy, so she acts legally according to the protocol. Further assumptions may be required on A and B (see §7.3.1, §7.4.1) depending on the protocol. The proof develops through the following strategy.

- Simplify the main subgoal by the definition of Issues.

- Isolate the first two conjuncts of the definition of Issues by proceeding in a backward style (resolving by the introduction rules for existential quantification first, and then conjunction).

- Prove the first conjunct, $\mathsf{Says}\,A\,B\,Y$, by assumption, and the second conjunct, $X \in \mathsf{parts}\{Y\}$, by symbolic evaluation of the parts operator.

- Apply structural induction over the protocol model to verify that all steps of the protocol definition preserve the following property: the occurrence of the event $\mathsf{Says}\,A\,B\,Y$ on a trace implies that Y never appears on the trace before that event.

- Simplify all subgoals.

- Prove the subgoal corresponding to the protocol step where the event $\mathsf{Says}\,A\,B\,Y$ takes place by applying lemmas 7.3 and 7.2, a few trivial ones, and finally a lemma introducing the event $\mathsf{Says}\,A\,B\,Y$ on the available assumptions (see §7.3.1, §7.4.1).

Although the theorem has limited importance in itself, since it merely says that an agent knows the components of the messages he sends, it is particularly useful to refine other theorems that enforce the event $\mathsf{Says}\,A\,B\,Y$. For example, using this strategy we have investigated the goals of non-injective agreement and key distribution on all protocols analysed so far, as demonstrated below on BAN Kerberos and Kerberos IV.

7.2 Agents' Knowledge via Message Reception

The extensions to the Inductive Approach mentioned in this section have been released with the 1999 distribution of Isabelle.

A primitive modelling message reception can be introduced [11] by extending the Isabelle datatype for events (§3.1.3) as follows

```
datatype event = Says  agent agent msg
               | Notes agent       msg
               | Gets  agent       msg
```

In the real world, a message can be received only if it was previously sent. This *reception invariant* can be easily enforced by the protocol model (see §7.2.2).

Technically speaking, the Notes event could be replaced by the Gets event imagining that when an agent notes down a message it is as if the agent received it from the network. However, this would compromise the reception invariant. Keeping the two events separate allows reasoning that turns out to be more readable, more faithful to reality and, ultimately, simpler. For example, thanks to the reception invariant, the messages that are received do not need to enrich the set of components used on a trace. The definition of used (§3.3) must be extended with the following rule

– $\mathsf{used}((\mathsf{Gets}\, A\, X) \# \mathit{evs}) \triangleq \mathsf{used}\; \mathit{evs}$

7.2.1 From Spy's Knowledge to Agents' Knowledge

The major outcome of introducing message reception is a realistic formalisation of agents' knowledge. The rudimentary version available initially within the Inductive Approach [87] was soon enhanced to express knowledge of a single agent, the spy, by means of the function spies [88, §3.5] (§3.2). The previous chapters have demonstrated that this function allows reasoning about confidentiality but not about key distribution. We generalise spies to a binary function that captures any agent's knowledge

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

The additional parameter represents the agent whose knowledge is being defined. The entire range of the function initState (§3.2) is now relevant to the rest of the treatment. The definition of knows generalises that of spies in the base case and the two inductive steps corresponding to the existing events. A third inductive step becomes necessary to account for the Gets event.

0. An agent knows his initial state.

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

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

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

2. An agent knows what he notes on a trace; the spy also knows the 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 receives on a trace. The spy's knowledge must not be extended with any of the received messages since the spy already knows them by case 1 and by the reception invariant.

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

According to the initial definition of agents' knowledge [87, §4.5], an agent A could *see* a message X when X had been sent by somebody to A. However, this could not ensure that X had been delivered to A. The introduction of the reception event allows a more faithful treatment of the matter through the last step of the definition of knows.

Recall that the function analz is applied to a set of messages and recursively extracts all components of compound messages and bodies of messages encrypted under keys that are known. If, in the real world, an agent A knows a message X, the protocol model contains a trace evs such that either $X \in (\mathsf{knows}\, A\ \mathit{evs})$ if A did not need any decryption to get hold of X, or $X \in \mathsf{analz}(\mathsf{knows}\, A\ \mathit{evs})$ if A had to retrieve X from within a larger message of the set $\mathsf{knows}\, A\ \mathit{evs}$. Therefore, also friendly agents may now need to apply the function analz in some circumstances.

7.2.2 Updating the Existing Models

Let prot be a formal protocol model. The reception invariant (that a message can be received only if it was sent) can be enforced by adding an inductive rule to the definition of the protocol. The rule, which is called *Reception* (figure 7.1), allows extending a trace of prot that contains an event Says $A\,B\,X$ with the event Gets $B\,X$.

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

Figure 7.1: Rule for message reception

Note that, since rules are never forced to fire on any trace, no Gets event can be forced to take place. Therefore, no guarantee can be established that a message that was sent will be ever received by the intended recipient, as is realistic in a setting where the spy can prevent message delivery. Furthermore, each rule can fire more than once so that the same message can be received more than once, or fire in the wrong order so the order in which messages are sent is not necessarily preserved upon their reception. Note also that agents discard the sender label upon reception because it is not reliable on an insecure network.

The protocol model needs further updates. If the event Says $P\,Q\,X$ appears in the premises of a rule and P does not appear in the conclusions, then the event can be replaced by Gets $Q\,X$. In the old model, Q acted when some other (undefined) agent had sent him X. However, no agent but the spy can monitor events performed by other agents, so the condition was not directly verifiable by Q until X was received. Thus, the new model expresses Q's behaviour more closely. Figure 7.2 shows a fragment of the updated model for BAN Kerberos, derived from that in figure 5.2, §5.3. Note that the *Fake* rule contains knows Spy rather than spies. The occurrences of spies in all existing theories must be updated similarly.

7.2.3 Basic Lemmas

Let prot be a protocol model whose definition includes the *Reception* rule. Let *evs* be a generic trace of prot. Induction easily proves that prot satisfies the reception invariant (lemma 7.4).

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

BK2
[| evs ∈ bankerberos; Key K ∉ used evs;
   Gets Server {|Agent A, Agent B|} ∈ set evs |]
⟹ Says Server A (Crypt (shrK A) {|Number (ct evs), Agent B, Key K,
                  Crypt (shrK B) {|Number (ct evs), Agent A, Key K|}|})
      # evs ∈ bankerberos

BK3
[| evs ∈ bankerberos;
   Says A Server {|Agent A, Agent B|} ∈ set evs;
   Gets A (Crypt (shrK A) {|Number Ts, Agent B, Key K, Ticket|})
     ∈ set evs;
   ¬ expiredS Ts evs |]
⟹ Says A B {|Ticket, Crypt K {|Agent A, Number (ct evs)|}|}
      # evs ∈ bankerberos
```

Figure 7.2: Updating the BAN Kerberos model (fragment)

Lemma 7.4. *If evs contains* $\mathsf{Gets}\, B\, X$*, then there exists* A *such that evs also contains* $\mathsf{Says}\, A\, B\, X$*.*

Applying this lemma and an existing one (omitted here) stating that the spy knows the messages that have been sent, we can prove that she also knows all messages that have been received (lemma 7.5).

Lemma 7.5. *If evs contains* $\mathsf{Gets}\, B\, X$*, then* $X \in (\mathsf{knows}\,\mathsf{Spy}\ evs)$*.*

Reading the lemma when B is the spy, we derive that the spy knows the messages she receives. The last case of the definition of knows allows this result to be generalised: any agent knows what he receives (lemma 7.6).

Lemma 7.6. *If evs contains* $\mathsf{Gets}\, B\, X$*, then* $X \in (\mathsf{knows}\, B\ evs)$*.*

Resolving lemma 7.5 with the lemma $H \subseteq \mathsf{parts}\, H$, we obtain that messages that are received appear in the traffic (lemma 7.7).

Lemma 7.7. *If evs contains* $\mathsf{Gets}\, B\, X$*, then* $X \in \mathsf{parts}(\mathsf{knows}\,\mathsf{Spy}\ evs)$*.*

7.2.4 Updating the Existing Theorems

Lemma 7.7 allows significant modifications to some theorems proved of the protocols analysed thus far.

In consequence, some theorems become available (according to our definition of §4.8) to the agents. For example, the unicity theorem 5.5 for BAN Kerberos can be made available to agents who are not compromised (theorem 7.8). The proof rests on a double application of lemma 7.7; a double application of the authenticity theorem 5.3 then introduces the necessary assumptions to apply the unicity theorem 5.5.

Theorem 7.8. *If A is not compromised and evs, belonging to* bankerberos, *contains*

$$\begin{array}{l}\mathsf{Gets}\,A\,(\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,B, \mathsf{Key}\,Kab,\ Ticket|\!\})\quad \textit{and}\\ \mathsf{Gets}\,A\,(\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Number}\ Tk', \mathsf{Agent}\,B', \mathsf{Key}\,Kab,\ Ticket'|\!\})\end{array}$$

then

$$Tk = Tk' \quad \textit{and} \quad B = B' \quad \textit{and} \quad Ticket = Ticket'$$

Therefore, should an agent who is not compromised receive the same session key within two different messages, she could suspect that something has gone wrong (unknown to our formalisation). Expectedly, the Unique predicate cannot be proved to hold on any reception event, for any message can be received more than once (§7.2.2).

Other theorems become more readable and faithful to reality. For example, authentication of A to B for Kerberos IV can be established upon B's reception of a suitable message (theorem 7.9). The proof applies lemma 7.7, then another stating that parts is closed under message decomposition, and finally the existing authentication theorem 6.23.

Theorem 7.9. *If B is not* Tgs *and is not compromised, and evs, belonging to* kerberos, *contains*

$$\begin{array}{l}\mathsf{Gets}\,B\ \{\!|\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Agent}\,A, \mathsf{Agent}\,B, \mathsf{Key}\,Kab, \mathsf{Number}\ Ts|\!\},\\ \qquad\qquad \mathsf{Crypt}\,Kab\{\!|\mathsf{Agent}\,A, \mathsf{Number}\ T3|\!\}|\!\}\quad \textit{and}\end{array}$$

and $Kab \notin$ analz(knows Spy *evs*)*, then evs contains*

$$\begin{array}{l}\mathsf{Says}\,A\,B\ \{\!|\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Agent}\,A, \mathsf{Agent}\,B, \mathsf{Key}\,Kab, \mathsf{Number}\ Ts|\!\},\\ \qquad\qquad \mathsf{Crypt}\,Kab\{\!|\mathsf{Agent}\,A, \mathsf{Number}\ T3|\!\}|\!\}\end{array}$$

7.2.5 Proving Knowledge

The crucial fact about agents' knowledge is lemma 7.6: any agent knows what he receives. Applying $H \subseteq \mathsf{analz}\,H$, we derive that an agent knows all components of the received messages that he can decrypt.

Hence, when designing guarantees for an agent A, we can inform her of B's knowledge of all components of X by proving that an event $\mathsf{Gets}\,B\,X$ took place. However, proving that the event occurred depends on the protocol under analysis. As examples, we show below how to prove the goal of key distribution for BAN Kerberos (see §7.3.2) and for Kerberos IV (see §7.4.2).

7.3 Revisiting the Guarantees on BAN Kerberos

We have tested both approaches to agents' knowledge on all existing protocol analyses. This section presents the outcome on the verification of BAN Kerberos concentrating on the goals of authentication and key distribution, which receive the most significant benefits. Other outcomes have been mentioned above (§7.2.4).

For the sake of readability, the statement "K is confidential on *evs*" will replace $K \notin \mathsf{analz}(\mathsf{knows}\,\mathsf{Spy}\ \mathit{evs})$.

7.3.1 Using Trace Inspection

The general strategy described above (§7.1.2) may be applied to prove that if a friendly agent sends an instance of the third message of BAN Kerberos, then that agent indeed created the message (theorem 7.10). The proof script may be found in Appendix B.

Theorem 7.10. *If A is not the spy and evs, belonging to* bankerberos, *contains*

$$\mathsf{Says}\,A\,B\,\{\!|\,\mathit{Ticket}, \mathsf{Crypt}\,\mathit{Kab}\{\!|\mathsf{Agent}\,A, \mathsf{Number}\ \mathit{Ta}|\!\}\,|\!\}$$

and Kab is confidential on evs, then

$$A\ \mathsf{Issues}\ B\ \mathsf{with}\ (\mathsf{Crypt}\,\mathit{Kab}\{\!|\mathsf{Agent}\,A, \mathsf{Number}\ \mathit{Ta}|\!\})\ \mathsf{on}\ \mathit{evs}$$

As anticipated by the general strategy, the most difficult subgoal to prove arises from the case formalising the third step of the protocol, which we quote below.

```
 [| A ≠ Spy; evs3 ∈ bankerberos;
    Says S A (Crypt (shrK A) {|Number Tk, Agent B, Key Kab, Ticket|})
      ∈ set evs3;
    Says A Server {|Agent A, Agent B|} ∈ set evs3;
    Key Kab ∉ analz (spies evs3);
    Says A B {|Ticket, Crypt Kab {|Agent A, Number (ct evs3)|}|}
      ∉ set evs3;
    Crypt Kab {|Agent A, Number (ct evs3)|}
      ∈ parts
         (spies
           (takeWhile
             (λz. z ≠ Says A B {|Ticket,
                                  Crypt Kab {|Agent A, Number (ct evs3)|}|})
             (rev evs3))) |]
 ==> False
```

If we resolve the monotonicity law for parts with lemma 7.3, we obtain a lemma with which the last assumption of the subgoal can be resolved, producing

$$\mathsf{Crypt}\, Kab \{\!|\mathsf{Agent}\, A, \mathsf{Number}(\mathsf{ct}\; evs3)|\!\} \in \mathsf{parts}(\mathsf{spies}\,(\mathsf{rev} evs3))$$

Then, the monotonicity law for parts and lemma 7.2 give that the authenticator belongs to $\mathsf{parts}(\mathsf{spies}\, evs3)$, namely it appears in the traffic. At this stage, the proof would terminate by application of the authentication theorem 5.10 but not all its assumptions are yet available. So, we try to fetch them. From the third assumption of the subgoal, we derive that

$$\mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|\mathsf{Number}\; Tk, \mathsf{Agent}\, B, \mathsf{Key}\, Kab,\; Ticket|\!\} \in \mathsf{analz}(\mathsf{spies}\; evs3)$$

(referred to as condition 7.1 below) and therefore A certainly is not compromised, otherwise Kab would not be confidential. By theorem 5.3, the agent S mentioned by the third assumption is in fact the server. Then, theorem 5.1 derives that the ticket in the intelligible form is in the traffic on *evs3*. One last assumption is missing for applying theorem 5.10: B must not be compromised. The subgoal does not give sufficient information to derive this fact. However, the initial level of the proof tree did. We backtrack to that level and derive that

$$Ticket \in \mathsf{analz}(\mathsf{spies}\; evs)$$

since the ticket appears in a compound message. Also, since A is a friendly agent, we can prove by induction that the form of the ticket she is sending in the fourth message is the same as that established by the server. Therefore,

$$Ticket = \mathsf{Crypt}(\mathsf{shrK}\, B)\{\!|\mathsf{Number}\; Tk, \mathsf{Agent}\, A, \mathsf{Key}\, Kab|\!\}$$

for some Tk. From the last two conditions, it follows that B must not be compromised, otherwise Kab would not be confidential (this could not follow from condition 7.1 because A is not compromised). Hence, the proof proceeds as mentioned above and the last subgoal can terminate as indicated.

The authentication theorem 5.10 can be resolved with the first assumption of theorem 7.10. To enhance readability, we use here the version of theorem 5.10 where the confidentiality assumption on the session key has not yet been relaxed by theorem 5.9. We obtain a guarantee available to B conveying A's knowledge of Kab and non-injective agreement of A with B on Kab (theorem 7.11). The theorem assumes A to be friendly, so she acts legally. This implies that she was alive at time Ta.

Theorem 7.11. *If A is not the spy, B is not compromised, and evs, belonging to* bankerberos*, is such that*

$\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,A, \mathsf{Key}\,Kab|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs)$ *and*

$\mathsf{Crypt}\,Kab\{\!|\mathsf{Agent}\,A, \mathsf{Number}\ Ta|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs)$

and Kab is confidential on evs, then

$A\ \mathsf{Issues}\ B\ \mathsf{with}\ (\mathsf{Crypt}\,Kab\{\!|\mathsf{Agent}\,A, \mathsf{Number}\ Ta|\!\})\ \mathsf{on}\ evs$

The same procedure relies on the authentication theorem 5.11 to prove the analogous guarantee for A (theorem 7.12). The theorem informs A that B knows Kab and establishes non-injective agreement of B with A on Kab.

Theorem 7.12. *If A is not compromised, B is not the spy, and evs, belonging to* bankerberos*, is such that*

$\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Number}\ Tk, \mathsf{Agent}\,B, \mathsf{Key}\,Kab, Ticket|\!\}$
$\quad \in \mathsf{parts}(\mathsf{spies}\ evs)$ *and*

$\mathsf{Crypt}\,Kab(\mathsf{Number}\ Ta) \in \mathsf{parts}(\mathsf{spies}\ evs)$

and Kab is confidential on evs, then

$B\ \mathsf{Issues}\ A\ \mathsf{with}\ (\mathsf{Crypt}\,Kab(\mathsf{Number}\ Ta))\ \mathsf{on}\ evs$

Unfortunately, the protocol only requires B to reply with an incremented Ta, so A only understands that B was alive after Ta. In other words, Ta is being used as a nonce. However, if B inserted the current time in place of Ta, then A would be informed of the exact instant when B was alive, which is a desirable outcome from the use of timestamps.

Theorems 7.11 and 7.12 formally prove that BAN Kerberos also achieves the goal of key distribution. In the next section, we shall see how the same goal is verified using message reception.

7.3.2 Using Message Reception

As seen above, proving the goal of key distribution on a protocol signifies proving to each agent who has obtained a session key via the protocol that the peer knows the same session key. Using the approach based on message reception for reasoning about agents' knowledge, we aim at proving that one of the peers received the session key on assumptions that the other peer can verify (§7.2.5). BAN Kerberos allows these proofs, thus achieving the goal of key distribution.

Let us suppose that an agent B who is not compromised receives an instance of the third message of the protocol that quotes an agent A who is not the spy. Assuming that this session key is confidential, the session key must be known to A (theorem 7.13).

Theorem 7.13. *If A is not the spy, B is not compromised, and evs, belonging to* bankerberos*, contains*

$$\begin{aligned}&\mathsf{Gets}\,B\,\{\!|\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Number}\,Tk, \mathsf{Agent}\,A, \mathsf{Key}\,Kab|\!\}, \\ &\qquad\quad \mathsf{Crypt}\,Kab\{\!|\mathsf{Agent}\,A, \mathsf{Number}\,Ta|\!\}|\!\}\end{aligned}$$

and Kab is confidential on evs, then

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

The proof is simple. Theorem 5.10, which authenticates A to B, can be updated according to the guidelines given above (§7.2.4) so that its conclusion can be established on the assumptions of theorem 7.13. This is why B must not be compromised. Then, having that A sends an instance of the third message, if she is a friendly agent, we can derive by induction that she received an instance of the second message containing Kab and therefore can extract the key by definition of knows .

Along with the updated version of theorem 5.10, theorem 7.13 establishes non-injective agreement of A with B on Kab.

Using the same strategy on the authentication theorem 5.11, we can prove B's knowledge of the session key on assumptions verifiable by A (theorem 7.14). The two theorems together establish non-injective agreement of B with A on Kab.

Theorem 7.14. *If A is not compromised, B is not the spy, and evs, belonging to* bankerberos*, contains*

$$\mathsf{Gets}\,A\,(\mathsf{Crypt}\,Kab(\mathsf{Number}\,Ta))\quad \textit{and}$$

$$\mathsf{Gets}\,A\,(\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Number}\,Tk, \mathsf{Agent}\,B, \mathsf{Key}\,Kab,\,Ticket|\!\})$$

and Kab is confidential on evs, then

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

Theorems 7.13 and 7.14 also signify that BAN Kerberos achieves the goal of key distribution.

7.4 Revisiting the Guarantees on Kerberos IV

Both approaches to agents' knowledge scale up to proving non-injective agreement and key distribution on Kerberos IV. The proofs are, as expected, longer than those for BAN Kerberos because distinguishing between the two kinds of session keys and tickets often requires case analyses.

As above, for the sake of readability, the statement "K is confidential on evs" will replace $K \notin \mathsf{analz}(\mathsf{knows}\,\mathsf{Spy}\ \ evs)$.

7.4.1 Using Trace Inspection

As previously mentioned, the authenticity theorem 6.6 establishes weak agreement of Kas with A. The general strategy (§7.1.2) may be used to establish that, if Kas sends an instance of the second message, then Kas is indeed issuing a message that never appeared before. Resolving the assumption of this result by theorem 6.6, we provide A with evidence that Kas knows the authkey mentioned in the message. The theorem (omitted here) establishes non-injective agreement of Kas with A on the authkey.

If A is a friendly agent sending an instance of the third message, then we can prove that A is the creator of the authenticator that the message contains. Note that, while Kas is certainly not the spy thanks to the injections created by the Isabelle datatype for messages (§3.1.2), the generic agent A must be explicitly assumed to be friendly in order for the result to hold. Combining this with the authentication theorem 6.22, we obtain a guarantee of non-injective agreement of A with Tgs on the authkey $authK$ (theorem 7.15). Relaxing the confidentiality assumption on $authK$ by theorem 6.18, the guarantee can be applied by Tgs within the minimal trust that A is not compromised. The theorem also informs Tgs that A was alive at time $T2$.

Theorem 7.15. *If A is not the spy and evs, belonging to* kerberos*, is such that*

$$\mathsf{Crypt}\,authK\{\!|\mathsf{Agent}\,A, \mathsf{Number}\ T2|\!\} \in \mathsf{parts}(\mathsf{spies}\ \ evs) \quad \textit{and}$$

$$\mathsf{Crypt(shrK\ Tgs)}\{\!|\mathsf{Agent}\ A, \mathsf{Agent\ Tgs}, \mathsf{Key}\ authK, \mathsf{Number}\ Ta|\!\}$$
$$\in \mathsf{parts(spies}\ evs)$$

and authK is confidential on evs, then

$$A\ \mathsf{Issues\ Tgs\ with}\ (\mathsf{Crypt}\ authK\{\!|\mathsf{Agent}\ A, \mathsf{Number}\ T2|\!\})\ \mathsf{on}\ evs$$

Like Kas, Tgs also acts legally. So, we can prove that if Tgs sends an instance of the fourth message, then the message is new. This result combined with theorem 6.8 (which expressed weak agreement of Tgs with A) yields a guarantee of non-injective agreement of Tgs with A on the keys *authK* and *servK* (theorem 7.16). The guarantee also tells A that Tgs was alive at time Ts.

Theorem 7.16. *If B is not* Tgs *and evs, belonging to* kerberos*, is such that*

$$\mathsf{Crypt}\ authK\{\!|\mathsf{Key}\ servK, \mathsf{Agent}\ B, \mathsf{Number}\ Ts, servTicket|\!\}$$
$$\in \mathsf{parts(spies}\ evs)$$

and authK is confidential on evs, then

$$\mathsf{Tgs\ Issues}\ A\ \mathsf{with}$$
$$(\mathsf{Crypt}\ authK\{\!|\mathsf{Key}\ servK, \mathsf{Agent}\ B, \mathsf{Number}\ Ts, servTicket|\!\})\ \mathsf{on}\ evs$$

We can also prove that a friendly agent who sends an instance of the fourth message does create the authenticator included in the message. Resolving the conclusion of theorem 6.23 with the main assumption of the stated result, we derive non-injective agreement of A with B on the servkey *servK* (theorem 7.17). The guarantee also tells B that A was alive at time $T3$.

Theorem 7.17. *If A is not the spy, B is not* Tgs *and is not compromised, and evs, belonging to* kerberos*, is such that*

$$\mathsf{Crypt}\ servK\{\!|\mathsf{Agent}\ A, \mathsf{Number}\ T3|\!\} \in \mathsf{parts(spies}\ evs)\quad and$$
$$\mathsf{Crypt(shrK}\ B)\{\!|\mathsf{Agent}\ A, \mathsf{Agent}\ B, \mathsf{Key}\ servK, \mathsf{Number}\ Ts|\!\}$$
$$\in \mathsf{parts(spies}\ evs)$$

and servK is confidential on evs, then

$$A\ \mathsf{Issues}\ B\ \mathsf{with}\ (\mathsf{Crypt}\ servK\{\!|\mathsf{Agent}\ A, \mathsf{Number}\ T3|\!\})\ \mathsf{on}\ evs$$

Finally, if a friendly agent B sends an instance of the last message sealed by a confidential servkey, then the message can be proved to be new. This result can be used to refine theorem 6.24, so expressing non-injective agreement of B with A on the servkey $servK$ (theorem 7.18). Also, B does not cheat because he is assumed to be friendly. Therefore, as observed on BAN Kerberos, A learns that B was alive after Ta but could get a stronger information if B replaced Ta with a fresh timestamp.

Theorem 7.18. *If A is not compromised, B is not the spy, and evs, belonging to* kerberos, *is such that*

$$\begin{aligned}&\mathsf{Crypt}\,servK(\mathsf{Number}\ T3) \in \mathsf{parts}(\mathsf{spies}\ evs) \quad \textit{and}\\ &\mathsf{Crypt}\,authK\{\!|\mathsf{Key}\ servK, \mathsf{Agent}\ B, \mathsf{Number}\ Ts, servTicket|\!\}\\ &\quad \in \mathsf{parts}(\mathsf{spies}\ evs) \quad \textit{and}\\ &\mathsf{Crypt}(\mathsf{shrK}\ A)\{\!|\mathsf{Key}\ authK, \mathsf{Agent}\ \mathsf{Tgs}, \mathsf{Number}\ Ta, authTicket|\!\}\\ &\quad \in \mathsf{parts}(\mathsf{spies}\ evs)\end{aligned}$$

and both $authK$ and $servK$ are confidential on evs, then

$$B\ \mathsf{Issues}\ A\ \mathsf{with}\ (\mathsf{Crypt}\,servK(\mathsf{Number}\ T3))\ \mathsf{on}\ evs$$

Theorems 7.17 and 7.18 signify that the Kerberos IV guarantees key distribution. The goal concerns a servkey and its peers. In addition, theorems 7.15 and 7.16 signify that key distribution is also met between the protocol initiator and Tgs on an authkey. In the next section, we shall see how to study the same goals by message reception.

7.4.2 Using Message Reception

The entire hierarchy of theorems presented below rests on a unique philosophy: the authenticity (§6.3.3) or authentication (§6.3.6) theorems updated by suitable reception events (§7.2.4) provide guarantees of weak agreement. Using the definition of knows we prove that specific agents have knowledge of specific session keys. Combining these guarantees, we derive non-injective agreement on the session keys.

If an agent who is not compromised receives the instance of the second message that is sealed by her shared key, an appeal to lemma 7.7 derives the necessary assumptions to apply theorem 6.6. The resulting theorem, stating that Kas sent the message received by the agent, establishes weak agreement of Kas with A. Since Kas knows all shared keys, it can extract and learn the authkey contained in the message. These two theorems, whose formal statement is omitted here, guarantee non-injective agreement of Kas with A on the authkey.

The updated authentication theorem 6.22 states that, upon reception by Tgs of an instance of the third message that includes a confidential authkey, the agent mentioned by the authenticator sent the message. Let A be such agent. If A is friendly, induction proves that A sends the third message only upon reception of the second, from which she can extract the authkey. Hence, Tgs can be assured that A knows the authkey (theorem 7.19). This and the updated version of 6.22 assert non-injective agreement of A with Tgs on $authK$.

Theorem 7.19. *If A is not the spy and evs, belonging to* $\mathsf{kerberos}$*, contains*

$$\begin{aligned}\mathsf{Gets\ Tgs}\ \{\!|&\mathsf{Crypt(shrK\ Tgs)}\{\!|\mathsf{Agent}\ A, \mathsf{Agent\ Tgs}, \mathsf{Key}\ authK, \mathsf{Number}\ Ta|\!\},\\ &\mathsf{Crypt}\ authK\{\!|\mathsf{Agent}\ A, \mathsf{Number}\ T2|\!\}, \mathsf{Agent}\ B|\!\}\end{aligned}$$

and authK is confidential on evs, then

$$\mathsf{Key}\ authK \in \mathsf{analz}(\mathsf{knows}\ A\ \ evs)$$

The authenticity theorem 6.8 can be proved assuming the protocol initiator's reception of an instance of the fourth message. If A is the initiator, then the guarantee conveys weak agreement of Tgs with A because it states that Tgs sent the message to A. Therefore, Tgs must have received the suitable instance of the third message, learning the authkey from the authticket. By definition of knows, Tgs also knows the servkey that it has associated to the authkey. Agent A can be therefore informed that Tgs knows both session keys (theorem 7.20). We have thus established non-injective agreement of Tgs with A on $authK$ and $servK$.

Theorem 7.20. *If B is not* Tgs *and evs, belonging to* $\mathsf{kerberos}$*, contains*

$$\mathsf{Gets}\ A\ \mathsf{Crypt}\ authK\{\!|\mathsf{Key}\ servK, \mathsf{Agent}\ B, \mathsf{Number}\ Ts, servTicket|\!\}$$

and authK is confidential on evs, then

$$\begin{aligned}&\mathsf{Key}\ authK \in \mathsf{analz}(\mathsf{knows\ Tgs}\ evs)\ \ \textit{and}\\ &\mathsf{Key}\ servK \in \mathsf{analz}(\mathsf{knows\ Tgs}\ evs)\end{aligned}$$

Theorem 7.9, expressing weak agreement of A with B, is the updated version of theorem 6.23. If A is friendly, she only sends an instance of the fifth message upon reception of a suitable instance of the fourth, from which she can extract the servkey. So, B is informed that A knows the servkey (theorem 7.21). The two theorems establish non-injective agreement of A with B on $servK$.

Theorem 7.21. *If A is not the spy, B is not* Tgs *and is not compromised, and evs belonging to* kerberos*, contains*

$$\begin{aligned}&\mathsf{Gets}\, B\, \{\!|\mathsf{Crypt}(\mathsf{shrK}\, B)\{\!|\mathsf{Agent}\, A, \mathsf{Agent}\, B, \mathsf{Key}\, servK, \mathsf{Number}\, Ts|\!\},\\ &\qquad \mathsf{Crypt}\, servK\{\!|\mathsf{Agent}\, A, \mathsf{Number}\, T3|\!\}|\!\}\end{aligned}$$

and servK is confidential on evs, then

$$\mathsf{Key}\, servK \in \mathsf{analz}(\mathsf{knows}\, A\ evs)$$

Finally, the athentication theorem 6.24 can be updated in terms of reception of the messages it mentions. A friendly agent only sends the last message of the protocol upon reception of the last but one. He can therefore extract the servkey from the received servticket. This strategy informs the initiator A that the servkey she has received is also known to the responder B (theorem 7.22). As a consequence, the protocol grants non-injective agreement of B with A on $servK$.

Theorem 7.22. *If A is not compromised, B is not the spy and evs belonging to* kerberos*, contains*

$$\begin{aligned}&\mathsf{Gets}\, A\, \{\!|\mathsf{Crypt}\, authK\{\!|\mathsf{Key}\, servK, \mathsf{Agent}\, B, \mathsf{Number}\, Ts, servTicket|\!\},\\ &\qquad \mathsf{Crypt}\, servK(\mathsf{Number}\, T3)|\!\}\quad \textit{and}\\ &\mathsf{Gets}\, A\ \mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|\mathsf{Key}\, authK, \mathsf{Agent}\,\mathsf{Tgs}, \mathsf{Number}\, Ta, authTicket|\!\}\end{aligned}$$

and both authK and servK are confidential on evs, then

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

Theorems 7.19 and 7.20 show that Kerberos IV achieves the goal of key distribution between the protocol initiator and Tgs on an authkey. Theorems 7.21 and 7.22 guarantee the goal of key distribution between initiator and responder on a servkey.

7.5 Comparing the Two Approaches

The main aim of formalising agents' knowledge was to investigate the goals of non-injective agreement and key distribution. We have verified them on all classical protocols analysed so far (e.g. Needham-Schroeder, Yahalom, Otway-Rees, Woo-Lam, etc.) using both our approaches to knowledge (§7.1, §7.2). Therefore, the approaches can only be compared in respect to the two goals.

The analysis of BAN Kerberos supports the claim that the two approaches might be equivalent. Theorems 7.11 and 7.12, obtained by trace inspection, appear to convey the same guarantees as theorems 7.13 and 7.14 respectively, which are obtained by message reception.

Since the approach based on trace inspection cannot rely on a formalisation of the instant of reception, it must refer to some earlier time: messages are considered when they appear in the traffic. In this light, that approach could be argued to be stronger because it can enforce guarantees since some earlier time than the approach based on message reception does. Nevertheless, no theorem has practical relevance unless its assumptions are verifiable. The guarantees proved by either approach become applicable at the same time, namely upon reception of the suitable messages. Also Kerberos IV confirms the equivalence: theorems 7.15 to 7.18 appear to be equivalent to theorems 7.19 to 7.22 respectively.

As demonstrated above, the guarantees obtained by trace inspection on protocols based on timestamps also add a temporal requisite to the goal of authentication: establishing that a friendly agent creates a message containing the current time as a timestamp also expresses *when* the agent was alive. The approach based on message reception clearly cannot express this, so the other approach will be used to compare the temporal requisites that timestamps or nonces add to a protocol design (see §7.6).

Note that both versions of Kerberos require both peers to use a session key to create new messages. On the contrary, let us consider a protocol that delivers a session key to a peer without requiring the peer to use it. Since the approach based on trace inspection expresses knowledge of a message (and its components) via the ability to create the message, it could certainly prove no significant property in this case. By contrast, the approach based on message reception could express the peer's knowledge of the session key upon its reception, thus allowing investigation of key distribution and non-injective agreement. If the delivery of the session key is not the last step of the protocol, then the goal of key distribution might be met (see §7.5.1) depending on the protocol design. Otherwise, according to our definition (§4.7), the goal is certainly not achieved because none of the events occurring during the protocol execution implies that the agent has received the message that contains the session key. Therefore, the fact that the agent knows the key cannot become known to the agent's peer.

7.5.1 On Otway-Rees

The Otway-Rees protocol [31, p.244] offers another term of comparison.

The responder B obtains the session key from the server and forwards it to the initiator A in the last message of the protocol (figure 7.3).

$$\begin{array}{llcll}
1. & A & \to & B & : \; M, A, B, \{\!|Na, M, A, B|\!\}_{Ka} \\
2. & B & \to & \mathsf{S} & : \; M, A, B, \{\!|Na, M, A, B, |\!\}_{Ka}, \{\!|Nb, M, A, B|\!\}_{Kb} \\
3. & \mathsf{S} & \to & B & : \; M, \{\!|Na, Kab|\!\}_{Ka}, \{\!|Nb, Kab|\!\}_{Kb} \\
4. & B & \to & A & : \; M, \{\!|Na, Kab|\!\}_{Ka}
\end{array}$$

Figure 7.3: The Otway-Rees protocol

Key distribution is achieved when A receives a session key and gets evidence that her peer B knows it, and B gets the same evidence about A (§4.7). According to this definition, Otway-Rees does not meet the goal of key distribution because it delivers the session key to A in the last message. The delivery of the key to B is not the last event occurring during the protocol, so we may wonder whether the protocol enforces *half key distribution to B*: upon reception of the session key, does A get evidence that B knows it?

The approach to agents' knowledge based on trace inspection cannot help to answer the question because B is simply forwarding a certificate, containing the session key, that is already in the traffic. So, B certainly does not issue the certificate. The analysis of the protocol by the BAN logic came to the following conclusion: "it is interesting to note that this protocol *does not make use of Kab as an encryption key*, so neither principal can know whether the key is known to the other"[1] [31, p.247]. We refute the claim, showing that there exists a protocol similar to Otway-Rees that does not use the session key as an encryption key but informs one agent that his peer does know the session key. The BAN logic, in fact, fails to capture knowledge of the messages that are received, as does our approach to agents' knowledge that is based on trace inspection. Our formal account for message reception lets us discover that the reason why Otway-Rees fails to guarantee to A that B shares the session key is in fact lack of message integrity.

Let us consider a protocol that differs from Otway-Rees only in the last two messages, which are shown in figure 7.4. The new protocol is not much dissimilar from the original Otway-Rees. For example, the third message still informs B of M, Nb and Kab but not of Na; the fourth merely forwards

[1]More precisely, the BAN logic cannot infer the usual formula $A \models B \models A \overset{Kab}{\longleftrightarrow} B$.

$$\begin{array}{llcll} 3. & \mathsf{S} & \rightarrow & B & : \{\!|A, M, \{\!|B, M, Na, Kab|\!\}_{Ka}, Nb, Kab|\!\}_{Kb} \\ 4. & B & \rightarrow & A & : \{\!|B, M, Na, Kab|\!\}_{Ka} \end{array}$$

Figure 7.4: A protocol based on Otway-Rees (fragment)

the certificate to A. The main difference is that the new third message enjoys integrity because it is a cipher (the old one does not because it is compound). Also the new fourth message is integral but this is irrelevant to the following reasoning. In particular, the message could be any compound one quoting the certificate for A, e.g. $\{\!|M, \{\!|B, M, Na, Kab|\!\}_{Ka}|\!\}$.

The integrity of the third message is crucial because it allows to keep track of B's activity, if B is not compromised. When the server sends the message on a trace *evs*, the certificate that it contains is not yet visible to the spy thanks to the message integrity. Precisely, it does not belong to $\mathsf{analz}(\mathsf{knows\,Spy}\ \ evs)$. On the contrary, the certificate becomes visible to the spy after B has acted. This is the leading philosophy of theorem 7.23. Let $\mathsf{or\_integral}$ be the formal model for the protocol.

Theorem 7.23. *If A and B are not compromised and evs, belonging to* $\mathsf{or\_integral}$*, is such that*

$$\begin{array}{l}\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,M, \mathsf{Nonce}\,Na, \mathsf{Key}\,Kab|\!\} \\ \quad \in \mathsf{analz}(\mathsf{knows\,Spy}\ \ evs)\end{array}$$

then evs contains

$$\mathsf{Says}\,B\,A\,(\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,M, \mathsf{Nonce}\,Na, \mathsf{Key}\,Kab|\!\})$$

The result cannot be proved on the original Otway-Rees where the assumption holds also before B acted. Incidentally, the result does not hold replacing analz by parts because the certificate appears as component of the traffic even before B acted during the new protocol. The proof is not straightforward as it requires a new rewriting rule for the analz operator. In particular, the subgoal arising from the formalisation of the fourth step of the protocol requires evaluating the assumption

$$\begin{array}{l}\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,M, \mathsf{Nonce}\,Na, \mathsf{Key}\,Kab|\!\} \\ \quad \in \mathsf{analz}(\{\mathsf{Key}\,K\} \cup (\mathsf{knows\,Spy}\ evs4))\end{array}$$

The symbolic evaluation is not trivial. We can prove a rewriting rule to inform the simplifier that no session key is used to encrypt a cipher in the protocol (lemma 7.24).

Lemma 7.24. *If K' is a session key, evs belongs to* $\mathsf{or\_integral}$*, and K is confidential on evs, then*

$$\begin{aligned}(\mathsf{Crypt}\, K X \in \mathsf{analz}(\{\mathsf{Key}\, K'\} \cup (\mathsf{knows}\,\mathsf{Spy}\ \mathit{evs})))&\\ \iff (\mathsf{Crypt}\, K X \in \mathsf{analz}(\mathsf{knows}\,\mathsf{Spy}\ \mathit{evs}))&\end{aligned}$$

The proof can be developed through the conventional strategy for the session key compromise theorem (§4.5). However, it requires several subsidiary results, such as the authenticity of the messages issued by the server, which did not hold of the original Otway-Rees.

The conclusive portion of reasoning asserts that an agent who is not compromised sends the last message of the protocol only upon reception of an integral, suitable instance of the last but one (theorem 7.25).

Theorem 7.25. *If B is not compromised, and evs, belonging to* $\mathsf{or\_integral}$*, contains*

$$\mathsf{Says}\, B\, A\, (\mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|\mathsf{Agent}\, B, \mathsf{Nonce}\, M, \mathsf{Nonce}\, Na, \mathsf{Key}\, Kab|\!\})$$

then, for some Nb, evs contains

$$\begin{aligned}\mathsf{Gets}\, B\, (\mathsf{Crypt}(\mathsf{shrK}\, B)\{\!|\mathsf{Agent}\, A, \mathsf{Nonce}\, M,&\\ \mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|\mathsf{Agent}\, B, \mathsf{Nonce}\, M, \mathsf{Nonce}\, Na, \mathsf{Key}\, Kab|\!\},&\\ \mathsf{Nonce}\, Nb, \mathsf{Key}\, Kab|\!\}&\end{aligned}$$

If B was merely a friendly agent, we could still prove that he sends the last message upon reception of a suitable one, as seen on similar steps of BAN Kerberos or Kerberos IV. However, B extracts the message that he sends from a larger message sealed under his own shared key. Therefore, B must not be compromised so that this message is integral, which ensures that it contains two occurrences of the same session key.

We now have all the fragments of A's reasoning. Upon reception of the last message of the protocol, A concludes that the certificate is available to the spy by lemma 7.5 and $H \subseteq \mathsf{analz}\, H$ (and decomposition under analz if the message was compound). Then, from theorem 7.23, A could derive that B acted. Therefore, B received an intelligible message quoting the same session key received by A, as stated by theorem 7.25, and could extract the key. In conclusion, the protocol informs A that the session key she receives is also known to her peer B (theorem 7.26).

Theorem 7.26. *If A and B are not compromised and evs, belonging to* $\mathsf{or\_integral}$*, contains*

$$\mathsf{Gets}\,A\,(\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Agent}\,B, \mathsf{Nonce}\,M, \mathsf{Nonce}\,Na, \mathsf{Key}\,Kab|\!\})$$

then

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

We remark that the new protocol only differs from the original Otway-Rees in the integrity of the message issued by the server. Nevertheless, as Otway-Rees, the protocol *does not make use of Kab as an encryption key* but does inform A that B knows the session key. The new protocol meets the goal of half key distribution to B.

7.5.2 On Public-Key Protocols

Testing our approaches to agents' knowledge on the public-key Needham-Schroeder protocol has unveiled a limitation of the one based on message reception. For example, let us consider the first step of the protocol and suppose that it takes place during the history modelled by the trace *evs*. Then, *evs* contains the event

$$\mathsf{Says}\,A\,B\,(\mathsf{Crypt}(\mathsf{pubK}\,B)\{\!|\mathsf{Agent}\,A, \mathsf{Nonce}\,Na|\!\})$$

whereby agent A issues a nonce Na and sends it to B inside a cipher sealed by B's public key. Since A does not know B's private key, which is necessary to decrypt the cipher, we cannot establish that Na belongs to $\mathsf{analz}(\mathsf{knows}\,A\ evs)$. However, A in fact knows Na because she has just created it. Trace inspection regains its attraction in this case because it could prove that A creates the entire cipher and therefore knows its components. Nonetheless, also B will know Na upon reception of the cipher, but this again requires reasoning with message reception. Therefore, a combination of the two approaches, which can be easily implemented, will yield the best results when analysing public-key protocols.

7.6 Timestamps vs. Nonces on the same Design

Our experiments support the claim that, on the same protocol design, timestamps add stronger temporal requisites than nonces to the goals of a protocol [18].

Since timestamps are a linear order, any agent may check the freshness of a message containing a timestamp at any point, even without having yet participated in the protocol. The agents only need to know the specific lifetime for the timestamp, but lifetimes are in general not secret. However, all agents must run a synchronisation protocol to synchronise their clocks. It could be argued that the threats to this protocol might offset any gains obtained from using timestamps in the cryptographic protocol, but this lies outside the scope of our research. By contrast, if the protocol is based on nonces, an agent wanting to check the freshness of a message should have participated earlier in the protocol and issued a nonce. Then, some other agent should have inserted the same nonce into the message.

BAN Kerberos may be viewed as the shared-key Needham-Schroeder "modified with the addition of timestamps" [77]. Thus, we find it significant to compare the temporal requisites of the goals of authenticity and authentication achieved by the two protocols (the other goals are independent from time).

Recall the protocols from figures 5.1, §5.2 and 2.6, §2.2.4. Since the model servers can be proved to function reliably (e.g. §5.4.1), they create the components they send. Therefore, also the approach to agents' knowledge based on message reception suffices to express the following reasoning (whose formal statements are omitted here). The authenticity argument for BAN Kerberos (theorem 5.3, §5.4.3) assures a non-compromised protocol initiator A that the message

$$\{\!|\, Tk, B, Kab, Ticket |\!\}_{Ka}$$

originated with the server. Since the server is reliable, the guarantee informs A that the message and the session key it contains were created at time Tk. In the same setting, Paulson proves that the message

$$\{\!| Na, B, Kab, Ticket |\!\}_{Ka}$$

of the Needham-Schroeder protocol originated with the server. Nonce Na being A's nonce, A is assured that the message is more recent than the time when she issued Na. This requisite also applies to the session key Kab. Though not identical, the two guarantees may be considered practically equivalent.

A non-compromised protocol responder B can be assured that the ticket

$$\{\!|\, Tk, A, Kab |\!\}_{Kb}$$

of the BAN Kerberos protocol originated with the server. By the presence of the timestamp, B derives that the ticket and the session key were created at time Tk. No such guarantee is available in the same setting to B when running Needham-Schroeder, because the ticket has the form

$$\{\!| Kab, A |\!\}_{Kb}$$

and so B obtains no information on how recent Kab is.

Concerning the goal of authentication, theorem 7.11 also assures B that A was alive at time Ta. The analogous guarantee (theorem 7.27) we have proved on Needham-Schroeder conveys a similar requisite. In fact, B is informed that A was alive after B created his nonce Nb. (Note that ns_shared denotes the formal protocol model, and that the double concatenation of Nb formalises the last message of the protocol).

Theorem 7.27. *If A is not the spy, B is not compromised, and evs, belonging to* ns_shared*, is such that*

$$\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|\mathsf{Key}\,Kab, \mathsf{Agent}\,A|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs) \quad \textit{and}$$
$$\mathsf{Crypt}\,Kab\{\!|\mathsf{Nonce}\,Nb, \mathsf{Nonce}\,Nb|\!\} \in \mathsf{parts}(\mathsf{spies}\ evs)$$

and Kab is confidential on evs, then

$$A\ \mathsf{Issues}\ B\ \mathsf{with}\ (\mathsf{Crypt}\,Kab\{\!|\mathsf{Nonce}\,Nb, \mathsf{Nonce}\,Nb|\!\})\ \mathsf{on}\ evs$$

Theorem 7.12 also assures A that B was alive after time Ta (namely, after A issued Ta — since B does not update Ta, it can be viewed as a nonce in the last message). The analogous guarantee (theorem 7.28) we have proved on Needham-Schroeder shows that the authentication goal for A enjoys no similar requisite.

Theorem 7.28. *If A is not compromised, B is not the spy, and evs, belonging to* ns_shared*, is such that*

$$\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Nonce}\,Na, \mathsf{Agent}\,B, \mathsf{Key}\,Kab, Ticket|\!\}$$
$$\in \mathsf{parts}(\mathsf{spies}\ evs) \quad \textit{and}$$
$$\mathsf{Crypt}\,Kab(\mathsf{Nonce}\,Nb) \in \mathsf{parts}(\mathsf{spies}\ evs)$$

and Kab is confidential on evs, then

$$B\ \mathsf{Issues}\ A\ \mathsf{with}\ (\mathsf{Crypt}\,Kab(\mathsf{Nonce}\,Nb))\ \mathsf{on}\ evs$$

In fact, A is simply informed that B creates a message containing the nonce Nb. Since Nb was issued by B, A cannot deduce any temporal requisite about B's presence.

Note that only the first of our approaches to agents' knowledge can be used to conduct the reasoning presented above. However, despite the conclusions obtained on the same design when using either timestamps or nonces, there exist protocols that make a more scrupulous and complicated use of nonces. For example, the authentication goals achieved for both peers by the Yahalom protocol, which is based on nonces, do enjoy significant temporal requisites [90].

Chapter 8

Modelling Smart Cards

The Inductive Approach is tailored to the analysis of protocols that make use of smart cards. The spy is given the chance of cloning other agents' cards and exploiting their computational resources.

The protection of long-term secrets was the main reason for the introduction of smart cards. Although several researchers believe that no smart card can be completely tamper-resistant, modern cards offer a high level of physical security. A cheap *integrated-circuit memory card* may store a few kilobytes and provide a strong shell for important information. Additionally, an *integrated-circuit microprocessor card* embeds an 8-bit microprocessor that can perform relatively simple operations such as DES encryption/decryption. Consequently, existing security protocols have been extended with smart cards (e.g. [51]) and new ones have been designed for the purpose (e.g. [98]). We refer to the protocols that are based on smart cards as *smart card protocols*, as opposed to the *traditional protocols*, which are not.

Recently, the microprocessor cards' operating systems allow the execution of user-chosen Java programs, cutting 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 [32, 41]. Smart card readers will become inexpensive pieces of hardware for home computers running smart card middleware.

Smart cards should strengthen the goals of the protocols that use them, and there exists an increasing demand for formal guarantees that this target is reached. However, to our knowledge, there are only two attempts of establishing such guarantees. Abadi et al. [1] pioneer the use of smart cards to establish mutual authentication between agents and workstations. Their treatment develops around the different functionalities of the smart cards employed, while facing the limited technology of the time, and is based on a belief logic (§2.1.1) that is a simple extension of the BAN logic. The calculus of the logic is used to prove the mutual authentication and delegation goals of three protocols which impose different requirements on the cards. Confidentiality issues are not considered, as belief logics are notoriously inadequate for this purpose. Shoup and Rubin [98] design a protocol based on smart cards and analyse it by provable security (§2.1.3). However, their treatment is very abstract, as we show in the next chapter.

Motivated by the insufficient research done in the field, we extend the Inductive Approach towards the verification of smart card protocols [13]. The model cards, which are associated to a new type of the formal language, can interact with their respective owners by receiving and sending messages. Each card stores a basic set of long-term secrets, which may depend on the specific protocol. For example, while a card for key distribution protocols has only to store specific keys, a card for e-commerce protocols may have to store a number representing the owner's balance. 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. Furthermore, she has cloned another set of cards, discovering their internal secrets. So, since the spy can act illegally, there is a set of cards that she can use even if they do not belong to her, while all other agents can only use their own card.

Several smart card protocols make the *assumption of secure means*, signifying that the spy cannot interpose between agents and their cards. So, messages can be exchanged in clear and each agent's knowledge of long-term secrets reduces to nothing. We account for both this and the opposite alternative by simple variations to the definition of spy's knowledge. Often in the following text, *secure means* will abbreviate that the assumption of secure means does hold; *insecure means* will abbreviate that the assumption does not hold.

This chapter begins with a formalisation of smart cards within the Inductive Approach (§8.1). Then, it presents the extensions necessary to the datatype of events (§8.2) and to the definition of agents' knowledge (§8.3). The spy's illegal operations now exploit certain smart cards that she does not legally own (§8.4), while the protocol model may require some extensions to account for this (§8.5). This extended approach will be demonstrated in the next chapter.

8.1 Smart cards

We aim at representing the operational aspects of smart cards, so we introduce a new free type card with several associated functions. There exists a bijective correspondence between agents and smart cards

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

In the real world, the cards' CPUs only provide certain, limited resources: a card will produce a specific output only if fed with the correct input. For example, if a card can compute a session key K from an input X, the card must necessarily be fed with X in order to obtain K. The formal protocol model can easily account for this. It will only allow for the outputs encompassed by the protocol under the condition that the cards are fed with the corresponding, specific inputs. It will not construct other outputs, even from cloned cards (see next section). As a consequence, there exists no card whose use can give the spy unlimited power.

8.1.1 Card Vulnerabilities

The model cards suffer from a number of realistic vulnerabilities due to theft, cloning and internal failures.

Theft

The small dimensions of the smart cards confer their portability but also raise the risk of loss or theft. In the worst case, all smart cards that have been lost by their owners or stolen to 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

The spy is not necessarily able to use a stolen card actively, unless she knows its pin. Nevertheless, she could be able to use modern techniques (such as *microprobing* [8]), break the physical security of the card, access its EEPROM[1] where the long-term secrets are stored and, in the worst case, reverse engineer the whole card chip. At this stage, the spy would be able to build a clone of the card for her own use. If this process succeeds, the card belongs to the set cloned in the model, and $\mathsf{cloned} \subseteq \mathsf{card}$.

[1]Electrically-Erasable Programmable Read-Only Memory.

Cloning without apparent theft

All cloning techniques that are currently known are *invasive*, in the sense that they spoil the original card. The chip of the card 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 not suspect anything. Alternatively, the spy might even be able to tailor *non-invasive* techniques (such as *fault generation* [68, 59] by exploiting the power and clock supply lines) to cloning in the near future, and return the original card to its owner after building a clone for herself. Modelling these opportunities simply requires stating no relation between the sets stolen and cloned, so that a card could be cloned and not be stolen. Such a card could be used both by its legal owner and by the spy, granting them identical computational resources.

Data bus failure

All cards' data buses are corrupted so that the travelling messages can be either forgotten (due to electronic decay), permuted or fed to the CPU repeatedly (due to simple layout modifications).[2] Therefore, a smart card can omit some computations or repeat others. In the worst case, the spy has caused all cards to deteriorate in this fashion before they are delivered to their respective owners, leaving no visible trace of tampering. To model this, events only occur by firing of inductive rules in the formal protocol model, but rules are not forced to fire even when their preconditions are met. Also, rules may fire in any order or fire more than once, and each of these possibilities is recorded by a trace.

Global internal failure

Smart cards may suffer unexpected failures and stop working at some point. The formal protocol model reflects this scenario by including traces that do not involve those cards after some event.

8.1.2 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 by

[2] Allowing message alteration or leakage at this level would give the spy excessive power: many protocols explicitly rely on a secure means between agents and smart cards.

its owner, namely it can be used legally. The spy cannot handle a non-stolen card unless it is her own.

Definition 8.1. $\mathsf{legallyU}(\mathsf{Card}\,A) \equiv (\mathsf{Card}\,A) \notin \mathsf{stolen}$.

When the assumption of secure means does not hold, the spy can listen in between any agent and his smart card, so she has electronic access to those cards of which she knows the pins.[3] Pins are sent between agents and cards (never between agents), so the spy might learn some of them on certain traces. On the contrary, should the cards not be pin-operated, they would all be illegally usable.

Definition 8.2. Let the assumption of secure means not hold;

$$\mathsf{illegallyU}(\mathsf{Card}\,A)\ \mathsf{on}\ evs \equiv \begin{cases} \textit{the spy knows A's pin on evs} \\ \qquad\qquad \text{if cards are pin-operated} \\ \mathsf{true} \\ \qquad \text{if cards are not pin-operated} \end{cases}$$

The informal predicate *the spy knows A's pin on evs* will be refined by the formal definition of agents' knowledge (see §8.3).

In case the assumption of secure means does hold, the spy needs to gain physical access to the cards in addition to the knowledge of their pins. Since she cannot monitor the events involving the smart cards, she has no chance of discovering any pins via any events. She can only know them initially (see §8.3), so the definition of illegal usability does not depend on the trace. If the cards are not pin-operated, we only need characterise the physical access to the card.

Definition 8.3. Let the assumption of secure means hold,

$$\mathsf{illegallyU}(\mathsf{Card}\,A) \equiv \begin{cases} (\mathsf{Card}\,A) \in \mathsf{cloned} \ \vee \ ((\mathsf{Card}\,A) \in \mathsf{stolen} \ \wedge \\ \qquad\qquad\qquad\qquad \textit{the spy knows A's pin}) \\ \qquad\qquad\qquad\qquad \text{if cards are pin-operated} \\ (\mathsf{Card}\,A) \in \mathsf{cloned} \ \vee \ (\mathsf{Card}\,A) \in \mathsf{stolen} \\ \qquad\qquad\qquad \text{if cards are not pin-operated} \end{cases}$$

The informal predicate *the spy knows A's pin* will be refined below.

The spy must be able to use her own card legally because she must be given the opportunity to act legally. However, she does not need to use her

[3]If a smart card is pin-operated, then it accepts no communication unless it is activated by means of its pin.

card illegally because she cannot acquire additional knowledge from it. So, we assume

$$\mathsf{Card\ Spy} \notin \mathsf{stolen} \cup \mathsf{cloned}$$

The same assumption is made on the card that belongs to the server.

We stress 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.

Every agent is able to verify whether his own card is stolen by checking that he holds it. All other assumptions about the agent's card or the agent's peer's card now belong to the minimal trust (§4.8).

8.1.3 Card Secrets

A smart card typically contains two long-term symmetric keys: the pin to activate its functionality and the card key.

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

The first of the two functions could be equivalently defined on cards rather than on agents. The card key serves to limit the data that must be stored in the card RAM. Suppose that when the card is required to issue a fresh nonce it also outputs the nonce encrypted under its key. This cipher may be used later to assess the nonce authenticity to the card, even if the card did not store the nonce, assuming that the card key is secure.

In case of key distribution protocols, each card also stores its owner's long-term key, which is not known to the agent in contrast with traditional protocols. We keep the original definition [88, §3.5]

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

Note that, since the model is operational, the notion of the smart cards' *storing* some secrets need not be formalised explicitly. We only need to define how these secrets are used, namely in which circumstances and to whom they will become known. This increases the flexibility of the approach. If the smart cards store additional secrets in certain applications, once such secrets are formalised by suitable functions, only the definition of agents' knowledge must be updated.

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

8.2 Events

Let us suppose that the assumption of secure means does not hold. If a message is sent by a card to its owner (or vice versa), its reception is not guaranteed. So, both events must be formalised. We upgrade the Isabelle datatype for events as follows.

```
datatype event = Says     agent  agent  msg
               | Notes    agent         msg
               | Gets     agent         msg
               | Inputs   agent  card   msg
               | Gets_c   card          msg
               | Outputs  card   agent  msg
               | Gets_a   agent         msg
```

The known *network events* (sending, noting and receiving a message, §7.2) have been extended with the new *card events*. Agents may send inputs to the cards (Inputs) and the cards may receive them (Gets_c); similarly, the cards may send outputs to the agents (Outputs) and the agents may receive them (Gets_a). 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 messages could have been forged by the spy.

Extending the reception invariant (§7.2), the protocol model only allows the cards to receive by a Gets_c event the messages that have been sent by an Inputs event, and the agents to receive by a Gets_a event the messages that have been sent by an Outputs event (see §8.5).

If the assumption of secure means does hold, then the events Gets_c and Gets_a can be omitted. As a matter of fact, a card certainly receives its owner's inputs, and an agent certainly receives his card outputs. Consequently, a smart card C can verify whether an event Inputs $A\,C\,X$ occurred and an agent A can verify whether an event Outputs $C\,A\,X$ occurred, while this is impossible on insecure means.

The formal definition of the function used (§3.3, §7.2) must be extended to cope with the new events

– $\mathsf{used}((\mathsf{Inputs}\,A\,C\,X)\ \#\ evs) \triangleq \mathsf{parts}\{X\} \cup \mathsf{used}\ evs$

– $\mathsf{used}((\mathsf{Gets\_c}\,A\,X)\ \#\ evs) \triangleq \mathsf{used}\ evs$

– $\mathsf{used}((\mathsf{Outputs}\,C\,A\,X)\ \#\ evs) \triangleq \mathsf{parts}\{X\} \cup \mathsf{used}\ evs$

– $\mathsf{used}((\mathsf{Gets\_a}\,A\,X)\ \#\ evs) \triangleq \mathsf{used}\ evs$

The cases concerning Gets_c and Gets_a do not extend the set of used components because the corresponding events only take place on messages that have already been accounted for by the reception invariant. If the assumption of secure means holds, both cases are omitted.

8.3 Agents' Knowledge

The function $\mathsf{initState}$ formalising agents' initial knowledge [88, §3.5] must be redefined to account for the secrets stored in smart cards. We quote here a fairly general definition for smart card protocols with pin-operated cards. The original datatype of agents is employed (§3.1.1).

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

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

Friendly agents' initial knowledge consists of their respective pins.

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

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).

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

Note that this definition is not influenced by the assumption of secure means because it formalises the situation before any protocol sessions have taken place.

The knowledge that agents can extract from traces, defined by the function knows (§7.2.1), must be extended to account for the card events.

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

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

5. No agent, including the spy, can extend his knowledge with any of the messages received by any smart card on a trace. The spy and the message originators already know them by case 4 thanks to the

reception invariant.

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

6. An agent knows no cards' outputs on a 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, other than the spy, knows what he receives from his card on a trace. The spy knows all messages received by any smart card by case 6, due to the reception invariant.

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

At this stage, definition 8.2 can be refined. Recall that the function analz extracts all message components from a set of messages using keys that are recursively available [88, §3.2].

Definition 8.2′. Let the assumption of secure means not hold;

$$\mathsf{illegallyU}(\mathsf{Card}\,A)\;\mathsf{on}\;\mathit{evs} \equiv \begin{cases} \mathsf{Key}\,(\mathsf{pin}\,A) \in \mathsf{analz}(\mathsf{knows}\,\mathsf{Spy}\;\mathit{evs}) \\ \qquad\qquad\text{if cards are pin-operated} \\ \mathsf{true} \\ \qquad\qquad\text{if cards are not pin-operated} \end{cases}$$

In particular, a cloned card or a card whose owner is compromised is illegally usable on any trace (by definition of initState, base case of knows, definition of analz). As expected, the illegal usability of a card over insecure means does not necessarily imply the spy's physical access to the card.

If the assumption of secure means does hold, the definition of knows simplifies (the implementation may be found in Appendix C). 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 listen in. Hence, cases (4) and (6) must be amended accordingly.

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

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

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

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

As it was desired, 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. By definition of initState and base case of knows, definition 8.3 can be refined.

Definition 8.3′. Let the assumption of secure means hold;

$$\mathsf{illegallyU}(\mathsf{Card}\, A) \equiv \begin{cases} (\mathsf{Card}\, A) \in \mathsf{cloned} \;\vee\; ((\mathsf{Card}\, A) \in \mathsf{stolen} \;\wedge\; A \in \mathsf{bad}) \\ \hfill \text{if cards are pin-operated} \\ (\mathsf{Card}\, A) \in \mathsf{cloned} \;\vee\; (\mathsf{Card}\, A) \in \mathsf{stolen} \\ \hfill \text{if cards are not pin-operated} \end{cases}$$

This definition insists on the spy's physical access to the illegally usable cards over secure means. Only when the cards are not pin-operated 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 could be extended to smart cards, but reasoning about cards' knowledge may be of little significance due to their limited RAM.

8.4 Spy's Illegal Behaviour

As demonstrated in chapters 5 and 6, the spy's illegal behaviour with traditional protocols is typically specified by a single inductive rule that extends the protocol model.[4]

[4]Only the model of the TLS protocol [89] contains a second rule, which allows the spy to construct a session key because the algorithm is public.

In modelling smart card protocols, the spy must be allowed to exploit the illegally usable smart cards. If the assumption of secure means does not hold, 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 different reactions. The *Fake* rule must be amended as outlined in figure 8.1. Note the condition of illegal usability over insecure means stated on A's card.

Fake
```
[| evsF ∈ smart_p_insecure_m; illegallyU(Card A) on evs;
   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 8.1: Spy's illegal operation in case of insecure means

If the assumption of secure means holds, then the spy cannot send fake card outputs to the agents. Figure 8.2 presents the corresponding, new *Fake* rule. Note the condition of illegal usability over secure means stated on A's card.

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 8.2: Spy's illegal operation in case of 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 insecure means, the illegally usable cards lie in the spy's hands. Suppose that A's card outputs X' when it is fed X. The formal protocol model will contain rule *Name* (figure 8.3), which requires the card to be legally usable. Hence, rule *Name_Fake* must be added to allow A's card to output X' towards the spy in case the card is illegally usable and was input X by the spy. The spy learns X' from the firing of the latter rule, not the former. Any extra assumptions in *Name* must be kept

```
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 8.3: Pair of rules for each output in case of secure means

in *Name_Fake*. Should A's card be both legally and illegally usable, both rules would be enabled to fire. Rule *Name_Fake* is unnecessary over insecure means, where the spy monitors all card events.

8.5 Formal Protocol Model

The formal model for a smart card protocol requires additional features if the assumption of secure means does not hold. 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, we introduce rules *Reception_c* and *Reception_a* respectively (figure 8.4), which are inspired to the *Reception* rule that allows reception of messages sent over the network (§7.2.2). Since the rules are not forced to fire, no kind of reception (either on the network or on the agent-smart-card means) is guaranteed, as it is the case in a world where the spy controls all means.

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

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

Figure 8.4: Rules for reception in case of insecure means

If the assumption of secure means does hold, then reception over the agent-smart-card means is guaranteed, so the rules of figure 8.4 are not needed.

Chapter 9

Verifying a Smart Card Protocol

The "provably secure" Shoup-Rubin protocol, which is based on smart cards, is mechanised. Two weaknesses due to lack of explicitness are unveiled, which affect the goals of confidentiality, authentication and key distribution.

Shoup and Rubin [98] take into account an existing session key distribution protocol due to Leighton and Micali [60] and prove it secure using the Bellare and Rogaway's framework [24]. Then, they develop a new protocol, based on the design by Leighton and Micali, for session key distribution in a three-agent setting where each agent is endowed with a smart card that can compute a few PRFs. Finally, they extend Bellare and Rogaway's approach to account for smart cards, and argue that the new protocol enjoys the two following properties. First, a pair of agents running the protocol share the same session key at the end of a protocol session in which the spy does not prevent the delivery of the relevant messages. There is no proof for this property, although this may not be obvious to readers unfamiliar with the formalism. Second, the adversary is shown by mathematical proof to have a *negligible advantage*, signifying that the session key remains confidential. The reasoning is performed without any kind of mechanised support.

We have applied the extended Inductive Approach described in the previous chapter to the Shoup-Rubin protocol and verified its goals of authenticity, unicity, confidentiality, authentication and key distribution [14]. We argue that the confidentiality stated of the protocol in terms of provable security is highly theoretical. We have discovered that the confidentiality theorems that hold of the protocol model cannot be applied by the peers, so the protocol lacks goal availability (§4.8). This is due to the lack of explicitness of two crucial protocol steps. Inspecting the corresponding proofs suggests a simple fix, which can be verified to be effective. To our knowledge, this work represents the first mechanisation of a full protocol based on smart cards.

This chapter presents the Shoup-Rubin protocol (§9.1), its modelling (§9.2) and its verification (§9.3). The verification is extended on an upgraded version of the protocol (§9.4) that achieves stronger goals.

9.1 The Shoup-Rubin Protocol

We present an abstract version of the protocol obtained both from 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 P's smart card long-term key by K_{Cp}.

The protocol relies on the concept of *pairkey* (due to Leighton and Micali [60]) to establish a long-term secret between the smart cards of a pair of agents. 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.

The full protocol (figure 9.1) 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 its certificate encrypted under her long-term key.

Phase II. A queries her card and receives a fresh nonce and its certificate encrypted under the card long-term key. 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}.

$$
\begin{array}{rlllll}
\mathrm{I}: & 1. & A & \rightarrow & \mathsf{S} & : A, B \\
 & 2. & \mathsf{S} & \rightarrow & A & : \mathit{\Pi}_{ab}, \{\!|\mathit{\Pi}_{ab}, B|\!\}_{Ka} \\[1em]
\mathrm{II}: & 3. & A & \rightarrow & C_a & : A \\
 & 4. & C_a & \rightarrow & A & : Na, \{\!|Na|\!\}_{KC_a} \\[1em]
\mathrm{III}: & 5. & A & \rightarrow & B & : A, Na \\[1em]
\mathrm{IV}: & 6. & B & \rightarrow & C_b & : A, Na \\
 & 7. & C_b & \rightarrow & B & : Nb, Kab, \{\!|Na, Nb|\!\}_{\pi_{ab}}, \{\!|Nb|\!\}_{\pi_{ab}} \\[1em]
\mathrm{V}: & 8. & B & \rightarrow & A & : Nb, \{\!|Na, Nb|\!\}_{\pi_{ab}} \\[1em]
\mathrm{VI}: & 9. & A & \rightarrow & C_a & : B, Na, Nb, \mathit{\Pi}_{ab}, \\
 & & & & & \{\!|\mathit{\Pi}_{ab}, B|\!\}_{Ka}, \{\!|Na, Nb|\!\}_{\pi_{ab}}, \{\!|Na|\!\}_{KCa} \\
 & 10. & C_a & \rightarrow & A & : Kab, \{\!|Nb|\!\}_{\pi_{ab}} \\[1em]
\mathrm{VII}: & 11. & A & \rightarrow & B & : \{\!|Nb|\!\}_{\pi_{ab}}
\end{array}
$$

Figure 9.1: The Shoup-Rubin protocol

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 its certificate, the two certificates for the nonces; A's card computes π_{ab} from $\mathit{\Pi}_{ab}$ 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 makes the assumption of secure means, so that the spy cannot listen in between agents and their respective cards. The cards indeed output the session keys in clear. Although this feature may seem unrealistic to use on a vast scale, in general it adds robustness to a protocol by reducing each agent's knowledge to the pin to activate his card. The current version of Shoup-Rubin in fact employs smart cards that are not pin-operated,[1] so no agent knows any long-term secrets.

[1]Although this is explicitly stated neither in the designers nor the implementors' pa-

Nevertheless, other features may seem incautious. The protocol reveals A's nonce to the spy in step 5, and B's in step 8. An informal account for the consequences can be hardly given. We formally verify that, even if the session key is computed out of B's nonce, the knowledge of this nonce does not help the spy to discover the session key as long as she cannot use A and B's cards.

9.2 Modelling Shoup-Rubin

The protocol never uses a pairkey as a cryptographic key but merely as a means to establish the corresponding pair-k. Moreover, a pairkey remains secret as long as the spy does not observe or compute it. Therefore, our model treats pairkeys as nonces.

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

On the contrary, a pair-k is used as a proper cryptographic key. The session key is constructed from a nonce and a pair-k.

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

At the operational level, we do not need to explore the implementation details beyond 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, they are respectively disjoint from the ranges of the functions formalising other long-term keys (§8.1.3), so that any pair-k differs from a card key and so forth.

The definition of initState must be updated (the implementation may be found in Appendix C). The protocol relies on cards that are not pin-operated, so all occurrences of the function pin may be omitted. The server's initial knowledge must also comprise all pairkeys and all pair-k's.

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

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

pers, Peter Honeyman — one of the implementors — kindly clarified it during a private conversation.

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

Recall the definition of pairkey and pair-k from the previous section. The spy's initial knowledge must be extended on 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.

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

The formalisations of smart cards, events and spy are inherited from the general treatment presented in the previous chapter. However, the server never uses its smart card in this protocol.

We declare the constant shouprubin as a set of lists of events. It designates the formal protocol model and is defined in the rest of the section by means of inductive rules. Since the protocol assumes secure means and the cards are not pin-operated, definition 8.3′ of illegal usability (§8.3) applies.

9.2.1 Basics

The basic rules of a formal protocol model are presented in figure 9.2. 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

Base
```
[] ∈ shouprubin
```

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

Figure 9.2: Modelling Shoup-Rubin: basics

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 respective intended recipients. Rule *Fake* is treated later (see §9.2.9).

9.2.2 Phase I

The rules modelling phase I of the protocol are presented in figure 9.3. Any agent except the server may initiate a protocol session at any time, hence the corresponding event may extend any trace of the model (*SR1*). Upon

```
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 9.3: Modelling Shoup-Rubin: phase I

reception of a message quoting two 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 clear, it does not reveal its peers. This information is carried by the certificate, which explicitly creates the association between pairkey and peers.

9.2.3 Phase II

The rules modelling phase II of the protocol are presented in figure 9.4. The initiator of a protocol session may query her own smart card provided that she received a message containing a nonce and a certificate (*SR3*).

```
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 9.4: Modelling Shoup-Rubin: phase II

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 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 integrity and authenticity of the pairkey. However, the protocol specification does not encompass this, so we are analysing the protocol with 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 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 by the card owner.

9.2.4 Phase III

The rules modelling phase III of the protocol are presented in figure 9.5. 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, §9.2.6), the responder obtains a message of the

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

Figure 9.5: Modelling Shoup-Rubin: phase III

same form with 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, 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 to state this check, which introduces ambiguity in the specification. Incidentally, recall that, when the certificate is a cipher, no agent can check its internal structure because its encryption key is only known to a smart card.

9.2.5 Phase IV

The rules modelling phase IV of the protocol are presented in figure 9.6. 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 card issues a fresh nonce, computes the pair-k for initiator and

```
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 9.6: Modelling Shoup-Rubin: phase IV

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 will make certain of obtaining it again from the network in the final phase.

9.2.6 Phase V

The rules modelling phase V of the protocol are presented in figure 9.7. When the responder obtains from his card a nonce followed by a key and two certificates, he prepares for sending the nonce and one certificate 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. Note that the three components following the nonce in the card output might be seen as a unique certificate, thus inviting the ambiguity discussed above (§9.2.4).

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 9.7: Modelling Shoup-Rubin: phase V

9.2.7 Phase VI

The rules modelling phase VI of the protocol are presented in figure 9.8. The scenario returns to 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

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;
   ALL 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 (Pairkey(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 9.8: Modelling Shoup-Rubin: phase VI

and *Cert1* as its 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*).

9.2.8 Phase VII

The rules modelling phase VII of the protocol are presented in figure 9.9. Upon reception of a cryptographic key and a certificate from her smart card, the initiator forwards the certificate to the responder.

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 9.9: Modelling Shoup-Rubin: phase VII

9.2.9 Threats

In addition to the legal behaviour described above, the spy may 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 9.10 (which is drawn from figure 8.2, §8.4).

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

Figure 9.10: 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 [89] 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 9.11, thus acquiring knowledge of it. Since the pair-k's are never sent on the network but merely used as encryption keys, they can only be known initially by definition of initState.

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 9.11: Modelling Shoup-Rubin: threats on session keys

The means between agents and smart cards being assumed secure, the model must be extended to allow the spy to obtain the outputs of the illegally usable cards. According to the template in figure 8.3 (§8.4), we introduce a further rule for each card output. Rule *SR4_Fake* in figure 9.12 is built from *SR4*, while analogous rules *SR7_Fake* (built from *SR7*) and *SR10_Fake* (built from *SR10*) are also needed but omitted here.

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 9.12: Modelling Shoup-Rubin: threats on card outputs

9.2.10 Accidents

We complete the model by allowing accidents (or breaches of security) on session keys 9.13. This is typically done by a single rule (as seen on BAN Kerberos, §5.3), or by two rules leaking two different kinds of session keys (as

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 9.13: Modelling Shoup-Rubin: accidents

seen on Kerberos IV, §6.2.5). Shoup-Rubin requires 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 could not 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.

9.3 Verifying Shoup-Rubin

The guarantees proved for a smart card protocol may, in general, also be expressed from the viewpoint of smart cards, helping optimise their hardware or software design. This section discusses those established about Shoup-Rubin. Note that 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. The minimal trust now includes that certain cards be not usable by the spy.

The reliability theorems show that the model makes the expected use of smart cards (§9.3.1) and that messages 7 and 10 crucially lack explicitness. Suitable regularity lemmas can be expressed about all three kinds of long-term keys employed by the protocol (§9.3.2). While the authenticity argument (§9.3.3) only yields a single guarantee for the card that belongs to the protocol initiator, the unicity argument (§9.3.4) will provide guarantees for both initiator and responder. Confidentiality (§9.3.5) is weakened by the mentioned lack of explicitness and so are the goals of authentication (§9.3.6) and key distribution (§9.3.7).

9.3.1 Reliability of the Shoup-Rubin Model

The model server functions reliably (theorem 9.1). However, this theorem cannot be made useful to A (unlike theorem 6.3 was made by theorem 6.6). The authenticity argument about the message $\{\!|\mathsf{Nonce}\,Pk, Cert|\!\}$ is extremely weak. Should A receive such message, she cannot be guaranteed that it is an instance of message 2, namely that the server sent it, because the message is compound; nor can she inspect the form of the certificate.

Theorem 9.1. *If evs contains*

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

then there exists B such that

$$\begin{aligned}&Pk = \mathsf{Pairkey}(A, B) \quad \textit{and}\\ &Cert = \mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Nonce}\,(\mathsf{Pairkey}(A, B)), \mathsf{Agent}\,B|\!\}\end{aligned}$$

Further guarantees concern the use of the smart cards allowed by the protocol, the outputs that they produce and the inputs that the friendly agents send them.

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 9.2). This signifies that a friendly agent can only use his own card and can only use it legally, as we required.

Theorem 9.2. *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

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

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 9.3). Since the spy's card is not illegally usable, the agent A mentioned by the theorem certainly differs from the spy.

Theorem 9.3. *If evs contains either*

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

then

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

On the Outputs of the Smart Cards

To establish that the model smart cards work reliably, two categories of guarantees can be proved on the $\mathsf{Outputs}$ events.

One category states that the cards only give the correct outputs when fed with the expected inputs, so the cards cannot grant the spy unlimited resources. The case for step 10 of the protocol is presented below (theorem 9.4), while those for steps 4 and 7 are similar and omitted here.

Theorem 9.4. *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

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

Another category of reliability theorems states that the cards' CPUs work correctly. So, given a specific output, the form of its component can be tracked down. One such guarantee can be established on an instance of message 4 (theorem 9.5). The length of the certificate must be checked because of the protocol ambiguity already encountered (§9.2.4). 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 9.5. *If evs contains*

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

and Cert is not compound, then

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

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 9.6). The existential form of the assertion tells 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." [4, §2.1]. The underlying transport protocol cannot reveal the peer's identity either. If B uses the session key with the wrong peer, the consequences should not be disastrous provided that the key remains confidential. But, this lack of explicitness does weaken the confidentiality, authentication and key distribution guarantees accomplished by the protocol, as discussed in the sequel.

Theorem 9.6. *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

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

The cards' CPUs are also reliable when producing an instance of message 10 (theorem 9.7). 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 from the message.

Theorem 9.7. *If evs contains*

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

then there exist B and Nb such that

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

The theorem also shows that step 10 binds the form of the session key to the card that creates it, and associates the session key with the certificate. Therefore, should the former be inspectable, the structure of the latter could be derived (corollary 9.8), and vice versa.

Corollary 9.8. *If evs contains*

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

then,

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

On the Inputs of the Smart Cards

The analogous categories of guarantees can be established for the Inputs events.

The friendly agents must use the legally usable smart cards in a legal manner. Therefore, the friendly 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 9.4, the card belongs to A) quoting an agent B. We can prove that A initiated a session with B, and received the components of the query either from the network or from the card, by means of suitable events (theorem 9.9).

Theorem 9.9. *If A is not the spy, and evs contains*

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

then evs also contains

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

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 9.4, A will get an output from her card only if she uses the correct components as input.

Similar theorems regard the other queries to the smart cards, respectively steps 3 and 6 of the protocol.

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 was exposed to the network risks, the certificate that was produced earlier by A's card has not, so its form can be derived (theorem 9.10) signifying that all agents use their own card correctly.

Theorem 9.10. *If evs contains*

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

then

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

Note that the guarantee also applies to the spy's use of her own card. Upon reception of a message, A's card can determine whether it is an instance of message 9 by looking at its cleartext 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, inspecting the third certificate would detect the misbehaviour. However, in our model friendly agents only act legally.

9.3.2 Regularity

The protocol sends no long-term keys over the network, so the spy could do so if and only if she knows them before the protocol begins. The spy can discover a card key and the card owner's key only from cloning the card (see definition of initState, §9.2). Using the latter key, she can compute all the pair-k's meant for the card owner.

Lemma 9.11.

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

Lemma 9.12.

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

Lemma 9.13.

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

9.3.3 Authenticity

The proof scripts of the theorems discussed in this section may be found in Appendix D. All Shoup-Rubin's certificates are sealed under long-term keys, so the agents get no authenticity guarantees about them. However, since the long-term keys are stored into the smart cards, in general the authenticity argument can be directed towards the smart cards, possibly helping to optimise their design. Shoup-Rubin inputs a smart card with encrypted certificates only in step 9. We develop the corresponding authenticity argument via some subsidiary authenticity lemmas that are not directly applicable either by agents or by cards. Incidentally, from the assumption of secure means, it follows that, if a card receives a certificate as part of an input on a trace *evs*, the definition of knows does not imply that the certificate is in the network traffic, namely in parts(knows Spy *evs*).

Along with a pairkey, the server issues a certificate that verifies it. When the certificate is in the traffic, we can prove that it originated with the server if the regularity lemma 9.11 is applicable. Therefore, given that the peer's card is not cloned, the certificate is authentic (lemma 9.14). At this stage, the form of the pairkey may be specified via theorem 9.1.

Lemma 9.14. *If A's card is not cloned, and evs is such that*

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

then evs contains

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

We can verify formally that the certificate that associates A and B's nonces is built in step 7 (lemma 9.15). Since the certificate is sealed under the corresponding pair-k, investigating its origin requires an appeal to the regularity lemma 9.13, 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 also output towards the spy.

Lemma 9.15. *If B's card is not illegally usable, and evs is such that*

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

then evs contains

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

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 9.15, except for the certificate considered and the assertion being in the scope of an existentially quantified nonce Na. The same certificate is also output by A's card in message 10. Proving this result (lemma 9.16) also requires B not to be the spy in order to solve case *SR7* (so that the corresponding event does not introduce the certificate in the traffic), and A's card not to be illegally usable to solve case *SR10_Fake*.

Lemma 9.16. *If B is not the spy, A and B's cards are not illegally usable, and evs is such that*

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

then evs contains

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

The authenticity lemmas serve to prove an authenticity theorem that is applicable by A's card (theorem 9.17). So, 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 advised by theorem 9.10. If the first certificate has the expected form, then theorem 9.9 and lemma 9.14 prove the first event of the assertion (once a message is received, its components appeared in the traffic). Similarly, if the second certificate is as expected, then theorem 9.9 and lemma 9.15 prove the second event. The third certificate does not need to be inspected thanks to its provable integrity, so theorem 9.9 alone justifies the third event of the assertion. This reasoning is mechanisable by one Isabelle command that applies theorem 9.9 only once, and then the necessary authenticity lemma.

Theorem 9.17. *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

$$\mathsf{Says\ Server}\ A\,\{\!|\mathsf{Nonce}\,Pk,\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|\mathsf{Nonce}\,Pk,\mathsf{Agent}\,B|\!\}|\!\}\quad \textit{and}$$

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

$$\mathsf{Outputs}\,(\mathsf{Card}\,A)\,A\,\{\!|\mathsf{Nonce}\,Na,\,Cert3|\!\}$$

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 in 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

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

leaves the following subgoal arising from case *Base*

```
          [| Card A ∉ cloned;
            Pairkey(A,B) = Pairkey (A',B');
      Card A' ∈ cloned; Card B' ∈ cloned |] ⟹ False
```

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 teaches us that the spy might exploit the collisions suffered by the exclusive-or operator and forge a pairkey without knowing its original components but others. The probability of this happening is influenced by the redundancy introduced by the encryption function and by the length of the ciphers.

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 9.13. Then, if a session key ever appears in the traffic, one of its peers necessarily leaked it by accident, while the trace recorded the corresponding oops event (lemma 9.18). This is in fact a counterguarantee of authenticity because it emphasises the conditions under which a session key that is in the traffic is not authentic: the spy in fact introduced it. However, it will be fundamental to assess a form of session key confidentiality.

Lemma 9.18. *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

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

Proving the authenticity lemmas requires a common strategy (simpler than Paulson's for the authenticity theorems on traditional protocols [88, §4.7]). We present below the strategy for Shoup-Rubin, which can be generalised straightforwardly to any smart card protocol.

1. Apply induction.
2. If the lemma concerns
   - a certificate sealed under a shared key, then simplify case *Fake* by lemma 9.11;
   - a certificate sealed under a card key, then simplify case *Fake* by lemma 9.12;
   - a certificate sealed under a pair-k, then simplify case *Fake* by lemma 9.13;
   - a session key, then apply "$H \subseteq \mathsf{parts}\,H$" to case *Forge* and simplify it by lemma 9.13.
3. Solve case *Fake* by a standard tactic [88, §4.5].
4. Apply the theorems assessing the reliability of the cards' functioning as follows: theorem 9.5 to case *SR9*, theorem 9.6 to cases *SR8* and *OopsB*, theorem 9.7 to case *SR11*, and a variant of theorem 9.7 — which binds the form of the certificate, given the form of the session key — to case *OopsA*.
5. Simplify remaining cases.

9.3.4 Unicity

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 9.19). When 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 9.19. *If evs contains*

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

then

$$B = B' \quad \mathit{and} \quad Nb = Nb' \quad \mathit{and} \quad \mathit{Cert1} = \mathit{Cert1}' \quad \mathit{and} \quad \mathit{Cert2} = \mathit{Cert2}'$$

A similar theorem, here omitted, holds about the output of step 4, exploiting the freshness of A's nonce. More surprisingly, it holds about the output of message 10 too (theorem 9.20), although the card uses no fresh components on that occasion. Corollary 9.8 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 one about *SR10_Fake* is terminated routinely.

Theorem 9.20. *If evs contains*

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

then

$$A = A' \quad \mathit{and} \quad \mathit{Cert} = \mathit{Cert}'$$

The unicity theorems may 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 has happened. Having violated theorem 9.19, 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 9.20 provides the equivalent information to A.

However, if B happens to receive the same session key within the same message more than once, theorem 9.19 would not be violated. Still, the scenario is unaccountable in the model, and thus should alarm B. Since all cards' CPUs function correctly, B's card must always compute a fresh key.

As a matter of fact, a further guarantee may be designed to assist B in this circumstance. Upon reception of any output commencing with a nonce, B can be assured that the corresponding event is unique (theorem 9.21). 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 9.21. *If evs contains*

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

then

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

9.3.5 Confidentiality

Some counterguarantees of confidentiality can be easily obtained. Even if a specific pairkey has not been issued by the server and its components cannot be forged, the pairkey cannot be proved confidential because of the weakness discovered by the authenticity argument (§9.3.3). Besides, it is straightforward to notice from messages 6 and 8 that neither A's nor B's nonce remain confidential.

On the contrary, the regularity lemmas may be viewed as non-trivial confidentiality guarantees. Moreover, applying $\mathsf{analz}\,H \subseteq \mathsf{parts}\,H$ to the authenticity theorem 9.18, we obtain a guarantee of session key confidentiality. Any session key that cannot be forged and that has not been leaked by accident is confidential (theorem 9.22). Unfortunately, the theorem is not useful to agents because the structure of the session key must be inspected.

Theorem 9.22. *If A and B are not the spy, their cards are not illegally usable, and evs does not contain*

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

then

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

This result cannot be strengthened sufficiently: we have discovered that the theorems of session key confidentiality cannot be applied by the peers within the minimal trust due to the lack of explicitness that affects two protocol steps [15]. It follows that the Shoup-Rubin protocol grants weak confidentiality guarantees to its peers unless the design is slightly modified (see §9.4). However, the guarantees presented below can be applied by the smart cards within the minimal trust, which is in general a significant outcome for a smart card protocol.

We follow Paulson's general strategy for verifying confidentiality (§4.5). The proof scripts of the resulting theorems may be found in Appendix D. The necessary simplification law for analz, the session key compromise theorem, is fairly easy to obtain, since Shoup-Rubin never sends session keys over the network. 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,

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

includes two certificates, *Cert1* and *Cert2*, that B cannot inspect because they are sealed by specific long-term keys (no agent knows any long-term keys).

We have attempted to prove Kab confidential on a trace *evs* that contains no oops event leaking Kab but that does contain the mentioned event. 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 signifies that B does not obtain explicit information about the peer with which the session key is to be used, 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" [4, §4]. If we inspect either one of the certificates, then B's peer, A, becomes explicit, so the relevant assumptions can be stated and theorem 9.23 proved.

Theorem 9.23. *If A and B are not the spy, A's card is not illegally usable, B's card is not cloned, and evs contains*

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

but does not contain

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

then

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

From B's viewpoint, trusting that the peer is not malicious and her card cannot be used by the malicious entity is indispensable. So is trusting that the key has not been leaked by accident. These assumptions constitute B's minimal trust. What is more important is that 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 based on provable security asserts an analogous property requiring that the peers' cards be "unopened" [98, §3.1], which may be interpreted as "not cloned" within our approach. However, the treatment does not highlight the lack of explicitness unveiled here, failing to investigate whether the property is in fact useful to the peers.

Similar considerations arise when reasoning from A's viewpoint in the attempt to prove confidentiality on the assumption that the event formalising step 10,

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

occurs, leaving 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. As a matter of fact, 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 9.24).

Theorem 9.24. *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

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

The analysis based on provable security does not account for this weakness. We stress that, since theorems 9.23 and 9.24 are applicable by the smart cards, they reveal lack of explicitness only through the verification of our principle of goal availability (§4.8).

9.3.6 Authentication

The lack of explicitness that affects messages 7 and 10 also weakens the goals of authentication. Only A's card obtains a useful guarantee and so can detect whether certain components are being used with the wrong peer.

Phase V terminates B's role in the protocol. Then, B's peer, 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 9.25). 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 in the traffic for some Na; then, it applies lemma 9.15.

Theorem 9.25. *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

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

This result may be interpreted as weak agreement of B's card with A's. The cards certainly know the components of their outputs, so the result also may be viewed as non-injective agreement of B's card with A's on Kab. However, expressing this formally requires extending the function knows on smart cards (§8.3), which may be of little significance due to the limited memory of the cards.

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 9.16 is applicable, A's card can be proved to have sent a suitable instance of message 10, which establishes A and her card's presence, and A's card intention to communicate with B (theorem 9.26).

Theorem 9.26. *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

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

Is this theorem useful to B? The answer is "no" because the agent cannot inspect the encrypted certificate. So, 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 that message does not state the peer either (see theorem 9.6). 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, §9.4), making the guarantee available also to B's card necessarily requires the additional steps.

9.3.7 Key Distribution

The Shoup-Rubin protocol does not achieve the goal of key distribution in the sense that each peer obtains no evidence that he shares the session key with the other one. This is due to the mentioned lack of explicitness. Nevertheless, A's card can be informed that the session key is known to B.

Applying the definition of knows to the conclusion of theorem 9.25 yields that, when A's card computes the session key for A, the key is already known to B (theorem 9.27); A cannot profit from this result. Note that the guarantee itself does not prevent B from being the spy.

Theorem 9.27. *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

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

Let us attempt to design the corresponding guarantee for B. His session key is obtained via message 7. By theorem 9.26, 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* key. This can only be done by inspecting one of the certificates of message 7, so to make A explicit. Then, theorem 9.6 specifies the form of the session key that is output by B's card (theorem 9.28).

Theorem 9.28. *If B is not the spy, A and B's cards are not illegally usable, and evs contains*

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

then

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

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 9.28 turns out to be applicable neither to B nor to his card, which is a poor outcome for the protocol.

9.4 Verifying an Upgraded Shoup-Rubin

Omitting B's name from message 2 of the public-key Needham-Schroeder protocol led to the well-known Lowe's attack [63]. Although public-key cryptography attempted to enforce confidentiality of the nonces, the spy could intercept the messages while interleaving two sessions, learn an important nonce and violate the authentication of the initiator to the responder. With Shoup-Rubin, the secure contexts between agents and smart cards prevent this. However, since the cards' data buses are not reliable (§8.1.1), when lack of explicitness affects the cards' outputs, the agents cannot distinguish which protocol session a single output belongs to.

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." [4, §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 ciphers cannot be decrypted by agents. Nor can message 10 inform A of B's identity. Besides, 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, it could be argued that, should the cards' 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, the messages 7 and 10 do violate the explicitness principles that have been mentioned throughout this chapter. Indeed, we have shown how they weaken the protocol goals when the extra assumption of reliable cards' data buses is not made.

These considerations suggest upgrading messages 7 and 10 by the components underlined in figure 9.14, while leaving the rest of the protocol unaltered. It is straightforward to upgrade the formal protocol model accordingly.

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

Figure 9.14: Upgrading the Shoup-Rubin protocol

In the new model, many of the theorems discussed above obtain slightly modified assertions and, crucially, assumptions that never inspect the ciphers. So, the assumptions have become verifiable by agents, signifying that the upgraded protocol makes its guarantees available to them. For example, theorem 9.4 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 9.6 can be stripped of one existential, so B learns the peer for the session key (theorem 9.6′). One existential still constrains the form of one of the certificates, but B's knowledge is not significantly affected.

Theorem 9.6′. *If evs contains*

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

then

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

and there exists Na such that

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

Similarly, proving theorem 9.7 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 9.7 obtains an extra component, while the rest of the authenticity argument remains unaltered.

The unicity results continue to hold. For example, theorem 9.20 must now cope with the additional components (theorem 9.20′).

Theorem 9.20′. *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

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

Theorem 9.23 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 9.24, which rests on

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

As a consequence, the peers will be able to decide, within the minimal level of 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 9.25′). Agent A must only verify receiving from her card a message with four components: an agent name, a nonce, a key, a cipher. The theorem could be more specific on the form of *Cert1*, but this would not enrich A's knowledge substantially.

Theorem 9.25′. *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, for some Cert1, evs also contains

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

The result may be interpreted (§9.3.6) as non-injective agreement of B and his card with A and her card on Kab.

Also theorem theorem 9.26 can be reformulated (theorem 9.26′) as to become applicable by B, who can check the reception from the network of a cipher previously obtained from his card.

Theorem 9.26′. *If B is not the spy, A and B's cards are not illegally usable, and evs contains*

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

then evs also contains

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

Note that the result is not applicable by B's card. It expresses non-injective agreement of A and her card with B on Kab.

Theorem 9.27 can be enforced on the same assumptions as those of theorem 9.25′, so becoming applicable by A (theorem 9.27′).

Theorem 9.27′. *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,

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

Similarly, the assertion of theorem 9.28 can be proved on the assumptions of theorem 9.26′ and become useful to B. The resulting theorem and theorem 9.27′ signify that the upgraded protocol achieves the goal of key distribution.

Chapter 10

Conclusions

The summary of the research presented throughout the dissertation is drawn and discussed. The treatment concludes with some statistics and some ideas for future work.

Establishing secure communication sessions with remote computers is vital. Security in this context may signify a variety of goals, which depend on the application domain. Enforcing them can be daunting. Practical experience has confuted many informal claims of protocol security and some formal ones. As a consequence, computer practitioners are still highly reluctant towards entering personal data into a network even when some cryptographic protocol is provided.

The main argument of this dissertation is that formal methods may substantially help to verify whether a protocol achieves the goals of authenticity, unicity, confidentiality, authentication and key distribution. We conduct this argument by extending Paulson's Inductive Approach [88] in such a way that it scales up to real-world protocols. We achieve the first full mechanisation of the following protocols: BAN Kerberos, Kerberos IV and Shoup-Rubin. The first requires the modelling of timestamps, the second necessitates of a meticulous distinction between different kinds of session keys, and the third makes it necessary to account for smart cards. Our treatment confirms and strengthens the importance of explicitness in designing protocols and supports a new principle, *goal availability*, which is met when there exist suitable guarantees assuring both peers that the protocol achieves its goals. Violating this principle raises the risk of attacks, as is the case with Kerberos IV, which makes a crucial confidentiality guarantee available to its initiator but not to its responder.

Also, it could be argued that the explicitness principles are not constructive. For example, *when* is the "identity of a principal essential to the meaning of a message" [4] ? By contrast, investigating goal availability is simple — it only requires viewing the existing formal guarantees by taking into account the agents' knowledge. We believe that, in general, checking goal availability may pinpoint unknown lack of explicitness, as demonstrated with the Shoup-Rubin protocol.

10.1 Summary

Chapter 2 presents the state of the art in reasoning about cryptographic protocols. The complexity of the formal approaches used to obtain some guarantee does not influence the relevance of the guarantee itself. On the contrary, a complex approach might reduce the importance of an eventual guarantee by making it obscure. Nonetheless, the guarantees that, at first, are easy to grasp may necessitate a more careful interpretation.

Chapter 3 outlines the original Inductive Approach to verifying cryptographic protocols, which is due to Paulson [88]. Rather than accounting informally for a malicious entity as most belief logics do, this approach clearly defines the abilities of a model spy. Paulson's approach responds with unbounded models to the limited size of the protocol models analysed by state enumeration techniques. The approach is mechanised with the generic theorem prover Isabelle, which verifies the proof steps performed by humans, as opposed to the provable security approach, whose theoretical reasoning is solely carried out on paper. The modifications and extensions we have developed on the Inductive Approach are described through the subsequent chapters along with their outcomes.

Chapter 4 discusses the protocol goals that we can prove formally and the related proof strategies. We design the reliability theorems, assessing that the formal protocol represents the real protocol faithfully. Then, we develop stronger unicity theorems than the existing ones, which state that certain events can only occur once. Some of the existing theorems are useful to show that the goals of integrity and authenticity are equivalent. Hence, the goal of confidentiality is reviewed. Furthermore, we explain how to treat the goals of authentication and key distribution within the approach. While beginning to build the machinery to reason about these goals (finalised in chapter 7), we show that a strong form of authentication is equivalent to key distribution.

Finally, the concept of minimal trust and the principle of goal availability are explained. If guarantees enforcing a protocol goal can be established on assumptions that the protocol peers can verify, excepted for those in the minimal trust, then we say that the protocol makes the goal available to its peers, namely the protocol conforms to the principle of goal availability in regard to that goal.

Chapter 5 describes how we extend the Inductive Approach with timestamps. The formalisation of time can be carried out in a simple fashion by exploiting the order of the elements of a trace.

Using a discrete view of time, the current time of a trace may be given by its length. This presupposes that no two events can occur at the same time, as does the protocol model itself, which extends the traces by a single event at a time. Following these extensions, the BAN Kerberos protocol, which is based on timestamps, can be mechanised. Its proven goals can be compared with those obtained by the BAN logic, which are only confidentiality and key distribution.

Chapter 6 contains the full mechanisation of Kerberos IV. The protocol model is quickly obtained from an existing formal specification by ASMs [21], which was itself obtained from the original informal specification of the protocol [77]. The most interesting feature of Kerberos IV is its use of the authkeys (the session keys issued during the first phase of the protocol) to encrypt the servkeys (the session keys issued during the second phase).

We also check if the protocol conforms to our principle of goal availability. In designing a confidentiality guarantee for the protocol responder, we discover that he must trust certain conditions pertaining to a protocol phase in which he has not taken part. On the basis of this observation, we point out an attack whereby the spy exploits certain servkeys within their lifetime, without the agent for whom they had been issued realising it. We prove that adding a simple temporal check to the functioning of one of the two trusted servers can prevent this attack. These considerations are unknown to any of the existing analyses of the protocol.

Chapter 7 concentrates on the knowledge that the agents' derive from handling the protocol messages. This is fundamental to reason formally about authentication and key distribution. We develop two approaches. One rests on the inspection of the trace to pinpoint the messages that each agent creates. Creation of a message implies knowledge of all its components.

Another approach, released with the 1999 distribution of Isabelle, introduces the event of message reception and defines agents' knowledge in terms of the messages that each agent sends or receives. The definition of agents' knowledge extends and replaces the existing notion of spy's knowledge.

Both approaches are demonstrated on proving authentication and key distribution on BAN Kerberos and Kerberos IV, where they seem to accomplish equivalent results. However, only the approach based on message reception can be used successfully to verify key distribution on a variant of the Otway-Rees protocol. Trace inspection regains its attraction when investigating the temporal requisites of the goal of authentication, and is indispensable to compare the outcomes of timestamps with those of nonces. Therefore, we deduce that a combination of both approaches might yield the best results, although we argue that the one based on message reception, more elegant and readable, should account for most realistic scenarios.

Chapter 8 tailors the Inductive Approach to the analysis of protocols based on smart cards. Many such protocols explicitly assume that the means between each agent and his card is secure, whereas others do not rest on this. We account for both cases by simple variations to the definition of agents' knowledge, and introduce suitable events to formalise the interaction between agents and cards.

We formalise the set of smart cards that the spy has got hold of, also discovering their pins. Another set of smart cards formalises those that she has managed to clone, discovering their internal secrets. We allow the spy to exploit all such cards. For this purpose, the rule of the protocol model formalising the spy's illegal operations must be extended. Also, in case the means between agents and cards is assumed secure, a new rule must be added each time a smart card sends an output. These new rules allow the spy to obtain the outputs of the cards that she can illegally use, and necessarily complicate the proofs.

Chapter 9 demonstrates the approach developed in the previous chapter on the Shoup-Rubin protocol, which is based on smart cards. This protocol is a significant case study because its designers have analysed it by the provable security approach, and because it has been subsequently implemented. The abstract treatment by the protocol designers makes it difficult to derive a compact view of the protocol. In this case, the implementors' paper helps significantly but, in general, long and informal specifications seem one of the major obstacles that formal analysis must face (as, for example, when modelling the SET protocol [16]).

Our proofs highlight that two of the protocol steps lack explicitness: no peer knows with whom to associate the session key received from his own card. This weakens the goals of confidentiality, authentication and key distribution achieved by the protocol. However, unlike the agents, the smart cards can decipher some of the exchanged ciphers. As a consequence, from the viewpoint of the smart cards, the mentioned lack of explicitness does not affect the three protocol goals: the cards can still apply most of the corresponding guarantees. Therefore, we remark that it is the process of verifying whether the protocol conforms to the principle of goal availability that unveils the lack of explicitness.

10.2 Statistics

All runtimes reported below are measured on a 600Mhz Intel Pentium III.

The full proof script for BAN Kerberos executes in 65 seconds when we use the approach to agents' knowledge based on trace inspection. The runtime decreases of 10 seconds when our model allows for message reception and the corresponding formalisation of agents' knowledge.

The proof runtime for Kerberos IV is 247 seconds when performing trace inspection, and 223 seconds when including message reception. Runtimes remain unvaried if we refine the operation of the second trusted server in order to prevent the attack on the servkeys.

The runtime of the full proof script for Shoup-Rubin is 160 seconds. It remains practically unvaried when we upgrade the protocol adding explicitness.

The proof scripts for different protocols can be compared as long as they contain guarantees that can be considered analogous. The scripts for BAN Kerberos, Kerberos IV and Shoup-Rubin have this feature. Kerberos IV is unquestionably a more complex protocol than BAN Kerberos. Therefore, considering the scripts for these two protocols, it appears that the more complex is the protocol under analysis, the more significant are the advantages, in terms of runtime, deriving from a simpler modelling of agents' knowledge.

Shoup-Rubin employs more messages, more objects (the smart cards, the pairkeys, etc.) and more events than Kerberos IV does. Therefore, we expected the proof script for the former to be longer than that for the latter. This turned out to be wrong: the complete script for Kerberos IV exceeds 2000 lines, is one third longer than that for Shoup-Rubin, and requires approximately one minute more to execute. It is the confidentiality argument that is more complex on Kerberos IV, due to the protocol use of encryption. Therefore, it seems that the complexity of the verification of a protocol is mostly due to the complexity of the ciphers encompassed by the protocol, which translates into complex reasoning about the analz operator, rather than to other features.

10.3 Future work

Future work mainly focuses on verifying new hierarchies of protocols and new protocol goals.

Non-repudiation protocols should be considered. They aim at solving disputes where the peers pretend not to have sent or received certain messages [105, 106]. We expect that our approach extended with message reception could easily scale up to proving the goals of *non-repudiation of origin* and *non-repudiation of receipt*.

Delegation is becoming an important issue in modern cyberlaw. Agents may want to delegate the *rights* [47] to perform certain tasks to somebody else, or they may want delegate the *responsibility* [34] of those tasks. From an operational standpoint, proving either form of delegation requires showing that the recipient of the delegation has received certain credentials.

E-commerce protocols [25, 76, 80] aim at new goals, such as *anonymity* and *accountability* [54]. Anonymity is achieved when an agent completes a protocol session without his identity being disclosed to anyone. In order to prove this goal, the agent names should be no longer guessable (this requires a simple modification to the definition of the function synth), then we should show that the initiator's identity does not belong to the knowledge that any agent can derive from the observation of the traffic. The proofs are expected to be rather complicated, as they involve frequent evaluations under the analz operator. Accountability may be viewed as a form of non-repudiation because it involves establishing that, at completion of a protocol sessions, certain events representing important steps of a financial transaction have taken place.

Non-denial of service is met when the protocol initiator can be assured that certain services will be granted [82]. From an operational point of view, the goal corresponds to establishing that certain events *will* take place on the basis of others. However, the inductive definition of the protocol does not constrain rules to fire and thus does not force events to occur, so we expect that verifying this goal will require substantial modifications to the basics of the approach.

Appendix A

Verifying Kerberos IV

We present the scripts for the session key compromise theorems proven on Kerberos IV (§6.3.5).

Script for lemma 6.11. `Reli2` is the reliability theorem 6.4.

```
Goal "[| K : AuthKeys evs Un range shrK; evs : kerberos |]   \
\    ==> ALL K'. ~ AKcryptSK K' K evs";
by (asm_full_simp_tac (simpset() addsimps [AKcryptSK_def]) 1);
by (blast_tac (claset() addDs [Reli2]) 1);
qed"AKcryptSK1";
```

Script for lemma 6.12. `Uniq_Tgs` is the unicity guarantee about the servkeys issued by `Tgs`.

```
Goal "[| AKcryptSK authK servK evs;                          \
\        K ~= authK; evs : kerberos |]                       \
\    ==> ~ AKcryptSK K servK evs";
by (asm_full_simp_tac (simpset() addsimps [AKcryptSK_def]) 1);
by (blast_tac (claset() addDs [Uniq_Tgs]) 1);
qed "AKcryptSK2";
```

Script for lemma 6.13. `Reli2` is the reliability theorem 6.4.

```
Goal "[| servK ~: AuthKeys evs; servK ~: range shrK;         \
\        evs : kerberos |]                                   \
\    ==> ALL K. ~ AKcryptSK servK K evs";
by (asm_full_simp_tac (simpset() addsimps [AKcryptSK_def]) 1);
by (blast_tac (claset() addDs [Reli2]) 1);
qed "AKcryptSK3";
```

Script for lemma 6.14. It is obtained from another lemma that holds on sets of keys (analz_insert_set_key_rewrite, below); analz_simpset is a specialised simpset for evaluation of expressions containing analz.

```
Goal "evs : kerberos ==>                                     \
\     (K ~: range shrK --> (~ AKcryptSK K K' evs) -->         \
\          (Key K' : analz (insert (Key K) (spies evs))) =    \
\          (K' = K | Key K' : analz (spies evs)))";
by (asm_full_simp_tac (analz_simpset
                 addsimps [analz_insert_set_key_rewrite]) 1);
qed"analz_insert_key_rewrite";
```

Script for lemma analz_insert_set_key_rewrite. Most of the applied theorems are omitted here.

```
Goal "evs : kerberos ==>                                     \
\     (ALL K KK. KK <= Compl (range shrK) -->                 \
\       (ALL K': KK. ~ AKcryptSK K' K evs) -->                \
\          (Key K : analz (Key``KK Un (spies evs))) =         \
\          (K : KK | Key K : analz (spies evs)))";
by (etac kerberos.induct 1);
by (forward_tac [Oops_range_spies1] 9);
by (forward_tac [Oops_range_spies2] 11);
by analz_sees_tac;
by (REPEAT_FIRST (rtac allI));
by (REPEAT_FIRST (rtac (analz_rewrite_lemma RS impI)));
by (ALLGOALS
    (asm_simp_tac
     (analz_simpset addsimps
       [AKcryptSK_Says, shrK_not_AKcryptSK,
        fresh_authK_not_AKcryptSK, fresh_servK_not_AKcryptSK,
        AKcryptSK_I, Spy_analz_shrK])));
(*Fake*)
by (spy_analz_tac 1);
(*K3*)
by (Blast_tac 1);
(*K4*)
by (blast_tac (claset() addEs spies_partsEs
                       addSDs [authK_not_AKcryptSK]) 1);
(*K5*)
br impI 1;
```

```
by (case_tac "Key servK : analz (spies evs5)" 1);
by (asm_simp_tac
     (simpset() addsimps [analz_insert_eq,
       impOfSubs (Un_upper2 RS analz_mono)]) 1);
by (case_tac "AKcryptSK servK K evs5" 1);
by (asm_simp_tac analz_image_freshK_ss 2);
by (blast_tac (claset()
               addSEs [servK_not_AKcryptSK RSN(2, rev_notE)]
       addEs spies_partsEs delrules [allE, ballE]) 1);
(*Level 16: OopsS*)
by (case_tac "Key servK : analz (spies evsOs)" 2);
by (forward_tac [analz_mono_KK] 2);
by (assume_tac 2);
by (assume_tac 2);
by (Asm_simp_tac 2);
by (Clarify_tac 2);
by (forward_tac [analz_cut] 2 THEN assume_tac 2);
by (blast_tac (claset()
               addDs [analz_cut,impOfSubs analz_mono]) 2);
by (dres_inst_tac [("x","K")] spec 2);
by (dres_inst_tac [("x","insert servK KK")] spec 2);
by (forward_tac [Reli2] 2 THEN assume_tac 2);
by (Clarify_tac 2);
by (forward_tac [Says_imp_spies RS parts.Inj RS parts.Body
                 RS parts.Snd RS parts.Snd RS parts.Snd] 2);
by (dres_inst_tac [("K","K")] servK_not_AKcryptSK 2);
by (assume_tac 2);
by (assume_tac 2);
by (assume_tac 2);
by (Asm_full_simp_tac 2);
by (blast_tac (claset() addDs [impOfSubs analz_mono]) 2);
(*Level 32: OopsA*)
by (dres_inst_tac [("x","K")] spec 1);
by (dres_inst_tac [("x","insert authK KK")] spec 1);
by (Asm_full_simp_tac 1);
by (case_tac "AKcryptSK authK K evsOa" 1);
by (blast_tac (claset() addSDs [AKcryptSK_analz_insert]) 1);
by (blast_tac (claset() addDs [impOfSubs analz_mono]) 1);
qed_spec_mp "analz_insert_set_key_rewrite";
```

Script for theorem 6.15.

```
Goal "[| K : (AuthKeys evs) Un range shrK;                    \
\        K' ~: range shrK; evs : kerberos|]                   \
\    ==> Key K : analz (insert (Key K') (spies evs)) =        \
\        (K = K' | Key K : analz (spies evs))";
by (forward_tac [AKcryptSK1] 1);
by (assume_tac 1);
by (asm_full_simp_tac (analz_simpset
                      addsimps [analz_insert_key_rewrite]) 1);
qed "sesK_compromise1";
```

Script for theorem 6.16.

```
Goal "[| AKcryptSK authK servK evs;                           \
\        K ~= authK; evs : kerberos |]                        \
\    ==> Key servK : analz (insert (Key K) (spies evs)) =    \
\        (servK = K | Key servK : analz (spies evs))";
by (forward_tac [AKcryptSK2] 1);
by (assume_tac 1);
by (asm_full_simp_tac (analz_simpset
                      addsimps [analz_insert_key_rewrite]) 1);
qed "sesK_compromise2";
```

Script for theorem 6.17.

```
Goal "[| servK ~: (AuthKeys evs); servK ~: range shrK;       \
\        evs : kerberos |]                                    \
\    ==> Key K : analz (insert (Key servK) (spies evs)) =    \
\        (K = ServKey | Key K : analz (spies evs))";
by (forward_tac [AKcryptSK3] 1);
by (assume_tac 1);
by (asm_full_simp_tac (analz_simpset
                      addsimps [analz_insert_key_rewrite]) 1);
qed "sesK_compromise3";
```

Appendix B

Proving an Issues Property

We present the scripts for obtaining theorem 7.11, which conveys non-injective agreement of A with B on the session key (§7.3.1).

Script for theorem 7.10. The script for `BK3_help` is given below; `BK2_auth` is the authenticity theorem 5.3; `Auth_A_to_B` is the version of the authentication theorem 5.10 where the confidentiality assumption is not yet relaxed by theorem 5.9 (see below).

```
Goal "[| Says A B {|Ticket, Crypt K {|Agent A, Number Ta|}|} \
\          : set evs;                                        \
\        Key K ~: analz (spies evs);                         \
\        A ~=Spy; evs : bankerberos |]                       \
\    ==> A Issues B with (Crypt K {|Agent A, Number Ta|})    \
\        on evs";
by (case_tac "B : bad" 1);
by (forward_tac [BK3_help] 1);
by (assume_tac 1);
by (assume_tac 1);
by (assume_tac 1);
bd (Says_imp_spies RS analz.Inj RS analz.Fst) 1;
by (fast_tac (claset() addDs
                        [MPair_analz] addss (simpset())) 1);
(*Now B is not bad*)
by (simp_tac (simpset() addsimps [Issues_def]) 1);
br exI 1;
br conjI 1;
ba 1;
```

```
by (Simp_tac 1);
be rev_mp 1;
be rev_mp 1;
by (etac bankerberos.induct 1);
by (forward_tac [BK3_in_parts_spies] 5);
by (REPEAT (FIRSTGOAL analz_mono_contra_tac));
by (ALLGOALS (asm_simp_tac (simpset()
                            addsimps [all_conj_distrib])));
(*BK3*)
by (Clarify_tac 1);
by (asm_full_simp_tac (simpset()
                       addsimps [takeWhile_tail]) 1);
by (not_bad_tac "A" 1);
by (blast_tac (claset() addDs [BK2_auth,
                         impOfSubs parts_spies_takeWhile_mono,
                             impOfSubs parts_spies_evs_revD2]
                        addSIs [Auth_A_to_B]
                        addEs spies_partsEs) 1);
qed"A_Issues_BK3";
```

Script for KB3_help; BK2_auth is the authenticity theorem 5.3; Reli is the reliability theorem 5.1.

```
Goal "[|Says A B {|Ticket, Crypt K {|Agent A, Number Ta|}|}  \
\         : set evs;                                          \
\       Key K ~: analz (spies evs);                           \
\       A ~= Spy; evs : bankerberos |]                        \
\   ==> EX Tk. Ticket =                                       \
\             (Crypt (shrK B) {|Number Tk, Agent A, Key K|})";
by (etac rev_mp 1);
by (etac rev_mp 1);
by (etac bankerberos.induct 1);
by (forward_tac [BK3_in_parts_spies] 5);
by (REPEAT (FIRSTGOAL analz_mono_contra_tac));
by (ALLGOALS (asm_simp_tac (simpset()
                            addsimps [all_conj_distrib])));
(*KB3*)
by (Clarify_tac 1);
by (not_bad_tac "A" 1);
by (blast_tac (claset() addDs [BK2_auth RS Reli]
                        addEs spies_partsEs) 1);
```

```
qed"KB3_help";
```

Script for theorem 7.11.

```
Goal "[| Crypt (shrK B) {|Number Tk, Agent A, Key K|}       \
\          : parts (spies evs);                              \
\        Crypt K {|Agent A, Number Ta|}                      \
\          : parts (spies evs);                              \
\        Key K ~: analz (spies evs);                         \
\        A ~= Spy; B ~: bad; evs : bankerberos |]            \
\    ==> A Issued B with (Crypt K {|Agent A, Number Ta|})    \
\        on evs";
by (blast_tac (claset() addDs
                        [Auth_A_to_B, A_Issues_BK3]) 1);
```

Script for Auth_A_to_B; BK3_auth is the authenticity theorem 5.4.

```
Goal "[| Crypt K {|Agent A, Number Ta|}                      \
\          : parts (spies evs);                              \
\        Crypt (shrK B) {|Number Tk, Agent A, Key K|}        \
\          : parts (spies evs);                              \
\        Key K ~: analz (spies evs);                         \
\        B ~: bad;  evs : bankerberos |]                     \
\    ==> Says A B                                            \
\           {|Crypt (shrK B) {|Number Tk, Agent A, Key K|}, \
\             Crypt K {|Agent A, Number Ta|}|} : set evs";
by (blast_tac (claset() addSDs [BK3_auth]
                        addSIs [Auth_A_to_B_lemma]) 1);
qed "Auth_A_to_B";
```

Script for Auth_A_to_B_lemma; Reli' is a variant of theorem 5.1; Unic1 is the unicity theorem 5.5.

```
Goal "[| B ~: bad; evs : bankerberos |]                      \
\    ==> Key K ~: analz (spies evs) -->                      \
\        Says Server A                                       \
\         (Crypt (shrK A) {|Number Tk, Agent B, Key K, X|}) \
\          : set evs -->                                     \
\        Crypt K {|Agent A, Number Ta|}                      \
\          : parts (spies evs) -->                           \
\        Says A B {|X, Crypt K {|Agent A, Number Ta|}|}      \
\            : set evs";
```

```
by (etac bankerberos.induct 1);
by (forward_tac [Reli'] 5 THEN assume_tac 5);
by (forward_tac [BK3_in_parts_spies] 5);
by (forward_tac [Oops_parts_spies] 7);
by (REPEAT (FIRSTGOAL analz_mono_contra_tac));
by (ALLGOALS (asm_simp_tac (simpset()
                            addsimps [all_conj_distrib])));
(*Fake*)
by (Blast_tac 1);
(*BK2*)
by (Clarify_tac 1);
by (dtac Crypt_imp_invKey_keysFor 1);
by (Asm_full_simp_tac 1);
(*BK3*)
by (Clarify_tac 1);
by (not_bad_tac "A" 1);
by (blast_tac (claset() addDs [BK2_auth, Unic1]) 1);
qed "Auth_A_to_B_lemma";
```

Appendix C

Defining Agents' Knowledge

We present the inductive definition of agents' knowledge for a smart card protocol that assumes secure means between agents and smart cards (§8.3). The initial knowledge is as definend for Shoup-Rubin (§9.2).

```
primrec

(*Server knows all long-term secrets*)
initState_Server
  "initState Server =
    (Key``(range shrK Un range crdK Un range pin Un
           range pairK)) Un
    (Nonce``(range Pairkey))"

(*Friendly agents know no secrets*)
initState_Friend  "initState (Friend i) = {}"

(*Spy knows the long-term secrets deriving from cloned cards*)
initState_Spy
  "initState Spy  =
    (Key``((crdK``cloned) Un (shrK``{A. Card A : cloned}) Un
           (pairK``{(A,B). Card B : cloned}))) Un
    (Nonce``(Pairkey``{(A,B). Card A : cloned &
                              Card B : cloned}))"
```

```
primrec

knows_Nil  "knows A [] = initState A"

knows_Cons
  "knows A (ev # evs) =
     (case ev of

        Says A' B X => if (A = A' | A = Spy)
                       then insert X (knows A evs)
                       else knows A evs

      | Notes A' X  => if (A = A' | (A = Spy & A' : bad))
                       then insert X (knows A evs)
                       else knows A evs

      | Gets A' X   => if (A = A' & A ~= Spy)
                       then insert X (knows A evs)
                       else knows A evs

      | Inputs A' C X => if A = A'
                         then insert X (knows A evs)
                         else knows A evs

      | Outputs C A' X => if A = A'
                          then insert X (knows A evs)
                          else knows A evs)"
```

Appendix D

Verifying Shoup-Rubin

We present the scripts for the arguments of authenticity (§9.3.3) and of confidentiality (§9.3.5) on Shoup-Rubin.

Tactics for applying the relevant reliability theorems; `B_Out_reli` is theorem 9.6; `A_Out_reli` is theorem 9.5; `A_Out_reli'` is theorem 9.7; the corollary of theorem 9.7 that binds the form of the session key, once the form of the certificate is known, is `A_Out_reli''`.
Theorem `Gets_imp_knows_Spy RS parts.Inj RS parts.Snd` says that, if a two-component message is received from the network, the second component is in the traffic.

```
val prepare_tac =
 (*SR8*)   forward_tac [B_Out_reli] 13 THEN
 (*SR9*)   dtac A_Out_reli 15 THEN
 (*SR11*)  forward_tac [A_Out_reli'] 20;

val parts_prepare_tac =
           prepare_tac THEN
 (*SR9*)   dtac (Gets_imp_knows_Spy RS parts.Inj RS parts.Snd)
             17 THEN
           dtac (Gets_imp_knows_Spy RS parts.Inj RS parts.Snd)
             18 THEN
 (*OopsB*) dtac B_Out_reli   25 THEN
 (*OopsA*) dtac A_Out_reli'' 27 THEN
 (*Base*)  Force_tac 1;
```

```
val analz_prepare_tac =
          prepare_tac THEN
 (*SR9*)   dtac (Gets_imp_knows_Spy RS analz.Inj RS analz.Snd)
            17 THEN
          dtac (Gets_imp_knows_Spy RS analz.Inj RS analz.Snd)
            18 THEN
          REPEAT_FIRST (eresolve_tac
                      [asm_rl, conjE] ORELSE' hyp_subst_tac);
```

Script for authenticity lemma 9.14.

```
Goal "[| Crypt (shrK A) {|Nonce Pk, Agent B|}               \
\          : parts (knows Spy evs);                           \
\        Card A ~: cloned; evs : shouprubin |]                \
\    ==> Says Server A {|Nonce Pk,                           \
\                     Crypt (shrK A) {|Nonce Pk, Agent B|}|}\
\          : set evs";
be rev_mp 1;
by (etac shouprubin.induct 1);
by parts_prepare_tac;
by (ALLGOALS Asm_simp_tac);
(*Fake*)
by (Fake_parts_insert_tac 1);
(*SR8*)
by (fast_tac (claset() addss (simpset())) 1);
(*SR11*)
by (fast_tac (claset() addss (simpset())) 1);
qed"SR2_cert_auth";
```

Script for authenticity lemma 9.15.

```
Goal "[| Crypt (pairK(A,B)) {|Nonce Na, Nonce Nb|}          \
\          : parts (knows Spy evs);                           \
\        ~illegallyU(Card B); evs : shouprubin |]            \
\    ==> Outpts (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)|} : set evs";
be rev_mp 1;
by (etac shouprubin.induct 1);
by parts_prepare_tac;
by (ALLGOALS Asm_simp_tac);
(*Fake*)
by (Fake_parts_insert_tac 1);
(*SR7_Fake*)
by (Clarify_tac 1);
(*SR8*)
by (fast_tac (claset() addss (simpset())) 1);
(*SR11*)
by (fast_tac (claset() addss (simpset())) 1);
qed"SR7_cert_auth";
```

Script for the authenticity lemma that resembles the preceding one but concerns the certificate for B's nonce.

```
Goal "[| Crypt (pairK(A,B)) (Nonce Nb)                    \
\          : parts (knows Spy evs);                        \
\        ~illegallyU(Card B); evs : shouprubin |]          \
\    ==> EX Na. Outpts (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)|} : set evs";
be rev_mp 1;
by (etac shouprubin.induct 1);
by parts_prepare_tac;
by (ALLGOALS Asm_simp_tac);
(*Fake*)
by (Fake_parts_insert_tac 1);
(*SR7*)
by (Blast_tac 1);
(*SR7_Fake*)
by (Blast_tac 1);
(*SR8*)
by (fast_tac (claset() addss (simpset())) 1);
(*SR10*)
by (blast_tac (claset() addDs [SR7_cert_auth,
                              Inputs_imp_knows_Spy RS parts.Inj]
                        addEs knows_Spy_partsEs) 1);
(*SR10_Fake*)
by (blast_tac (claset() addDs [SR7_cert_auth,
                              Inputs_imp_knows_Spy RS parts.Inj]
                        addEs knows_Spy_partsEs) 1);
```

```
(*SR11*)
by (Clarify_tac 1);
by (Asm_full_simp_tac 1);
by (blast_tac (claset() addDs [SR7_cert_auth,
                               SR7_cert_in_trafficSR10]) 1);
qed"SR7_cert_auth'";
```

Script for lemma `SR7_cert_in_trafficSR10`; `Inp_CardA_reli` is the reliability theorem 9.17.

```
Goal "[| Outpts (Card A) A                                  \
\           {|Key K, Crypt (pairK(A,B)) (Nonce Nb)|}   \
\          : set evs; evs : shouprubin |]                   \
\    ==> EX Na. Crypt (pairK(A,B)) {|Nonce Na, Nonce Nb|}   \
\                  : parts (knows Spy evs)";
be rev_mp 1;
by (etac shouprubin.induct 1);
by parts_prepare_tac;
by (ALLGOALS Asm_simp_tac);
(*Fake*)
by (Fake_parts_insert_tac 1);
(*SR7*)
by (Force_tac 1);
(*SR7_Fake*)
by (Force_tac 1);
(*SR8*)
by (fast_tac (claset() addss (simpset())) 1);
(*SR10*)
by (blast_tac (claset() addDs
                         [Inputs_imp_knows_Spy, parts.Inj,
                          Inp_CardA_reli, Gets_imp_knows_Spy]
                        addEs knows_Spy_partsEs) 1);
(*SR10_Fake*)
by (blast_tac (claset() addDs
                         [Inputs_imp_knows_Spy, parts.Inj,
                          Inp_CardA_reli, Gets_imp_knows_Spy]
                        addEs knows_Spy_partsEs) 1);
(*SR11*)
by (fast_tac (claset() addss (simpset())) 1);
qed"SR7_cert_in_trafficSR10";
```

Script for authenticity lemma 9.16.

```
Goal "[| Crypt (pairK(A,B)) (Nonce Nb)                     \
\          : parts (knows Spy evs);                          \
\        ~illegallyU(Card A); ~illegallyU(Card B);           \
\        B ~= Spy; evs : shouprubin |]                       \
\    ==> Outpts (Card A) A {|Key (sesK(Nb,pairK(A,B))),      \
\                 Crypt (pairK(A,B)) (Nonce Nb)|} : set evs";
be rev_mp 1;
by (etac shouprubin.induct 1);
by parts_prepare_tac;
by (ALLGOALS Asm_simp_tac);
(*Fake*)
by (Fake_parts_insert_tac 1);
(*SR7*)
by (Clarify_tac 1);
(*SR7_Fake*)
by (Clarify_tac 1);
(*SR8*)
by (fast_tac (claset() addss (simpset())) 1);
(*SR10F: level 25*)
by (Clarify_tac 1);
(*SR11*)
by (fast_tac (claset() addss (simpset())) 1);
qed"SR10_cert_auth";
```

Script for the authenticity theorem 9.17.

```
Goal "[| Inputs A (Card A)                                   \
\        {|Agent B, Nonce Na, Nonce Nb, Nonce Pk,            \
\         Crypt (shrK A) {|Nonce Pk, Agent B|},              \
\         Crypt (pairK(A,B)) {|Nonce Na, Nonce Nb|}, Cert3|}\
\           : set evs;                                       \
\        Card A ~: cloned; ~illegallyU(Card B);              \
\        A ~= Spy; evs : shouprubin |]                       \
\    ==> Says Server A                                       \
\         {|Nonce Pk, Crypt (shrK A) {|Nonce Pk, Agent B|}|}\
\          : set evs &                                       \
\        Outpts (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)|}               \
\          : set evs &                                      \
\        Outpts (Card A) A {|Nonce Na, Cert3|} : set evs";
by (blast_tac (claset() addDs [Inp_CardA_reli,
                Gets_imp_knows_Spy RS parts.Inj RS parts.Snd,
                               SR2_cert_auth,
                               SR7_cert_auth]) 1);
```

Script for the authenticity lemma 9.18.

```
Goal "[| Key (sesK(Nb,pairK(A,B))) : parts (knows Spy evs); \
\        ~illegallyU(Card A); ~illegallyU(Card B);          \
\        A ~= Spy; B ~= Spy; evs : shouprubin |]            \
\     ==> Notes Spy {|Key (sesK(Nb,pairK(A,B))),            \
\                      Nonce Nb, Agent A, Agent B|}          \
\           : set evs";
be rev_mp 1;
by (etac shouprubin.induct 1);
by parts_prepare_tac;
by (ALLGOALS Asm_simp_tac);
(*fake*)
by (Fake_parts_insert_tac 1);
(*Forge*)
by (fast_tac (claset() addDs [analz.Inj]
                       addss (simpset())) 1);
(*SR7*)
by (Clarify_tac 1);
(*SR7_Fake*)
by (Clarify_tac 1);
(*SR8*)
by (fast_tac (claset() addDs [B_Out_reli]
                       addss (simpset())) 1);
(*SR10*)
by (Clarify_tac 1);
(*SR10_Fake*)
by (Clarify_tac 1);
(*SR11*)
by (fast_tac (claset() addss (simpset())) 1);
(*OopsB*)
by (Asm_full_simp_tac 1);
```

```
qed"sesK_counter_auth";
```

Script for the confidentiality theorem 9.23

```
Goal "[| Outpts (Card B) B {|Nonce Nb, Key K, Verifier,     \
\                            Crypt (pairK(A,B)) (Nonce Nb)|}\
\          : set evs;                                       \
\       Notes Spy {|Key K, Nonce Nb, Agent A, Agent B|}     \
\          ~: set evs;                                      \
\        ~illegallyU(Card A); Card B ~: cloned;             \
\        A ~= Spy; B ~= Spy; evs : shouprubin |]            \
\     ==> Key K ~: analz (knows Spy evs)";
be rev_mp 1;
be rev_mp 1;
by (etac shouprubin.induct 1);
by analz_prepare_tac;
by (ALLGOALS (asm_simp_tac (simpset() addsimps
                     [analz_insert_eq, analz_insert_freshK] @
     pushes @ split_ifs)));
(*Fake*)
by (spy_analz_tac 1);
(*Forge*)
by (blast_tac (claset() addDs
                    [B_Out_reli, parts.Inj]) 1);
(*SR7*)
by (blast_tac (claset() addSDs [B_Out_reli,
                           impOfSubs analz_subset_parts]) 1);
(*SR7_Fake*)
by (fast_tac (claset() addDs [Outpts_parts_used] addss
                       (simpset())) 1);
(*SR8*)
by (fast_tac (claset() addDs [A_Out_reli''] addss
                       (simpset() addsimps split_ifs)) 1);
(*SR10*)
by (fast_tac (claset() addDs [B_Out_reli]
                       addss (simpset())) 1);
(*SR10_Fake*)
by (Clarify_tac 1);
by (dtac B_Out_reli 1 THEN assume_tac 1);
by (Asm_full_simp_tac 1);
(*SR11*)
```

```
by (fast_tac (claset() addss (simpset()
                       addsimps split_ifs)) 1);
(*OopsB*)
by (blast_tac (claset() addSDs [B_Out_reli']) 1);
(*OopsA*)
by (blast_tac (claset() addSDs [B_Out_reli',
                                A_Out_reli'']) 1);
qed"B_sesK_conf";
```

Script for the confidentiality theorem 9.24.

```
Goal "[| Outpts (Card A) A                              \
\           {|Key K, Crypt (pairK(A,B)) (Nonce Nb)|}        \
\           : set evs;                                  \
\        Notes Spy {|Key K, Nonce Nb, Agent A, Agent B|}     \
\           ~: set evs;                                 \
\         ~illegallyU(Card A); ~illegallyU(Card B);          \
\         A ~= Spy; B ~= Spy; evs : shouprubin |]          \
\     ==> Key K ~: analz (knows Spy evs)";
be rev_mp 1;
be rev_mp 1;
by (etac shouprubin.induct 1);
by analz_prepare_tac;
by (ALLGOALS (asm_simp_tac (simpset()
           addsimps [analz_insert_eq, analz_insert_freshK] @
     pushes @ split_ifs)));
(*Fake*)
by (spy_analz_tac 1);
(*Forge*)
by (blast_tac (claset() addDs [A_Out_reli'', parts.Inj]) 1);
(*SR7*)
by (blast_tac (claset() addSDs [A_Out_reli'']) 1);
(*SR7_Fake*)
by (fast_tac (claset() addDs [Outpts_parts_used]
                       addss (simpset())) 1);
(*SR8*)
by (fast_tac (claset() addDs [A_Out_reli''] addss
                       (simpset() addsimps split_ifs)) 1);
(*SR10*)
br conjI 1;
by (blast_tac (claset() addDs [A_Out_reli'']) 1);
by (blast_tac (claset() addDs [Gets_imp_knows_Spy RS
                               parts.Inj, Inp_CardA_reli,
                               SR7_cert_auth, B_sesK_conf]
                        addEs knows_Spy_partsEs) 1);
(*SR10_Fake*)
by (blast_tac (claset() addDs [A_Out_reli'']) 1);
(*SR11*)
by (fast_tac (claset() addss (simpset()
                       addsimps split_ifs)) 1);
(*OopsB*)
by (blast_tac (claset() addSDs [B_Out_reli',
                                A_Out_reli'']) 1);
(*OopsA*)
by (blast_tac (claset() addDs [CardA_unic]) 1);
qed"A_sesK_conf";
```

Bibliography

[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 A. Gordon. A Calculus for Cryptographic Protocols: the Spi Calculus. Research Report 414, University of Cambridge — Computer Laboratory, 1997.

[3] M. Abadi and A. Gordon. Reasoning about Cryptographic Protocols in the Spi Calculus. In A. W. Mazurkiewicz and J. Winkowski, editors, *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *Lecture Notes in Computer Science*, pages 59–73. Springer-Verlag, 1997.

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

[5] Martin Abadi and Mark Tuttle. A Semantics for a Logic of Authentication. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing (PODC'91)*, pages 201–216, 1991.

[6] R. Anderson. Why Cryptosystems Fail. In *Proceedings of the 1st ACM Conference on Communications and Computer Security (CCS'93)*, pages 217–227. ACM Press and Addison Wesley, 1993.

[7] R. Anderson and R. M. Needham. Programming Satan's Computer. In J. Van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–441. Springer-Verlag, 1995.

[8] R. J. Anderson and M. J. Kuhn. Low Cost Attacks on Tamper Resistant Devices. In M. et al. Lomas, editor, *Proceedings of the 5th*

International Workshop on Security Protocols, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, 1997.

[9] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated Group Key Agreement and Friends. In *Proceedings of the 5th ACM Conference on Computer and Communication Security (CCS'98)*, pages 17–26. ACM Press and Addison Wesley, 1998.

[10] M. Barjaktarovic, C. Shiu-Kai, J. Faust, C. Hosmer, D. Rosenthal, M. Stillman, G. Hird, and D. Zhou. Analysis and Implementation of Secure Electronic Mail Protocols. In H. Orman and C. Meadows, editors, *Proceedings of the Workshop on design and formal verification of security protocols, DIMACS*, 1997.

[11] G. Bella. Message Reception in the Inductive Approach. Research Report 460, University of Cambridge — Computer Laboratory, 1999.

[12] G. Bella. Modelling Agents' Knowledge Inductively. In *Proceedings of the 7th International Workshop on Security Protocols*, volume 1796 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

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

[14] G. Bella. Inductive Verification of Smart Card Protocols. *Submitted to Journal of Computer Security*, 2000.

[15] G. Bella. Lack of Explicitness Strikes Back. In *Proceedings of the 8th International Workshop on Security Protocols*, Lecture Notes in Computer Science. Springer-Verlag, 2000. In press.

[16] G. Bella, F. Massacci, L. C. Paulson, and P. Tramontano. Making Sense of Specifications: the Formalization of SET. In *Proceedings of the 8th International Workshop on Security Protocols*, Lecture Notes in Computer Science. Springer-Verlag, 2000. In press.

[17] G. Bella and L. C. Paulson. Using Isabelle to prove Properties of the Kerberos Authentication System. In H. Orman and C. Meadows, editors, *Proceedings of the Workshop on Design and Formal Verification of Security Protocols, DIMACS*, 1997.

[18] G. Bella and L. C. Paulson. Are Timestamps Worth the Effort? A Formal Treatment. Research Report 447, University of Cambridge — Computer Laboratory, 1998.

[19] G. Bella and L. C. Paulson. Kerberos Version IV: Inductive Analysis of the Secrecy Goals. In J.-J. Quisquater, Y. Desware, C. Meadows, and D. Gollmann, editors, *Proceedings of the 5th European Symposium on Research in Computer Security (ESORICS'98)*, volume 1485 of *Lecture Notes in Computer Science*, pages 361–375. Springer-Verlag, 1998.

[20] G. Bella and L. C. Paulson. Mechanising BAN Kerberos by the Inductive Method. In A. J. Hu and M. Y. Vardi, editors, *Proceedings of the International Conference on Computer-Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, 1998.

[21] G. Bella and E. Riccobene. Formal Analysis of the Kerberos Authentication System. *Journal of Universal Computer Science*, 3(12):1337–1381, 1997.

[22] G. Bella and E. Riccobene. A Realistic Environment for Crypto-Protocol Analyses by ASMs. In U. Glässer, editor, *Proceedings of the 5th International Workshop on Abstract State Machines (Informatik'98)*, pages 127–138, 1998.

[23] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In D. R. Stinson, editor, *Proceedings of Advances in Cryptography — CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

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

[25] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Jung. The KryptoKnight Family of Light-Weight Protocols for Authentication and Key Distribution. *IEEE/ACM Transactions on Networking*, 3(1):31–41, February 1995.

[26] A. Bleeker and L. Meertens. A Semantics for BAN Logic. In H. Orman and C. Meadows, editors, *Proceedings of the Workshop on design and formal verification of security protocols, DIMACS*, 1997.

[27] E. Börger. Annotated Bibliography on Evolving Algebras. In E. Börger, editor, *Specification and Validation Methods*, pages 37–52. Oxford University Press, 1995.

[28] E. Börger and L. Mearelli. Integrating ASMs into the Software Development Life Cycle. *Journal of Universal Computer Science*, 3(5):603–665, 1997.

[29] S. Brackin. A HOL Extension of GNY for Automatically Cryptographic Protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[30] S. H. Brackin. Automatic Formal Analyses of Two Large Commercial Protocols. In H. Orman and C. Meadows, editors, *Proceedings of the Workshop on Design and Formal Verification of Security Protocols, DIMACS*, 1997.

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

[32] S. Cingolani. Il Nuovo Boom di Internet e la Vendetta di Zio Paperone. *Corriere della Sera*, page 2, 4th January 2000. Italian newspaper.

[33] E. M. Clarke, S. Jha, and W. Marrero. Using State Space Exploration and a Natural Deduction Style Message Derivation Engine to Verify Security Protocols. In D. Gries and W. P. De Roever, editors, *Proceedings of the IFIP Working Conference on Programming Concepts and Methods (PROCOMET'98).* Chapmann & Hall, 1998.

[34] B. Crispo. Delegation of Responsibilities. In B. Christianson, B. Crispo, W. S. Harbison, and M. Roe, editors, *Proceedings of the 6th International Workshop on Security Protocols*, volume 1550 of *Lecture Notes in Computer Science*, pages 118–130. Springer-Verlag, 1998.

[35] Z. Dang and R. A. Kemmerer. Using the ASTRAL Model Checker for Cryptographic Protocol Analysis. In H. Orman and C. Meadows, editors, *Proceedings of the Workshop on Design and Formal Verification of Security Protocols, DIMACS*, 1997.

[36] D. E. Denning and Sacco G. M. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[37] D. L. Dill. *Murphi Description Language and Verifier*, 1996. `http://verify.stanford.edu/dill/murphi.html`.

[38] K. Eastaughffe. Algebraic Properties of Binary Xor and the Verification of Authentication Protocols. Technical report, University of Cambridge — Computer Laboratory, 1999. In preparation.

[39] F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand Spaces. Research Report 67, The MITRE Corporation, 1997.

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

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

[42] A. Gargantini and E. Riccobene. Encoding Abstract State Machines in PVS. In *Proceedings of the ASM2000 Workshop*, Lecture Notes in Computer Science. Springer-Verlag, 2000. In Press.

[43] D. Gollmann. What do we mean by Entity Authentication? In *Proceedings of the 15th IEEE Symposium on Security and Privacy*, pages 46–54. IEEE Computer Society Press, 1996.

[44] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning About Belief in Cryptographic Protocols. In *Proceedings of the 9th IEEE Symposium on Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.

[45] M. J. C. Gordon and T. F. Melham. *Introduction to HOL*. Cambridge University Press, 1993.

[46] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.

[47] B. Harbison. Delegating Trust. In B. Christianson, B. Crispo, W. S. Harbison, and M. Roe, editors, *Proceedings of the 6th International Workshop on Security Protocols*, volume 1550 of *Lecture Notes in Computer Science*, pages 108–117. Springer-Verlag, 1998.

[48] J. Hastad, R. Impagliazzo, L. Levin, and M. Luby. Construction of a pseudo-random generator from any one-way function. Manuscript. Earlier version in *Symposium on Theory of Computing (STOC'90)*.

[49] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[50] International Organization for Standardization. *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture.* ISO 7498-2, 1989.

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

[52] 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, *Proceedings of the 3rd Smart Card Research and Advanced Application Conference (CARDIS'98)*, 1998.

[53] D. Kahn. *The Codebreakers.* Macmillan, 1967.

[54] R. Kailar. Reasoning About Accountability in Protocols for Electronic Commerce. In *Proceedings of the 14th IEEE Symposium on Security and Privacy*, pages 236–250. IEEE Computer Society Press, 1995.

[55] S. Katzenbeisser and F. A. P. Petitcolas, editors. *Information Hiding Techniques for Steganography and Digital Watermarking.* Artech House, 2000.

[56] R. Kemmerer, C. A. Meadows, and J. Millen. Three Systems for Cryptographic Protocol Analyses. *Journal of Cryptology*, 7(2), 1994.

[57] J. Kohl and B. Neuman. *The Kerberos Network Authentication Service (version 5).* Internet Request for Comment RFC-1510, September 1993.

[58] J. Kohl, B. Neuman, and T. Ts'o. The Evolution of the Kerberos Authentication System. In *Distributed Open System*, pages 78–94. IEEE Computer Society Press, 1994.

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

[60] T. Leighton and S. Micali. Secret-key Agreement without Public-key Cryptography. In D. R. Stinson, editor, *Proceedings of Advances in*

Cryptography — CRYPTO'93, volume 773 of *Lecture Notes in Computer Science*, pages 456–479. Springer-Verlag, 1993.

[61] H. Lipmaa. IDEA: A cipher for Multimedia Architectures? In S. Tavares and H. Meijer, editors, *Proceedings of the 5th Workshop on Selected Areas in Cryptography (SAC '98)*, volume 1556 of *Lecture Notes in Computer Science*, pages 248–263. Springer-Verlag, 1998.

[62] G. Lowe. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[63] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-key Protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[64] G. Lowe. Some New Attacks upon Security Protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop.* IEEE Computer Society Press, 1996.

[65] G. Lowe. A Hierarchy of Authentication Specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 31–43. IEEE Computer Society Press, 1997.

[66] G. Lowe. Towards a Completeness Result for Model Checking of Security Protocols. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 96–105. IEEE Computer Society Press, 1998.

[67] G. Lowe and B. Roscoe. Using CSP to Detect Errors in the TMN Protocol. *IEEE Transactions on Software Engineering*, 3(10), 1997.

[68] D. P. Maher. Fault Induction Attacks, Tamper Resistance, and Hostile Reverse Engineering in Perspective. In R. Hirschfeld, editor, *Proceedings of Financial Cryptography '97*, volume 1318 of *Lecture Notes in Computer Science*, pages 109–121. Springer-Verlag, 1997.

[69] W. Mao and C. Boyd. Towards Formal Analysis of Security Protocols. In *Proceedings of the 6th IEEE Computer Security Foundations Workshop*, pages 147–158. IEEE Computer Society Press, 1993.

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

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

[72] Mastercard & VISA. *SET Secure Electronic Transaction Specification: Protocol Definition*, May 1997. `http://www.setco.org/set_specifications.html`.

[73] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publisher, 1993.

[74] C. A. Meadows. Formal Verification of Cryptographic Protocols: A Survey. In *Advances in Cryptology — Asiacrypt 94*, volume 917 of *Lecture Notes in Computer Science*, pages 133–150. Springer-Verlag, 1995.

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

[76] G. Medvinsky and Neuman B. C. NetCash: A Design for Practical Electronic Currency on the Internet. In *Proceedings of the 1st ACM Conference on Communications and Computer Security (CCS'93)*, pages 102–106. ACM Press and Addison Wesley, 1993.

[77] S. P. Miller, J. I. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos Authentication and Authorisation System. Technical Plan Sec. E.2.1, MIT - Project Athena, 1989.

[78] R. Milner. *Communicating and Mobile Systems: the* Π-*Calculus*. Cambridge University Press, 1999.

[79] J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murphi. In *Proceedings of the 16th IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1997.

[80] R. Molva, G. Tsudik, E. Van Herrewegen, and S. Zatti. KryptoKnight Authentication and Key Distribution System. In Y. Deswarte, G. Eizenberg, and J.-J Quisquater, editors, *Proceedings of the 2nd European Symposium on Research in Computer Security (ESORICS'92)*,

volume 648 of *Lecture Notes in Computer Science*, pages 155–174. Springer-Verlag, 1992.

[81] National Bureau of Standards. *Data Encryption Standard*, January 1977. Federal Information Processing Standards Publications, FIPS Pub. 46.

[82] R. M. Needham. Denial of Service. In *Proceedings of the 1st ACM Conference on Communications and Computer Security (CCS'93)*, pages 151–153. ACM Press and Addison Wesley, 1993.

[83] R. M. Needham and Schroeder M. D. Using Encryption for Authentication in large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[84] L. C. Paulson. *Theory for the shared-key Needham-Schroeder protocol.* `http://www4.informatik.tu-muenchen.de/~isabelle/library/HOL/Auth/NS_Shared.html`.

[85] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[86] L. C. Paulson. Mechanized Proofs for a Recursive Authentication Protocol. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 84–95. IEEE Computer Society Press, 1997.

[87] L. C. Paulson. Proving Properties of Security Protocols by Induction. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 70–83. IEEE Computer Society Press, 1997.

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

[89] L. C. Paulson. Inductive Analysis of the Internet protocol TLS. *ACM Transactions on Computer and System Security*, 1999. In press.

[90] L. C. Paulson. Relations between Secrets: two Formal Analyses of the Yahalom Protocol. *Journal of Computer Security*, 2000. In press.

[91] A. Perrig. Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication. In M. Blum and C. H. Lee, editors, *Proceedings of the International Workshop on Cryptographic Techniques & E-Commerce (CrypTEC'99)*, pages 192–202. City University of Hong Kong, 1999.

[92] R. Rivest. Chaffing and Winnowing: Confidentiality without Encryption. *CryptoBytes (RSA Laboratories)*, 4(1):12–17, 1998. `http://theory.lcs.mit.edu/~rivest/chaffing.txt`.

[93] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1976.

[94] B. Roscoe. Model-Checking CSP. *A Classical Mind, Essays in Honour of C. A. R. Hoare*, 1994.

[95] P. Y. A. Ryan. Modelling and Analysis of Security Protocols. Research Proposal, Defence Research Agency, 1994.

[96] P. Y. A. Ryan and S. A. Schneider. An Attack on a Recursive Authentication Protocol: A Cautionary Tale. In *Information Processing Letters 65*. Elsevier Science Publishers (North-Holland), Amsterdam, 1998.

[97] S. Schneider. Verifying Authentication Protocols with CSP. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 3–17. IEEE Computer Society Press, 1997.

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

[99] P. Smith. LUC Public-key Encryption. *Dr. Dobb's Journal*, 18(1):44–49, 90–92, January 1993.

[100] P. Syverson. A Taxonomy of Replay Attacks. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 187–191. IEEE Computer Society Press, 1994.

[101] M. Tatebayashi, N. Matsuzaki, and D. B. Jr. Neuman. Key Distribution Protocol for Digital Mobile Communication Systems. In G. Brassard, editor, *Proceedings of Advances in Cryptography — CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 324–334. Springer-Verlag, 1990.

[102] K. Winter. Methodology for Model Checking ASM: Lessons learned from the FLASH Case Study. In *Proceedings of the ASM2000 Work-*

shop, Lecture Notes in Computer Science. Springer-Verlag, 2000. In Press.

[103] T. Y. C. Woo and S. S. Lam. Authentication for Distributed Systems. *Computer*, 25(1):39–52, 1992.

[104] C. Zarba. *Model Checking the Needham-Schroeder Protocol*, 1998. `http://rodin.Stanford.EDU/case-studies/security`.

[105] J. Zhou, R. H. Deng, and F. Bao. Evolution of Fair Non-repudiation with TTP. In *Proceedings of the 4th Australasian Conference on Information Security and Privacy* , volume 1587 of *Lecture Notes in Computer Science*, pages 258–269. Springer-Verlag, 1998.

[106] J. Zhou and D. Gollmann. A Fair Non-repudiation Protocol. In *Proceedings of the 15th IEEE Symposium on Security and Privacy*, pages 55–61. IEEE Computer Society Press, 1996.