1. Introduction

Communications across modern computer networks should be *secure*, an adjective that embodies multiple properties. For example, one may wonder whether a message just received was altered during its transfer. If not, then the message is said to enjoy *integrity*. Even if a message that is received quotes someone as its creator, he might be a fake one. If not, then the message conveys *authentication* of its creator. Another important property is whether the message that is received was intercepted and understood by others besides its creator and intended receiver. If not, then the message enjoys *confidentiality*. *Peer* generically refers to an endpoint of a remote communication. The peers belong to a set of *agents*.

Devising a complete list of properties to assure secure communications in the context of modern computer networks is matter of current research. Each property implicitly assumes the existence of a malicious agent, the *Spy*, whose aim exactly is to violate the communications profitably. The Spy can overhear messages during transfers, create fake messages and introduce them in the traffic. While history tells us that various forms of security have been important ever since ancient times, significant frauds have been orchestrated in recent years with the help of computers containing inexpensive hardware and software.

Cryptography may help. Used extensively also during the World Wars [97], it is the art of coding and decoding information by means of a *cryptographic key*. A cleartext message is transformed into a ciphertext one using the key (through an operation that is called *encryption*). In the best case, the cleartext can be retrieved from the ciphertext (through an operation that is called *decryption*) if and only if the key is available. In 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 [125], IDEA [105]); otherwise, it is *asymmetric* or *public-key* (RSA [138], LUC [149]). A *digital signature* uses techniques of asymmetric cryptography to confirm the author of a digital message. A *message authentication code*, *MAC* in brief, for a message is another message computed using techniques of symmetric cryptography from the original message and a key that the peers share.

This brief outline of cryptographic terminology signifies that the underlying mathematical foundations are not of specific interest to this book. Additional readings are easy to suggest ([97, 119, 146, 150]).

Steganography [98] may also be used for 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 after seeing the larger one. For example, the low-order pixel bits of a digital image may be changed to the bits of a message to be sent confidentially while the image does not suffer perceptible variations.

A more recent technique aiming for confidentiality and authentication is called *chaffing and winnowing* [137]. It may be considered a form of steganography, but it makes use of MACs. Sender and receiver must initially agree on a secret key by using a key-exchange protocol such as Diffie-Hellmann [73] or Oakley [129]. The sender authenticates his message by computing the correct MAC for it and sending the pair formed by the message and its MAC off to the receiver. The sender also sends *chaff*, namely a large number of other pairs, each made by an intelligible random message and a wrong MAC. Only the intended receiver of the message can discern which pair brings the correct MAC, as he knows the secret key used to compute it. So, he alone 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 the first technique, cryptography; hence, they typically are *cryptographic protocols*. These are sequences of steps that pairs of remote peers must take to subsequently establish a secure communication session between themselves. Each step requires the transmission of a *message*, possibly encrypted, between the peers. Messages include peer names, cryptographic keys, random numbers, timestamps, ciphertexts and concatenations of those components. A security protocol attempts to achieve certain *goals* at the time of its completion, namely a set of security properties.

Experience shows that security protocols often are flawed in the sense that they fail to enforce their claimed goals. A security protocol precisely is a concurrent 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 protocol sessions. By doing so, she can exploit on a session the messages obtained from others. Moreover, the vulnerabilities of current transport protocols let her overhear the messages exchanged by other agents. Using security measures such as cryptography to enforce the protocol goals in this setting is not easy. This claim is supported by the large number of flaws that have been reported. To only mention a few flaws, some affect well-known protocols [13, 106], others can be classified [151]. Another group affects less publicly known banking protocols, whose weaknesses have been exploited by

dishonest employees [12]. Other flaws are due to specific implementations of the cryptographic primitives [143].

These few citations confirm that establishing whether a protocol lives up its promises may be very difficult. This process was only carried out by informal reasoning until the late 1980s. If a protocol claimed to achieve a goal, some researchers studied the protocol in detail and decided whether this was true. At present, informal reasoning still retains its importance:

- it is crucial for grasping the semantics of protocol designs beyond their bare representation as message sequences;
- it may find minor weaknesses or simple flaws in a protocol more quickly than formal reasoning;
- it is fundamental for understanding certain flaws thoroughly;
- it is easier to follow by an inexperienced audience;
- it ultimately helps for developing formal reasoning and understanding formal guarantees.

While informal reasoning was failing to capture serious protocol flaws, the early 1990s saw increasing awareness that formal reasoning can be conducted profitably on abstract protocol models [58, 99, 117]. It can effectively prove a protocol correct in a model or otherwise detect realistic flaws. As we shall see (Chapter 2), some methods of conducting formal reasoning lack expressiveness or automation, others are just too complicated to use on realistic protocols or to suscitate industrial interest.

This book is about the use of the *Inductive Method*, which is supported by the theorem prover *Isabelle*, to formally prove correctness of realistic protocol models. While the foundations of the Inductive Method are due to Paulson [133], our aim exactly is its development to make it capable for real-size protocols. In achieving this aim, we have considerably extended the method, deepened the formal reasoning about protocols and ultimately developed a general principle of prudent protocol analysis. Therefore, this book can be profitably read by at least anyone interested in any of the following:

- understanding the entangled technicalities hidden behind various types of security protocols;
- learning a method of conducting formal analysis of realistic security protocols;
- teaching (verification of) security protocols;
- practicing with the theorem prover Isabelle;
- practicing with a general principle to realistically conduct formal analysis of security protocols.

An excellent companion to the present manuscript is Boyd and Mathuria's recent book on security protocols [55], which eminently discusses a variety of protocols and their underlying philosophy using a precise though informal language. By contrast, our book uses a formal language to systematically

disassemble the protocol features down to the smallest component and bring to light details that might otherwise remain hidden.

The organisation of this chapter is simple. First, the motivation to our work is discussed (§1.1), and our contribution to knowledge is sketched (§1.2). Then, the notation that will be used throughout the book is presented (§1.3), and the remaining chapters are outlined (§1.4).

1.1 Motivation

The foundations of the Inductive Method date back to 1996, and are presented in Chapter 3. This section refers to that initial development stage, when our research and the subject of this book initiated. Hence, the present tense here refers to late 1996. Our motivation is threefold: the development of a young and promising method; the verification of real-size protocols that have never been formally explored in a realistic setting; and the investigation of general principles underlying correct protocols.

To gradually introduce the Inductive Method, here is its main underlying idea: simple mathematical induction suffices to model security protocols and reason about their goals. A key concept is the *trace*, a list of network events occurring while an unbounded 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 represents the formal protocol model. Proofs can be carried out by induction on a generic trace of the model, with mechanical support offered by the theorem prover Isabelle. They establish trace properties representing goals of the underlying protocol.

1.1.1 Developing the Inductive Method

Further testing. A general theory of messages and an extendible formalisation of the Spy are already available in the method. An important feature is that no bound is stated on 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. Also, each agent is allowed to interleave an arbitrary number of protocol sessions.

However, the method has only been applied to a few classical security protocols [131, 132]. To convince ourselves of the practicality of mathematical induction in this context, further case studies are necessary. The security community appears to lament that the size of the existing case studies is not realistic. Hence, we choose to turn our attention to largely deployed protocols and to intrinsically different protocols, such as non-repudiation ones.

Deeper understanding. Other informal criticism of the method derives from difficulties in accepting the concept of trace, considered a "low-level" view of the network traffic, or a structure that is "non-existent" in reality.

A trace can be viewed as a possible *history* of the network events occurring while the protocol is executed. This interpretation may help us understand the key concepts of the method, although it should be verified over additional case studies.

All proofs follow the natural inductive style adopted by humans: verifying that the various protocol steps preserve a certain property. However, proofs may seem "cryptic" and, consequently, their results may be accepted with reluctance. This is often due to streamlined proof scripts, which feature highly automatic proof methods implementing several proof steps. We intend to favour human inspection of proofs by often preserving the linear application of the proof steps.

Additional elements. Many protocols use timestamps to assure freshness of important components such as session keys. The current datatype of messages does not feature timestamps. A related open issue is how to express *freshness* in terms of timestamps.

The reception of protocol messages is not modelled. However, understanding a formal guarantee very often involves some informal reasoning about reception. Thus, it seems desirable to treat this event formally, although it is not obvious whether the existing analyses would be simple to update accordingly.

Another important issue is how to account for e-commerce protocols, which often involve smartcards. How could the cards be modelled, taking into account the risks of cloning? How could their functionalities and interactions with the agents be represented? Along the same lines, non-repudiation protocols are expected to require a non-standard threat model in which no agents trust each other. Its modelling price is not trivial to anticipate.

1.1.2 Verifying the Protocol Goals

Existing guarantees. The Inductive Method already features a general method for proving the goal of confidentiality, but the early literature [131, 132] fails to mention the concept of *viewpoint*. The formal guarantees about the protocols are expressed in terms of theorems established with Isabelle's support. They are useful to the protocol peers only when the peers can verify whether the theorem assumptions 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 guarantee confidentiality to the responder. As an extreme concern, we may wonder whether a guarantee featuring assumptions that are impossible to verify would be of any practical importance.

While the treatment of confidentiality is, as mentioned, satisfactory, that of authentication is not. The latter is an important and complicated goal

that may hold in a hierarchy of forms, as confirmed by Lowe using another formal method [109]. However, the current formalisation of authentication using the Inductive Method fails to express the knowledge of the very message components that authenticate the agents (a satisfactory explanation of the various authentication forms must be deferred until later, §4.6). Investigating how many and which forms our method captures at present is challenging.

Novel guarantees. The well-known goals of integrity, authenticity, key distribution and strong forms of authentication need to be treated formally with the Inductive Method. To illustrate, we observe that even when the initiator is informed that session key confidentiality holds, it is not consequently obvious that the responder shares the same session key or that he means to share it with the initiator. It is not clear at this early stage what and how substantial the extensions necessary to formalise these goals might be. Some of the existing guarantees might have to be reinterpreted.

1.1.3 Investigating the Protocol Principles

One of the ultimate goals of analysing security protocols formally is to derive the general principles that make them secure. Ideally, we would like to have simple rules, adherence to which would be easy to check and would directly guarantee the security goals.

The best-known set of principles of prudent protocol design is due to Abadi and Needham [7]. The main principle is *explicitness*, which prescribes each message to say exactly what it means without ambiguity. Other principles include avoiding unnecessary encryption and synchronising the network clocks when timestamps are used. Unfortunately, these principles are far from ideal, as they are neither sufficient nor always necessary to assure security.

It is interesting to investigate whether the findings obtained using the Inductive Method would support these principles or clarify how relevant they are to the goals of a protocol. Moreover, the deeper study of the goals that we advocated above might unveil new principles not only to design the protocols but also to analyse them realistically.

1.2 Contribution

Our contribution is multifaceted. It is simplest to present it in relation to the motivation of the research.

1.2.1 Inductive Method

The Inductive Method (Chapter 3) turns out to be easily extendible. Timestamps are modelled using a discrete formalisation of time based on the position of each event in a trace. Each history of the network is equipped

with a global clock corresponding to the length of the corresponding trace, and so each trace has a global clock yielding the current time of the trace. All agents refer to it, so the model hides problems of clock synchronisation. A message component is considered fresh in a trace if the time interval between the creation of the component and the current time of the trace is less 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 proofs of correctness of three protocols that make use of timestamps: the BAN Kerberos protocol (Chapter 6), the larger and deployed Kerberos IV (Chapter 7), and finally the more recent Kerberos V (Chapter 9). Although they share the structure of a few messages, each protocol hides peculiar subtleties. Their proof scripts are fairly intuitive because of the limited use of automatic proof methods, which is another of our goals.

New events can be modelled. In particular, introducing message reception makes the specifications more readable and the proofs easier to follow, increasing overall intuitiveness. The existing scripts can be updated pragmatically, with minor effort. 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. Moreover, the reception event allows the formalisation of any agent's knowledge, rather than just the Spy's, in terms of message deducibility.

New elements, such as extra trusted servers or smartcards, are modelled as new types of the language. For smartcards, the interaction with their owners is formalised by additional events (Chapter 10). The agents' knowledge, in particular the Spy's, must be reviewed for two reasons: (i) all long-term secrets are now stored only in the cards; (ii) certain protocols that are based on smartcards assume that the Spy cannot listen to communication between a card and its owner, while other protocols do not make this assumption. We verify the entire Shoup-Rubin protocol (Chapter 11) 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.

Another success is the treatment of *accountability protocols*, which aim at giving peers evidence of each other's participation. They require a fundamental change to the threat model because the Spy no longer is an opponent between two peers who trust each other but, rather, can hide behind any of the peers (Chapter 12). We shall see that this change is not difficult to implement through the complete analyses of a non-repudiation protocol by Zhou and Gollmann and a certified e-mail protocol by Abadi et al. (Chapter 13). Some of these protocols assume the existence of a communication channel secured by a standard protocol such as TLS/SSL [72]. We have found simple formalisations for secure transmission over such channels using specific forms of the available events.

To summarise, our research equips the Inductive Method with all the necessary features to tackle industrial protocols. The subsequent verification of the SET protocol [36, 37, 38] confirms this. In general, the method has become so expressive that the bare statements of the theorems convey most guarantees without considerable informal argument to make on top of them. Therefore, the exposition accompanying each theorem reduces to the very minimum.

1.2.2 Protocol Goals

We find that the argument about any protocol goal can (and must, see the next section) be interpreted from the viewpoint of each peer, although this may not be trivial regardless of the formal method in use (Chapter 2). While this practice provides the human analyser with a better understanding of each protocol step, it also produces formal guarantees that the peers can apply in practice, as we shall see. During this interpretation process, we realise that one assumption of a theorem that has been proved for the shared-key Needham-Schroeder protocol is in fact entirely superfluous. Our study of the protocol goals supports the claim that the goals of authenticity and integrity are equivalent (§4.3), while the corresponding guarantees can be derived from reinterpreting some of the existing theorems (Chapter 4).

Paulson's method 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: it lets the Spy exploit certain session keys in realistic circumstances within their lifetime. Moreover, 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.

Two different definitions of agents' knowledge are developed and variously compared. One may appear to be simpler as it exclusively relies on message creation through trace inspection. The other is based on full message deducibility from the traffic that is sent or received (Chapter 8). The latter requires an explicit formalisation of message reception, and so allows us to formally study the goal of key distribution. We argue that this goal is equivalent to a strong form of authentication (§4.7), as we demonstrate on both BAN Kerberos and Kerberos IV. As for Kerberos V, we shall see that its goals are somewhat equivalent to those of Kerberos IV, although its design calls for alternative proof methods.

Verifying the goals of the protocols that are based on smartcards only requires minor modifications to the existing proof methods. A set of simplification rules must be proved to deal with the new events and the new definition of agents' knowledge. Also the smartcards may 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' smartcards. The proofs suggest a simple fix to the protocol, yielding stronger guarantees against veracious threats.

The goals of accountability protocols can be also analysed by induction. We have developed simple proof methods for the main goals of validity of evidence and of fairness. The former confirms that certain messages truly count as evidence of an agent's participation in the protocol. Fairness is an additional goal, requiring the appropriate evidence to be either available to both peers or to none, so that no one is advantaged. Various forms of both goals can be proved by simply showing that certain events always precede specific others on the protocol traces. This treatment is demonstrated on both the non-repudiation protocol and the certified e-mail protocol.

Admittedly, our proofs were difficult to develop. The analyses of Kerberos IV and Shoup-Rubin, for example, saw certain proofs take up to four man-weeks each to be developed, while the corresponding script was up to 50 Isabelle commands long. Polishing the original scripts often shortens them up to approximately one fifth of their original length, thanks to a moderate use of Isabelle's automatic proof methods, in which subsidiary lemmas can be installed. As mentioned, this must be done with care, for it may affect the resulting intelligibility. However, proof scripts are doomed to change over time as Isabelle evolves, while proof methods will rarely change. Hence, this book concentrates on the general methods rather than on the actual Isabelle scripts, some of which are demonstrated in the appendices.

1.2.3 Protocol Principles

Our research favours the development of protocol principles, namely those meta-rules that contribute to guaranteeing security. However, no principle is found to be sufficient in general.

To be precise, the importance of explicitness is confirmed. The messages that are not explicit about their meaning force the peers to heuristic interpretations 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 Shoup-Rubin, a smartcard one. We stress that studying adherence to the explicitness principle always requires an assessment of the underlying threat model. For example, if a communication is assumed to be preauthenticated, then quoting the peer names may be unnecessary.

Explicitness is also interesting from the proof perspective. We find that, if a message lacks explicitness, then carrying out any proofs about it requires quantifying existentially the exact components that are not sufficiently explicit. The prover needs either to bind them to the assumptions or to conjecture that they exist. It could be argued that mere expertise in theorem

proving can discover lack of explicitness and therefore make up for competency of prudent protocol design.

The verification of Kerberos IV confirms the principle 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, under a realistic threat model. Additional support for the principle derives from the analysis of Kerberos V, which attains similar confidentiality goals and suffers the same attack although it disposes with double encryption.

Our attention to the agents' viewpoints in carrying out formal protocol analyses leads us to the development of a principle of prudent protocol analysis. This seems to be the first time that a principle is spelled out to guide the analysis of protocols rather than directly their design. Secure protocol design of course remains the ultimate aim. But analysis and design are equally important because the former is meant to influence the latter. Only a realistic analysis will contribute to a truly more robust design.

Our principle of prudent protocol analysis is called *goal availability* (Chapter 5). It holds for a protocol, one of its goals and one of its peers if there exist guarantees from the peer's viewpoint that the goal is met. The guarantees must rely on assumptions that the peer is able to verify in practice. However, for each peer we can identify a set of assumptions that are necessary although the peer can never verify them: they form the peer's *minimal trust*. Goal availability tolerates the minimal trust.

Adherence to goal availability may be sufficient to prevent certain attacks or weaknesses either directly (as with Kerberos IV) or indirectly (as with Shoup-Rubin). The attack on Kerberos IV is in fact due to a violation of goal availability where the goal is session key confidentiality for the protocol responder. The weaknesses of Shoup-Rubin are due to the lack of explicitness discovered through the verification of adherence to goal availability. In this light, our principle seems easier to verify than the explicitness principle, whose definition is less constructive.

1.3 Notation

The notational conventions that are used throughout this book are summarised here. Although they are rather standard, digesting them appropriately is very useful to understanding this book thoroughly.

1.3.1 Presenting the Protocols

Getting to grasps with the syntax of messages is fundamental. Fat braces "$\{\!|$" and "$|\!\}$" [133, §2.1] are used to distinguish protocol messages from sets, and also to indicate the encryption operation. They are omitted in the case of

messages whose outermost constructor is concatenation, and in the case of ciphertexts whose body is a single-component message. For example:

- a ciphertext made by encrypting with key K a two-component message consisting of m concatenated with n is indicated as $\{\!|m, n|\!\}_K$;
- a two-component message consisting of m concatenated with n is indicated as m, n;
- a single-component message m encrypted with key K is indicated as m_K.

The security protocols will be presented using what today is a rather 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.

$$\begin{array}{llcl} 1. & A \longrightarrow B & : & A, Na \\ 2. & B \longrightarrow A & : & \{\!|Na, Kab|\!\}_{sK_B^-} \end{array}$$

Fig. 1.1. Example protocol

Figure 1.1 presents an example protocol that helps to demonstrate the notation. The protocol consists of two steps. In the first step, agent A sends agent B the two-component concatenated message formed with her own identity and Na. In the second step, agent B replies to A with a ciphertext obtained by signing with his key sK_B^- the concatenation of Na with another key Kab. It is unnecessary to detail the messages precisely at this stage. We in fact state nothing about the keys and only anticipate that Na and Nb are *nonces*. A nonce is a random "number that is used only once" [126]. In our example protocol, A invents the nonce Na: it has never been used before. Hence, its use tells A, upon reception of the second message, that B's reply is more recent than the instant Na was created. Nonces can also help establish various forms of authentication, as we shall see in this book.

A full understanding of the notation requires a reference to the *threat model* in which all protocols will be studied. The Spy can intercept all messages and prevent their delivery. She can also tamper with them by decomposing concatenated messages and opening up ciphertexts sealed with keys she knows. Then, she can use the learnt message components to form new messages at will by concatenation and encryption. This threat model, which we shall discuss more extensively (§3.9), is due to Dolev and Yao [75], and is a *de facto* standard for protocol analysis at present. In this threat model, a message that is sent is not necessarily ever received, and the receiver of a message is not necessarily its intended recipient. Therefore, a protocol step such as $A \longrightarrow B : A, Na$ signifies that A sends the two-component message to B but says nothing about B's reception of the message.

1.3.2 Naming the Theorems

This section presents a general naming paradigm that we adopt for all theorems throughout this book. The theorem names are identical to those in the proof scripts, which come with the Isabelle repository [33, 34] from the 2006 distribution. We shall see (§3.1) that it is useful to execute the relevant proof scripts interactively while the theorems are discussed throughout the book.

When we terminated a large protocol proof, which perhaps had taken up to weeks of concentration, one of the last things we usually worried about was the choice of the best, most expressive name for the theorem just proved. However, we now realise that a uniform naming system becomes especially important when putting together theorems about distinct protocols, as is the case with this book. It also helps in interpreting the theorems properly. Precisely, we have to face the following three issues.

1. *Theorem names should be correctly expressive.* When we read a theorem name, we would like to grasp its meaning, namely the goal it is trying to express, from its name. To achieve this, it was necessary to go back to all proof scripts and change many original theorem names. For example, the word "trust" was abused by many theorems establishing the originator of a message. Since that word is currently used mostly in relation with trustworthiness of agents, we found that it was inadequate. Authenticity of messages or authentication of agents seemed more appropriate terms. To only give an instance of this evolution of names, Theorem 6.5.2 is now addressed as **BK_Kab_authentic**, but its original name was **A_trusts_Kb2**. All theorem names are now coherent with the goal names that we set below (Chapter 4).
2. *Theorem names should be coherent between various protocols though not identical.* If we establish the same guarantee (say, of confidentiality) for more than one protocol, we would like the relevant theorems to have the same names so that they would favour comparative considerations. However, having two or more theorems named identically in a book may appear contradictory. To set about this issue, we decided to prefix *only in the book* each theorem name with the acronym in capital letters of the protocol name it refers to. In this vein, the prefix **BK_** of the mentioned Theorem 6.5.2 reminds us that it refers to the BAN Kerberos protocol. An analogous guarantee for Kerberos IV is Theorem 7.3.3, whose name **KIV_authK_authentic** correctly identifies the protocol. The different key name addresses the ambiguity only in this case. A more evident example derives from the session key confidentiality guarantees for Kerberos IV, such as Theorem 7.3.14 called **KIV_Confidentiality_B**, and for Kerberos V, such as Theorem 9.3.1 called **KV_Confidentiality_B**.
3. *Theorem names should be coherent between various versions of the same guarantee.* There are often various versions of the same theorem for various reasons. We maintain coherence between the various version names by adding a prefix and/or a suffix as follows.

- Once a theorem is proved, it is often possible to weaken or strengthen its assumptions and derive weaker or stronger facts. We indicate such a variant theorem by adding the **_bis** suffix to the original theorem name. For example, **SR_Outpts_A_Card_form_10_bis** is the name of Theorem 11.3.8. There may also be another variant, in which case the **_ter** suffix is added to the original theorem name.
- The fewer the assumptions made, the easier to grasp a theorem normally turns out to be. This is particularly so with assumptions of key confidentiality, which normally expand into several facts when relaxed. For the sake of presentation, it often becomes preferable to leave such assumptions unrelaxed. The theorem versions where all assumptions are relaxed into elementary facts are indicated by the **_r** suffix. For example, Theorem 6.5.10 is called **BK_A_authenticates_B_r**.
- Once a protocol is thoroughly analysed, we may find it relevant to update its design and repeat the analysis in a separate theory file. The theorem names for updated protocols receive a conventional prefix as described above. For example, Theorem 8.5.1 for our updated Otway-Rees protocol is called **ORB_analz_hard**, the **B** standing for Bella. Also the name of the new theory file is updated accordingly (§3.2). By contrast, the theorem names reflecting minor updates that can coexist within the original theory file are only continued with a **u**. Some of these theorems are mentioned but never presented in this book to limit redundancy.
- When we introduce the Gets event to formally model message reception (§8.2), the existing protocol models must be updated accordingly (in a separate theory file). The prefix of their theorem names is suffixed with a **g**. For example, Theorem 8.4.7 for the Kerberos IV model with message reception is called **KIVg_B_authenticates&keydist_to_A**. Also the name of the new theory file is updated accordingly (§3.2).

1.3.3 Wording the Symbols

Logical statements contain specific symbols with dedicated semantics. We replace most Isabelle symbols with the corresponding English phrases whenever their semantics is obvious and the replacement can improve readability.

- Logical conjunction ($\wedge$) with *and*.
- Logical disjunction ($\vee$) with *or*.
- Logical negation ($\neg$) with *not*.
- Logical equivalence ($\leftrightarrow$) with *if and only if*.
- Disequality ($\neq$) with *is not*.
- Meta-level implication ($[\![\ldots]\!]\Longrightarrow\ldots$) with *if ... then ...*.
- Existential quantification ($\exists$) with *for some*.
- Universal quantification ($\forall$) with *for any*.

By contrast, it is best to preserve other symbols exactly for the sake of readability. Isabelle's graphical interface offers perfect mathematical symbols [160].

- It is convenient to keep logical equality in symbols ($=$) because it can be used to specify variable expansions (abbreviations) among the theorem preconditions, while the variable can be compactly mentioned even repeatedly among the theorem conclusions. Isabelle supports well this form of equational reasoning, as this book confirms.
- Having conducted some experiments of wording set membership ($\in$), we concluded that it is quicker to grasp in symbols — with only one exception. It is preferable to state that *a list contains an element*, which is often needed below, rather than to use the symbol for set membership over the set of elements that the list contains.
- Other set operators, such as union ($\cup$) and inclusion ($\subseteq$), which are rarely used below, are kept in symbols.

Syntax is left completely unaltered when it is quoted for the sake of demonstration — always in figures — such as when the protocol models are presented.

1.4 Contents Outline

This section briefly outlines the chapters of this book.

Chapter 2 reviews some of the main formal methods for analysing security protocols. Such a large variety of methods has been advanced that the chapter cannot present all of them. The importance of interpreting the findings cautiously is emphasised. The difficulties in conducting the analyses do not seem to be related to those in interpreting the findings, as various examples confirm.

Chapter 3 outlines the Inductive Method as it was in late 1996, when our research initiated. The presentation of the method is gradual and informative, as it gives particular attention to the intuition behind each construct. The chapter begins with an introduction to the working environment for the method, namely the theorem prover Isabelle, and terminates with an example of a protocol model.

Chapter 4 formalises in the Inductive Method the most important guarantees for the protocol models. With every protocol that is analysed, those guarantees form the aim of our analysis, and hence the chapter introduces important terminology. Seven groups of guarantees are given, each expressing an important protocol goal, except for a group that helps to validate the protocol models. Proof methods and examples are provided.

Chapter 5 defines our general principle of prudent protocol analysis, goal availability. An abstract version is given first, in order to favour the reader's intuition, while a more detailed version only comes after additional discussion. Then, the related concept of minimal trust is put forward. The principle is demonstrated only on a simple protocol but additional examples are frequent in the subsequent chapters.

Chapter 6 extends the Inductive Method with a treatment of timestamping, which requires both a formalisation of time and a definition of timestamps. We find simple solutions for both issues. Then, the BAN Kerberos protocol can be formally analysed. Finally, a temporal modelling of the accidental loss of session keys is introduced. The protocol model is updated accordingly and all guarantees revisited correspondingly.

Chapter 7 presents the formal analysis of Kerberos IV. The treatment of timestamping and the temporal modelling of accidents is inherited from the previous chapter. The protocol achieves strong goals but it fails to conform to our principle of goal availability in the case of confidentiality of a group of session keys for the responder. This leads to realistic attacks, but a simple fix is introduced and verified.

Chapter 8 introduces the modelling of agents' knowledge in the Inductive Method using two definitions. They are demonstrated on the protocols presented above by adding treatments of the key distribution goal and of a stronger version of authentication, which were impossible before. The two definitions are variously compared and contrasted, offering an argument that can refute a claim previously made by the BAN logic.

Chapter 9 reports on the formal analysis of Kerberos V, the most recent version of the Kerberos protocol. The main difference with the previous version in terms of design is the removal of the use of double encryption. We verify that this difference does not significantly influence the protocol goals, which are analogous to those of the previous version. However, different proof methods become necessary.

Chapter 10 describes a realistic treatment of smartcards in the Inductive Method. Since some protocols explicitly assume that the communication means between the cards and their owners is secure, while others do not, our treatment develops around both options. The Spy is allowed to exploit an unspecified set of smartcards, some through simple theft, others through elaborate tampering ultimately leading to cloning.

Chapter 11 uses the extended method of the previous chapter to analyse the Shoup-Rubin protocol, which makes use of smartcards. The protocol is generally strong, but it is found to violate our goal availability principle for the goal of session key confidentiality. This reveals two important shortcomings of explicitness that affect three goals: confidentiality, authentication and key distribution. A simple fix is introduced and verified.

Chapter 12 extends the Inductive Method to deal with accountability protocols. Non-repudiation and certified e-mail delivery are recognised as forms

of accountability, where the peers get evidence of each other's participation. Abstract formalisations of these novel goals are provided along with appropriate methods to prove them. The concept of second-level protocol, which relies on another security protocol, is advanced.

Chapter 13 uses the extensions introduced in the previous chapter to describe the analyses of two emblematic accountability protocols. They are both studied in terms of validity of the evidence provided to the peers and in terms of its fairness. The threat model appropriate for this group of protocols is used: an honest agent enjoys the protocol goals even when his peer is the Spy. Both protocols appear to achieve their claimed goals.

Chapter 14 concludes the book with a few final remarks, and summarises our contribution through its key concepts. It also briefly advances some lines of possible future work. Finally, it comments on some statistics about file sizes, proof runtimes on two common and inexpensive platforms, and human efforts necessary for the entire book.

There are four appendices to complete the presentation. They present a few relevant fragments of the proof scripts about the main protocols that the book discusses. Such fragments are released with the 2006 distribution of Isabelle [33, 34] (before that distribution appears, they are available with the development snapshot [156]).

Appendix A concerns the Kerberos IV protocol, presenting the guarantees of reliability, session-key compromise, and session-key confidentiality.

Appendix B concerns the Kerberos V protocol, presenting the guarantees of unicity, unicity relying on timestamps, and conjunct key distribution and non-injective agreement.

Appendix C concerns the Shoup-Rubin protocol, presenting the definitions of two important functions and related technical lemmas, and the guarantees of authentication.

Appendix D concerns the Zhou-Gollmann protocol, presenting the guarantees of validity of main and subsidiary evidence, and fairness.