The Inductive Approach to Verifying Cryptographic Protocols

Lawrence C. Paulson
Computer Laboratory
University of Cambridge
Pembroke Street
Cambridge CB2 3QG
England

lcp@cl.cam.ac.uk

6 February 1998

Abstract

Informal arguments that cryptographic protocols are secure can be made rigorous using *inductive definitions*. The approach is based on ordinary predicate calculus and copes with infinite-state systems. Proofs are generated using Isabelle/HOL. The human effort required to analyze a protocol can be as little as a week or two, yielding a proof script that takes a few minutes to run.

Protocols are inductively defined as sets of traces. A trace is a list of communication events, perhaps comprising many interleaved protocol runs. Protocol descriptions incorporate attacks and accidental losses. The model spy knows some private keys and can forge messages using components decrypted from previous traffic. Three protocols are analyzed below: Otway-Rees (which uses shared-key encryption), Needham-Schroeder (which uses public-key encryption), and a recursive protocol [9] (which is of variable length).

One can prove that event ev always precedes event ev' or that property P holds provided X remains secret. Properties can be proved from the viewpoint of the various principals: say, if A receives a final message from B then the session key it conveys is good.

Contents

1 Introduction **1**

2 Overview of the Inductive Method **2**
2.1 Messages . . . 3
2.2 The Operators parts, analz and synth . . . 4
2.3 The Attacker . . . 5
2.4 Modelling a Protocol . . . 5
2.5 Standard Rules . . . 7
2.6 Induction . . . 8
2.7 Regularity Lemmas . . . 8
2.8 Secrecy Theorems . . . 9
2.9 Finding Attacks . . . 10

3 A Mechanized Theory of Messages **11**
3.1 Agents and Messages . . . 11
3.2 Defining parts, analz and synth . . . 12
3.3 Derived Laws Governing the Operators . . . 13
3.4 Rewrite Rules for Symbolic Evaluation . . . 14
3.5 Events and Intruder Knowledge . . . 16

4 A Shared-Key Protocol: Otway-Rees **17**
4.1 Proving Possibility Properties . . . 19
4.2 Proving Forwarding Lemmas . . . 19
4.3 Proving Regularity Lemmas . . . 20
4.4 Proving Unicity Theorems . . . 20
4.5 Proving Secrecy Theorems . . . 21
4.6 Proving the Session Key Secrecy Theorem . . . 22
4.7 Proving Authenticity Guarantees . . . 22
4.8 Proving a Simplified Protocol . . . 24

5 A Public-Key Protocol: Needham-Schroeder **24**
5.1 The Protocol and Lowe's Attack . . . 24
5.2 Modelling the Protocol . . . 25
5.3 Proving Guarantees for A . . . 26
5.4 Proving Guarantees for B . . . 27
5.5 A Glimpse at the Machine Proofs . . . 28
5.6 Analyzing the Strengthened Protocol . . . 30

6 A Recursive Protocol **30**
6.1 The Recursive Authentication Protocol . . . 31
6.2 Deviations from the Protocol . . . 33
6.3 Modelling the Protocol . . . 33
6.4 Modelling the Server . . . 36

6.5 Main Results Proved 37
6.6 Potential Attacks 40

7 Related Work **40**

8 Conclusions **42**

1 Introduction

Cryptographic protocols are intended to let agents communicate securely over an insecure network. An obvious security goal is *secrecy*: a spy cannot read the contents of messages intended for others. Also important is *authenticity*: if a message appears to be from Alice, then Alice sent precisely that message, and any nonces or timestamps within it give a correct indication of its freshness. This paper will not discuss denial of service.

A typical protocol allows A to make contact with B, delivering a key to both parties for their exclusive use. They may involve as few as two messages, but are surprisingly hard to get right. One problem is the combinatorial complexity of the messages that an intruder could generate. A quite different problem is to specify precisely what properties the protocol is intended to achieve. Anderson and Needham's excellent tutorial [3] presents several examples and defines the terminology used below.

Formal methods can be used to analyze security protocols. Two popular approaches are *state exploration* and *belief logics*.

- State exploration methods [40] model the protocol as a finite state system. An exhaustive search checks that all reachable states are safe. Lowe uses a general-purpose model-checker, FDR [24, 26]; the Interrogator [22] is a specialized tool. Attacks are quickly found, but keeping the state space small requires drastic simplifying assumptions.

- Belief logics formalize what an agent may infer from messages received. The original BAN logic [10] allows short, abstract proofs. It has identified some protocol flaws but missed others. New belief logics [28] address some weaknesses of BAN but sacrifice its simplicity.

We can fruitfully borrow from both approaches: from the first, a concrete notion of events, such as A sending X to B; from the second, the idea of deriving guarantees from each message. Protocols are formalized as the set of all possible traces, which are lists of events such as 'A sends X to B.' An agent may extend a trace in any way permitted by the protocol, given what he can see in the current trace. Agents do not know the true sender of a message and may forward items that they cannot read. One agent is an active attacker.

Properties are proved by induction on traces, using the theorem prover Isabelle [32]. Analyzing a new protocol requires several days' effort, while exploring the effects of a change to an existing protocol often takes just a few hours. Laws and proof techniques developed for one protocol are often applicable generally.

The approach is oriented around proving guarantees, but their absence can indicate possible attacks. In this way, I have discovered an attack on the variant of the Otway-Rees protocol suggested by Burrows et al. [10, page

247]. (At the time, I was unaware of Mao and Boyd's earlier attack [28].) Even if no attacks are found, the structure of the proof yields insights into the protocol.

The paper goes on to describe the method, first in overview (§2) and then in some detail (§3). Three protocols are then analyzed. Otway-Rees illustrates the shared-key model (§4); Needham-Schroeder illustrates the public-key model (§5); the recursive authentication protocol [9] demonstrates how to deal with n-way authentication (§6). Related work is discussed (§7) and conclusions given (§8).

2 Overview of the Inductive Method

Informal arguments for a protocol's correctness are conducted in terms of what could or could not happen. Here is a hypothetical dialogue:

> *Salesman.* At the end of a run, only Alice and Bob can possibly know the session key Kab.
>
> *Customer.* What about an eavesdropper?
>
> *Salesman.* He can't read the certificates without Alice or Bob's long-term keys, which he can't get.
>
> *Customer.* Could an attacker trick Bob into accepting a key shared with himself?
>
> *Salesman.* The use of identifying nonces prevents that.

The customer may find such arguments unconvincing, but they can be made rigorous. The necessary formal tool is the *inductive definition* [2]. Each inductive definition lists the possible actions that an agent or system can perform. The corresponding induction rule lets us reason about the consequences of an arbitrary finite sequence of such actions. Induction has long been used to specify the semantics of programming languages [20]; it copes well with nondeterminism. (Plotkin conceived this application of inductive definitions, while Huet pioneered their use in proof tools.)

For security protocols, the model must specify the capabilities of an attacker. Several inductively-defined operators are useful. One (parts) merely returns all the components of a set of messages. Another (analz) models the decryption of past traffic using available keys. Another (synth) models the forging of messages. The attacker is specified—independently of the protocol!—in terms of analz and synth. Algebraic laws governing parts, analz and synth have been proved by induction and are invaluable for reasoning about protocols.

The inductive protocol definition models the behaviour of honest agents faithfully executing protocol steps in the presence of the attacker. It can even model carelessness, such as agents accidentally revealing secrets. The

inherent nondeterminism models the possibility of an agent's being unavailable.

Belief logics allow short proofs; the main reason for mechanizing them [7] is to eliminate human error. In contrast, inductive verification of protocols involves long and detailed proofs. Each safety property is proved by induction over the protocol. Each case considers a state of the system that might be reached by the corresponding protocol step. Simplifying the safety property for that case may reveal a combination of circumstances leading to its violation. Only if all cases are covered has the property been proved.

> *Customer.* What's to stop somebody's tampering with the nonce in step 2 and later sending Alice the wrong certificate?
>
> *Salesman.* Is there somebody less experienced I could talk to?

2.1 Messages

Traditional protocol notation is not ideal for mechanization. Expressing concatenation by a comma, as in A, B, can be ambiguous; enclosing it in braces, as in $\{A, B\}$, invites confusion with a two-element set. The machine syntax uses fat braces to express concatenation: $\{\!|A, B|\!\}$. Informal protocol descriptions omit outer-level braces and indicate encryption by a notation such as $\{\!|Na, Kab|\!\}_{Ka}$.

Individual protocol descriptions rest on a common theory of message analysis. Message items may include

- agent names A, B, ... ;
- nonces Na, Nb, ... ;
- keys Ka, Kb, Kab, ... ;
- compound messages $\{\!|X, X'|\!\}$,
- hashed messages $\mathsf{Hash}\, X$,
- encrypted messages $\mathsf{Crypt}\, K\, X$.

With public-key encryption, K^{-1} is the inverse of key K. The equality $K^{-1} = K$ expresses that K is a symmetric key. The theory assumes $(K^{-1})^{-1} = K$ for all K.

Nonces are of two kinds: those that are guessable and those that are not. Sequence numbers and timestamps can be regarded as guessable, but not 40-byte random strings.

An encrypted message can neither be altered nor read without the appropriate key; different types of components cannot be confused. Including redundancy in message bodies can satisfy these assumptions.

Some published attacks involve accepting a nonce as a key [25] or regarding one component as being two [11]. One could alter the model to admit type confusion attacks, but a little explicitness in protocols [1] can cheaply prevent them.

2.2 The Operators parts, analz and synth

Three operations are defined on possibly infinite sets of messages. Each is defined inductively, as the least set closed under specified extensions. Each extends a set of messages H with other items derivable from H. Typically, H contains an agent's initial knowledge and the history of all messages sent in a trace.

The set $\mathsf{parts}\,H$ is obtained from H by repeatedly adding the components of compound messages and the bodies of encrypted messages. (It does not regard the key K as part of $\mathsf{Crypt}\,K\,X$ unless K is part of X itself.) It represents the set of all components of H that are potentially recoverable, perhaps using additional keys. Proving $X \notin \mathsf{parts}\,H$ establishes that X does not occur in H (except, possibly, in hashed form). Here are two facts proved about parts:

$$\mathsf{Crypt}\,K\,X \in \mathsf{parts}\,H \Longrightarrow X \in \mathsf{parts}\,H$$
$$\mathsf{parts}\,G \cup \mathsf{parts}\,H = \mathsf{parts}(G \cup H).$$

The set $\mathsf{analz}\,H$ is obtained from H by repeatedly adding the components of compound messages and by decrypting messages whose keys are in $\mathsf{analz}\,H$. The set represents the most that could be gleaned from H without breaking ciphers. If $K \notin \mathsf{analz}\,H$, then nobody can learn K by listening to H. Here are some facts proved about analz:

$$\mathsf{Crypt}\,K\,X \in \mathsf{analz}\,H,\ K^{-1} \in \mathsf{analz}\,H \Longrightarrow X \in \mathsf{analz}\,H$$
$$\mathsf{analz}\,G \cup \mathsf{analz}\,H \subseteq \mathsf{analz}(G \cup H)$$
$$\mathsf{analz}\,H \subseteq \mathsf{parts}\,H.$$

The set $\mathsf{synth}\,H$ models the messages a spy could build up from elements of H by repeatedly adding agent names, forming compound messages and encrypting with keys contained in H. Agent names are added because they are publicly known. Nonces and keys are not added because they are unguessable; the spy can only use nonces and keys given in H. Here are two facts proved about synth:

$$X \in \mathsf{synth}\,H,\ K \in H \Longrightarrow \mathsf{Crypt}\,K\,X \in \mathsf{synth}\,H$$
$$K \in \mathsf{synth}\,H \Longrightarrow K \in H.$$

2.3 The Attacker

The enemy observes all traffic in the network—the set H—and sends fraudulent messages drawn from the set $\mathsf{synth}(\mathsf{analz}\,H)$. Interception of messages is modelled indirectly: any message can be ignored.

No protocol should demand perfect competence from all players. If the spy should get hold of somebody's key, communications between other agents should not suffer. The model gives the spy control over an unspecified set of compromised agents; he holds their private keys. Most protocol descriptions include an Oops event to allow accidental loss of session keys.

Our spy is accepted by the others as an honest agent. He may send normal protocol messages using his own long-term secret key, as well as sending fraudulent messages. This combination lets him participate in protocol runs using intercepted keys, thereby impersonating other agents.

The spy is powerful, but he is the same in all protocols. A common body of laws and *tactics* (mechanical proof procedures) is available. A tactic often proves the spy's case of the induction automatically.

2.4 Modelling a Protocol

Most events in a trace have the form $\mathsf{Says}\,A\,B\,X$, which means 'A sends message X to B.' Another possible event is $\mathsf{Notes}\,A\,X$, which means 'A stores X internally.' Other events could be envisaged, such as the replacement of a long-term key. Each agent's state is represented by its initial knowledge (typically, its private key) and what it can scan from the list of events. Apart from the spy, agents only read messages addressed to themselves. The event $\mathsf{Notes}\,A\,X$ is visible to A and, if A is compromised, to the spy.

Consider a variant of the Otway-Rees protocol [10, page 247]:

1. $A \to B : Na, A, B, \{\!|Na, A, B|\!\}_{Ka}$
2. $B \to S : Na, A, B, \{\!|Na, A, B|\!\}_{Ka}, Nb, \{\!|Na, A, B|\!\}_{Kb}$
3. $S \to B : Na, \{\!|Na, Kab|\!\}_{Ka}, \{\!|Nb, Kab|\!\}_{Kb}$
4. $B \to A : Na, \{\!|Na, Kab|\!\}_{Ka}$

Informally, (1) A contacts B, generating Na to identify the run. Then (2) B forwards A's message to the authentication server, adding a nonce of his own. Then (3) S generates a new session key Kab and packages it separately for A and B. Finally, (4) B decrypts his part of message 3, checks that the nonce is that sent previously, and forwards the rest to A, who will similarly compare nonces before accepting Kab.

The protocol steps are modelled as possible extensions of a trace with new events. The server is the constant S, while A and B are variables ranging over all agents, including S and the spy. We transcribe each step in turn:

1. If evs is a trace, Na is a fresh nonce and B is an agent distinct from A and S, then evs may be extended with the event

$$\mathsf{Says}\, A\, B\, \{\!|Na, A, B, \{\!|Na, A, B|\}_{Ka}|\!\}.$$

2. If evs is a trace that has an event of the form

$$\mathsf{Says}\, A'\, B\, \{\!|Na, A, B, X|\!\},$$

and Nb is a fresh nonce and $B \neq \mathsf{S}$, then evs may be extended with the event

$$\mathsf{Says}\, B\, \mathsf{S}\, \{\!|Na, A, B, X, Nb, \{\!|Na, A, B|\}_{Kb}|\!\}.$$

The sender's name is shown as A' and is not used in the new event because B cannot know who really sent the message. The component intended to be encrypted with A's key is shown as X, because B does not attempt to read it.

3. If evs is a trace containing an event of the form

$$\begin{aligned}\mathsf{Says}\, B'\, \mathsf{S}\, \{\!|Na, A, B, &\{\!|Na, A, B|\}_{Ka}, Nb,\\ &\{\!|Na, A, B|\}_{Kb}|\!\}\end{aligned}$$

and Kab is a fresh key and $B \neq \mathsf{S}$, then evs may be extended with the event

$$\mathsf{Says}\, \mathsf{S}\, B\, \{\!|Na, \{\!|Na, Kab|\}_{Ka}, \{\!|Nb, Kab|\}_{Kb}|\!\}.$$

The server too does not know where the message originated, hence the B' above. If he can decrypt the components using the keys of the named agents, revealing items of the right form, then he accepts the message as valid and replies to B.

4. If evs is a trace containing the two events

$$\begin{aligned}&\mathsf{Says}\, B\, \mathsf{S}\, \{\!|Na, A, B, X', Nb, \{\!|Na, A, B|\}_{Kb}|\!\}\\ &\mathsf{Says}\, S'\, B\, \{\!|Na, X, \{\!|Nb, K|\}_{Kb}|\!\}\end{aligned}$$

and $A \neq B$, then evs may be extended with the event

$$\mathsf{Says}\, B\, A\, \{\!|Na, X|\!\}.$$

Agent B receives a message of the expected format, decrypts his portion, checks that Nb agrees with the nonce he previously sent to the server, and forwards component X to A. The sender of the first message is shown as B because B knows if he has sent such a message. The rule does not specify the message from S' to be more recent than that from B; this holds by the freshness of Nb.

There is a fifth, implicit, step, in which A checks her nonce and confirms the session. Implicit steps can be modelled, if necessary. For Otway-Rees, it suffices to prove authenticity of the certificate that A receives in step 4. For TLS [15, 35], the model includes a rule for session confirmation in order to support the resumption of past sessions.

We cannot assume that a message sent in step i will be received. But we can identify the sending of a message in step $i+1$ with the receipt of a satisfactory message in step i. Because the model never forces agents to act, there will be traces in which A sends X to B but B never responds. We may interpret such traces as indicating that X was intercepted, B rejected X, or B was down.

An agent may participate in several protocol runs concurrently; the trace represents his state in all those runs. He may respond to past events, no matter how old they are. He may respond any number of times, or never. If the protocol is safe even under these liberal conditions, then it will remain safe when time-outs and other checks are added. Letting agents respond only to the most recent message would prevent modelling middle-person attacks. Excluding some traces as ill-formed weakens theorems proved about all traces.

2.5 Standard Rules

A protocol description usually requires three additional rules. One is obvious: the empty list, [], is a trace. Two other rules model fake messages and accidents.

If evs is a trace, $X \in \mathsf{synth}(\mathsf{analz}\, H)$ is a fraudulent message and $B \neq \mathsf{Spy}$, then evs may be extended with the event

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

Here H contains all messages in the past trace. It includes the spy's initial state, which holds the long-term keys of an arbitrary set of 'bad' agents. The spy may say anything he plausibly could say and can masquerade as any of the bad agents.

The TLS protocol [15] arrives at session keys by exchanging nonces and applying a pseudo-random-number function. I have modelled TLS [35] by assuming this function to be an arbitrary injection. In the protocol specification, agents apply the random-number function when necessary. The spy has an additional rule that allows him to apply the function to any message items at his disposal. Other protocols in which keys are computed will require an analogous rule.

If evs is a trace and S distributed the session key K in a run involving the nonces Na and Nb, then evs may be extended with the event

$$\mathsf{Notes}\ \mathsf{Spy}\ \{\!|Na, Nb, K|\!\}.$$

This strange-looking rule, the Oops rule, models the loss (by any means) of session keys. We need an assurance that lost keys cannot compromise future runs. The Oops message includes nonces in order to identify the protocol run, distinguishing between recent and past losses.

For some protocols, such as Yahalom, the Oops rule brings hidden properties to light [36]. For others, it is not clear whether Oops can be expressed at all.

2.6 Induction

The specification defines the set of possible traces *inductively*: it is the least set closed under the given rules. To appreciate what this means, it may be helpful to recall that the set $\mathbf{N}$ of natural numbers is inductively defined by the rules $0 \in \mathbf{N}$ and $n \in \mathbf{N} \Longrightarrow \mathsf{Suc}\, n \in \mathbf{N}$.

For reasoning about an inductively defined set, we may use the corresponding induction principle. For the set $\mathbf{N}$, it is the usual mathematical induction: to prove $P(n)$ for each natural number n, prove $P(0)$ and prove $P(x) \Longrightarrow P(\mathsf{Suc}\, x)$ for each $x \in \mathbf{N}$. For the set of traces, the induction principle says that $P(evs)$ holds for each trace evs provided P is preserved under all the rules for creating traces.

We must prove $P[]$ to cover the empty trace. For each of the other rules, we must prove an assertion of the form $P(evs) \Longrightarrow P(ev\#evs)$, where event ev contains the new message. (Here $ev\#evs$ is the trace that extends evs with event ev: new events are added to the front of a trace.) The rule may resemble list induction, but the latter considers all conceivable messages, not just those allowed by the protocol.

A trivial example of induction is to prove that no agent sends a message to himself: no trace contains an event of the form $\mathsf{Says}\, A\, A\, X$. This holds vacuously for the empty trace, and the other rules specify conditions such as $B \neq \mathsf{S}$ to prevent the creation of such events.

2.7 Regularity Lemmas

These lemmas concern occurrences of a particular item X as a possible message component. Such theorems have the form $X \in \mathsf{parts}\, H \longrightarrow \cdots$, where H is the set of all messages available to the spy. These are strong results: they hold in spite of anything that the spy might do.

For most protocols, it is easy to prove that the spy never gets hold of any agent's long-term key, excluding the bad agents. The inductive proof amounts to examining the protocol rules and observing that none of them involve sending long-term keys. The spy cannot send any either because, by the induction hypothesis, he has none at his disposal except those of the bad agents.

Unicity results state that nonces or session keys identify certain messages. Naturally we expect the server never to re-issue session keys, or agents their nonces. If they choose these items to be fresh, then it is straightforward to prove that the key (or nonce) part of a message determines the values of the other parts.

2.8 Secrecy Theorems

Regularity lemmas are easy to prove because they are stated in terms of the parts operator. Secrecy cannot be so expressed; if X is a secret then some agents can see X and others cannot. Secrecy theorems are, instead, stated in terms of analz. Their proofs can be long and difficult, typically splitting into cases on whether or not certain keys are compromised.

A typical result involving analz states that if the spy holds some session keys, he cannot use them to reveal others. It would suffice to prove that nobody sends messages of the form $\mathsf{Crypt}\, Kab\, \{\!|\ldots Kcd\ldots|\!\}$, but this claim is false: the spy can send such messages and make other agents send them. Fortunately, he does not thereby learn new session keys; to work such mischief, he must already possess Kcd.

The discussion above suggests the precise form of the theorem. If K can be obtained with the help of a session key K' and previous traffic, then either $K = K'$ or K can be obtained from the traffic alone. Because some protocol steps introduce new keys, proof by induction seems to require strengthening the formula, generalizing K' to a set of session keys. This is the *session key compromise theorem*.

Proving a theorem of this form is often the hardest task in analyzing a protocol. A huge case analysis often results. While it can be automated, the processor time required seems to be exponential in the number of different keys used for encryption in any single protocol message. A bit of creativity here can yield substantial savings; see §6.4 below. For simple key-exchange protocols, however, essentially the same six-command proof script always seems to work.

The theorem makes explicit something we may have taken for granted: that no agent should use session keys to encrypt other keys (see also Gollmann [17, §2.1]). A generalization of the theorem can be used to prove the secrecy of B's nonce in Yahalom [36].

The *session key secrecy theorem* states that if the server distributes a session key Kab to A and B, then the spy never gets this key. Since the spy is treated in every respect as an honest agent, we may conclude that no other agent gets the key either, even by accident.

The theorem stipulates that A and B are uncompromised and that no Oops message has given the session key to the spy. If we must forbid all Oops messages for Kab, not just those involving the current nonces, then we should consider whether the protocol is vulnerable to a replay attack.

Secrecy properties can usually be proved using six or seven commands. A constant problem in secrecy proofs is being presented with gigantic formulas. We need to discard just the right amount of information and think carefully about how induction formulas are expressed.

2.9 Finding Attacks

Secrecy is necessary but not sufficient for correctness. The server might be distributing the key to the wrong pair of agents. When A receives message 4 of the Otway-Rees protocol, can she be sure it really came from B, who got it from S? For the simplified version of the protocol outlined above (§2.4), the answer is no.

The only secure part of message 4 is its encrypted part, $\{\!|Na, Kab|\!\}_{Ka}$. But it need not have originated as the first encrypted part of message 3. It could as well have originated as the second part, if S received a fraudulent message 2 in which a previous Na had been substituted for Nb.

The machine proof leads us to consider a scenario in which Na is used in two roles. It is then easy to invent an attack. A spy, C, intercepts A's message 1 and records Na. He masquerades first as A (indicated as C_A below), causing the server to issue him a session key Kca and also to package Na with this key. He then masquerades as B.

$$
\begin{array}{ll}
1. & A \to C_B : Na, A, B, \{\!|Na, A, B|\!\}_{Ka} \\
1'. & C \to A : Nc, C, A, \{\!|Nc, C, A|\!\}_{Kc} \\
2'. & A \to C_S : Nc, C, A, \{\!|Nc, C, A|\!\}_{Kc}, Na', \{\!|Nc, C, A|\!\}_{Ka} \\
2''. & C_A \to S : Nc, C, A, \{\!|Nc, C, A|\!\}_{Kc}, Na, \{\!|Nc, C, A|\!\}_{Ka} \\
3'. & S \to C_A : Nc, \{\!|Nc, Kca|\!\}_{Kc}, \{\!|Na, Kca|\!\}_{Ka} \\
4. & C_B \to A : Na, \{\!|Na, Kca|\!\}_{Ka}
\end{array}
$$

Replacing nonce Na' by Na in message $2'$ eventually causes A to accept key K_{ca} as a key for talking with B, because Na is A's original nonce. This attack is more serious than that discovered by Mao and Boyd [28], where the server could detect the repetition of a nonce. It cannot occur in the original version of Otway-Rees, where Nb is encrypted in the second message.

Otway-Rees uses nonces not just to assure freshness, but for binding: to identify the principals [1]. Verifying the binding complicates the formal proofs. One can prove—for the corrected protocol—that Na and Nb uniquely identify the messages they originate in and never coincide. Then we can prove guarantees for both agents: if they receive the expected messages, and the nonces agree, then the server really did distribute the session key to the intended parties.

3 A Mechanized Theory of Messages

The approach has been mechanized using Isabelle/HOL, an instantiation of the generic theorem prover Isabelle [32, 37] to higher-order logic. Isabelle is appropriate because of its support for inductively defined sets and its automatic tools. Some Isabelle syntax appears below in order to convey a feel for how proofs are conducted.

The methodology is tailored to Isabelle and makes heavy use of its classical reasoner [34]. However, it can probably be modified to suit other higher-order logic provers such as PVS [31] or HOL [18]. At a minimum, the prover should provide a simplifier that takes conditional rewrite rules and that can perform automatic case splits for if-then-else expressions. Unless some form of set theory is available, the algebraic laws for parts, analz and synth will be lost. HOL predicates make satisfactory sets, but finite lists do not.

Isabelle/HOL has a polymorphic type system resembling ML's [33]. An item of type agent can never appear where something of type msg is expected. Type inference eliminates the need to specify types in expressions. Laws about lists, sets, etc., are polymorphic; the rewriter uses the appropriate types automatically.

3.1 Agents and Messages

There are three kinds of agents: the server S, the friendly agents, and the spy. Friendly agents have the form Friend i, where i is a natural number. The following declaration specifies type agent to Isabelle. (Note that S is called `Server` and that nat is the type of natural numbers.)

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

A datatype declaration creates a union type, with injections whose ranges are disjoint. It follows that the various kinds of agent are distinct, with $\mathsf{S} \neq \mathsf{Friend}\, i$, $\mathsf{S} \neq \mathsf{Spy}$, $\mathsf{Spy} \neq \mathsf{Friend}\, i$, and moreover $\mathsf{Friend}\, i = \mathsf{Friend}\, j$ only if $i = j$.

The various kinds of message items (discussed above, §2.1) are declared essentially as shown below. Observe the use of type agent and the recursive use of type msg. Not shown are further declarations that make $\{X_1, \ldots X_{n-1}, X_n\}$ abbreviate $\mathsf{MPair}\, X_1\ \ldots\, (\mathsf{MPair}\, X_{n-1}\, X_n)$.

```
datatype msg = Agent agent
             | Number nat       (*guessable*)
             | Nonce nat        (*non-guessable*)
             | Key   key
             | MPair msg msg
             | Hash  msg
             | Crypt key msg
```

Again, the various kinds of message are distinct, with $\mathsf{Agent}\, A \neq \mathsf{Nonce}\, N$ and so forth. The injections Agent, Number, Nonce and Key are simply type

coercions.

Because the datatype creates injections, hashing is collision-free: we have $\mathsf{Hash}\,X = \mathsf{Hash}\,X'$ only if $X = X'$. Encryption is strong. Injectivity yields the law

$$\mathsf{Crypt}\,K\,X = \mathsf{Crypt}\,K'\,X' \Longrightarrow K = K' \wedge X = X'.$$

Moreover, the spy cannot alter an encrypted message without first decrypting it using the relevant key. Exclusive-or violates these assumptions, as does RSA [38] unless redundancy is incorporated. Such forms of encryption could be modelled, but the loss of injectiveness would complicate the theory.

3.2 Defining parts, analz and synth

The operators parts, analz and synth are defined inductively, as are protocols themselves. If H is a set of messages then $\mathsf{parts}\,H$ is the least set including H and closed under projection and decryption. Formally, it is defined to be the least set closed under the following rules.

$$\frac{X \in H}{X \in \mathsf{parts}\,H} \qquad \frac{\mathsf{Crypt}\,K\,X \in \mathsf{parts}\,H}{X \in \mathsf{parts}\,H}$$

$$\frac{\{\!|X, Y|\!\} \in \mathsf{parts}\,H}{X \in \mathsf{parts}\,H} \qquad \frac{\{\!|X, Y|\!\} \in \mathsf{parts}\,H}{Y \in \mathsf{parts}\,H}$$

Similarly, $\mathsf{analz}\,H$ is defined to be the least set including H and closed under projection and decryption by known keys.

$$\frac{X \in H}{X \in \mathsf{analz}\,H} \qquad \frac{\mathsf{Crypt}\,K\,X \in \mathsf{analz}\,H \quad K^{-1} \in \mathsf{analz}\,H}{X \in \mathsf{analz}\,H}$$

$$\frac{\{\!|X, Y|\!\} \in \mathsf{analz}\,H}{X \in \mathsf{analz}\,H} \qquad \frac{\{\!|X, Y|\!\} \in \mathsf{analz}\,H}{Y \in \mathsf{analz}\,H}$$

Finally, $\mathsf{synth}\,H$ is defined to be the least set that includes H, agent names and guessable numbers, and is closed under pairing, hashing and encryption.

$$\mathsf{Agent}\,A \in \mathsf{synth}\,H \qquad \mathsf{Number}\,N \in \mathsf{synth}\,H$$

$$\frac{X \in H}{X \in \mathsf{synth}\,H} \qquad \frac{X \in H}{\mathsf{Hash}\,X \in \mathsf{synth}\,H}$$

$$\frac{X \in \mathsf{synth}\,H \quad Y \in \mathsf{synth}\,H}{\{\!|X, Y|\!\} \in \mathsf{synth}\,H} \qquad \frac{X \in \mathsf{synth}\,H \quad K \in H}{\mathsf{Crypt}\,K\,X \in \mathsf{synth}\,H}$$

To illustrate Isabelle's syntax for such definitions, here is the one for analz.

```
consts  analz   :: msg set => msg set
inductive "analz H"
 intrs
   Inj  "X∈H ⟹ X∈analz H"
   Fst  "{|X,Y|}∈analz H ⟹ X ∈analz H"
   Snd  "{|X,Y|}∈analz H ⟹ Y ∈analz H"
   Decrypt "[| Crypt K X ∈analz H;  Key(invKey K)∈analz H |]
            ⟹ X ∈analz H"
```

Given such a definition, Isabelle defines an appropriate fixedpoint and proves the desired rules. These include the introduction rules (those that constitute the definition itself) as well as case analysis and induction.

The definition of parts does not make X a part of Hash X even though it is a part of Crypt $K\,X$. There is no inconsistency here: for typical protocols, private keys might be included in hashes (where they serve as signatures) but never in encrypted messages. We can prove that uncompromised private keys are not part of any traffic, and use this basic lemma to prove deeper properties.

3.3 Derived Laws Governing the Operators

Section 2.2 presented a few of the laws proved for the operators, but protocol verification requires many more. Let us examine them systematically. All have been mechanically proved from the preceding definitions.

The operators are monotonic: if $G \subseteq H$ then

$$\mathsf{parts}\,G \subseteq \mathsf{parts}\,H \quad \mathsf{analz}\,G \subseteq \mathsf{analz}\,H \quad \mathsf{synth}\,G \subseteq \mathsf{synth}\,H.$$

They are idempotent:

$$\begin{aligned}\mathsf{parts}(\mathsf{parts}\,H) &= \mathsf{parts}\,H\\ \mathsf{analz}(\mathsf{analz}\,H) &= \mathsf{analz}\,H\\ \mathsf{synth}(\mathsf{synth}\,H) &= \mathsf{synth}\,H.\end{aligned}$$

Similarly, we have the equations

$$\mathsf{parts}(\mathsf{analz}\,H) = \mathsf{parts}\,H \quad \mathsf{analz}(\mathsf{parts}\,H) = \mathsf{parts}\,H.$$

Building up, then breaking down, results in two less trivial equations:

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

We have now considered seven of the nine possible combinations involving two of the three operators. The remaining combinations, $\mathsf{synth}(\mathsf{parts}\,H)$ and $\mathsf{synth}(\mathsf{analz}\,H)$, appear to be irreducible. The latter one models the

fraudulent messages that a spy could derive from H. We can still prove laws such as

$$\{\!|X, Y|\!\} \in \mathsf{synth}(\mathsf{analz}\, H) \iff X \in \mathsf{synth}(\mathsf{analz}\, H) \wedge Y \in \mathsf{synth}(\mathsf{analz}\, H).$$

More generally, we can derive a bound on what the enemy can say:

$$\frac{X \in \mathsf{synth}(\mathsf{analz}\, H)}{\mathsf{parts}(\{X\} \cup H) \subseteq \mathsf{synth}(\mathsf{analz}\, H) \cup \mathsf{parts}\, H}$$

H is typically the set of all messages sent during a trace. The rule eliminates the fraudulent message X, yielding an upper bound on $\mathsf{parts}(\{X\} \cup H)$. Typically, $\mathsf{parts}\, H$ will be bounded by an induction hypothesis. There is an analogous rule for analz.[1]

3.4 Rewrite Rules for Symbolic Evaluation

Applying rewrite rules to a term such as

$$\mathsf{parts}\{\{\!|\mathsf{Agent}\ A, \mathsf{Nonce}\ Na|\!\}\}$$

can transform it to the equivalent three-element set

$$\{\{\!|\mathsf{Agent}\ A, \mathsf{Nonce}\ Na|\!\}, \mathsf{Agent}\ A, \mathsf{Nonce}\ Na\}.$$

This form of evaluation can deal with partially specified arguments such as $\{\{\!|\mathsf{Agent}\ A, X|\!\}\}$ and

$$\{\{\!|\mathsf{Agent}\ A, \mathsf{Nonce}\ Na|\!\}\} \cup H.$$

Symbolic evaluation for parts is straightforward. For a protocol step that sends the message X we typically consider a subgoal containing the expression $\mathsf{parts}(\{X\} \cup H)$ or $\mathsf{analz}(\{X\} \cup H)$. The previous section has discussed the case in which X is fraudulent. In other cases, X will be something more specific, such as

$$\begin{array}{l}\{\!|\ \mathsf{Nonce}\ Na, \mathsf{Agent}\ A, \mathsf{Agent}\ B,\\ \quad \mathsf{Crypt}\ Ka\ \{\!|\mathsf{Nonce}\ Na, \mathsf{Agent}\ A, \mathsf{Agent}\ B|\!\}\ |\!\}.\end{array}$$

Now $\mathsf{parts}(\{X\} \cup H)$ expands to a big expression involving all the new elements that are inserted into the set $\mathsf{parts}\, H$, from $\mathsf{Nonce}\ Na$ and $\mathsf{Agent}\ A$ to X itself. The expansion may sound impractical, but a subgoal such as $\mathsf{Key}\ K \notin \mathsf{parts}(\{X\} \cup H)$ simplifies to $\mathsf{Key}\ K \notin \mathsf{parts}\, H$ (for the particular X shown above) because none of the new elements has the form $\mathsf{Key}\ K'$. If this

[1] The Isabelle theories represent the set $\{X\} \cup H$ by $\mathsf{insert}\ X\ H$, and similarly $\{X, Y\} \cup H$ by $\mathsf{insert}\ X\ (\mathsf{insert}\ Y\ H)$, etc.

element were present, then the subgoal would still simplify to a manageable formula, $K \neq K' \conj \mathsf{Key}\, K \notin \mathsf{parts}\, H$.

The rules for symbolic evaluation of parts are fairly obvious. They have straightforward inductive proofs.

$$\begin{aligned}
\mathsf{parts}\, \emptyset &= \emptyset \\
\mathsf{parts}(\{\mathsf{Agent}\, A\} \cup H) &= \{\mathsf{Agent}\, A\} \cup \mathsf{parts}\, H \\
\mathsf{parts}(\{\mathsf{Nonce}\, N\} \cup H) &= \{\mathsf{Nonce}\, N\} \cup \mathsf{parts}\, H \\
\mathsf{parts}(\{\mathsf{Key}\, K\} \cup H) &= \{\mathsf{Key}\, K\} \cup \mathsf{parts}\, H \\
\mathsf{parts}(\{\{\!|X, Y|\!\}\} \cup H) &= \{\{\!|X, Y|\!\}\} \cup \mathsf{parts}(\{X\} \cup \{Y\} \cup H) \\
\mathsf{parts}(\{\mathsf{Hash}\, X\} \cup H) &= \{\mathsf{Hash}\, X\} \cup \mathsf{parts}\, H \\
\mathsf{parts}(\{\mathsf{Crypt}\, K\, X\} \cup H) &= \{\mathsf{Crypt}\, K\, X\} \cup \mathsf{parts}(\{X\} \cup H)
\end{aligned}$$

Symbolic evaluation of analz is more difficult. Let us first define the set of keys that can decrypt messages in H:

$$\mathsf{keysFor}\, H \stackrel{\mathrm{def}}{=} \{K^{-1} \mid \exists X.\ \mathsf{Crypt}\, K\, X \in H\}$$

A key can be pulled through analz if it is not needed for decryption.

$$\frac{K \notin \mathsf{keysFor}(\mathsf{analz}\, H)}{\mathsf{analz}(\{\mathsf{Key}\, K\} \cup H) = \{\mathsf{Key}\, K\} \cup (\mathsf{analz}\, H)}$$

The rewrite rule for encrypted messages involves case analysis on whether or not the matching key is available.

$$\mathsf{analz}(\{\mathsf{Crypt}\, K\, X\} \cup H) = \begin{cases} \{\mathsf{Crypt}\, K\, X\} \cup (\mathsf{analz}(\{X\} \cup H)) & K^{-1} \in \mathsf{analz}\, H \\ \{\mathsf{Crypt}\, K\, X\} \cup (\mathsf{analz}\, H) & \text{otherwise} \end{cases}$$

Nested encryptions give rise to nested if-then-else expressions. Sometimes we know whether the relevant key is secure, but letting automatic tools generate a full case analysis gives us short proof scripts. Impossible cases are removed quickly. Redundant case analyses—those that simplify to 'if P then Q else Q'—can be simplified to Q. The resulting expression might be enormous, but symbolic evaluation at least expresses $\mathsf{analz}(\{X\} \cup H)$ in terms of $\mathsf{analz}\, H$, which should let us invoke the induction hypothesis.

Rewriting by the following rule, which is related to idempotence, simplifies the cases that arise when an agent forwards to another agent some message that is visible in previous traffic.

$$\frac{X \in \mathsf{analz}\, H}{\mathsf{analz}(\{X\} \cup H) = \mathsf{analz}\, H}$$

Symbolic evaluation of synth is obviously impossible: its result is infinite. Fortunately, it is never necessary. Instead, we need to simplify assumptions of the form $X \in \mathsf{synth}\, H$, which arise when considering whether a certain message might be fraudulent. The inductive definition regards nonces and keys as unguessable, giving rise to the implications

$$\mathsf{Nonce}\, N \in \mathsf{synth}\, H \Longrightarrow \mathsf{Nonce}\, N \in H$$
$$\mathsf{Key}\, K \in \mathsf{synth}\, H \Longrightarrow \mathsf{Key}\, K \in H$$

If $\mathsf{Crypt}\, K\, X \in \mathsf{synth}\, H$ then either $\mathsf{Crypt}\, K\, X \in H$ or else $X \in \mathsf{synth}\, H$ and $K \in H$. If we already know $K \notin H$, then the rule tells us that the encrypted message is a replay rather than a forgery. There are similar rules for $\mathsf{Hash}\, X \in \mathsf{synth}\, H$ and $\{\!|X, Y|\!\} \in \mathsf{synth}\, H$.

The facts mentioned in this section are among over 110 theorems that have been proved about parts, analz, synth and keysFor. Most of them are stored in such a way that Isabelle can apply them automatically for simplification. Logically speaking, some of these proofs are complex. They need on average under two commands (tactic invocations) each, thanks to Isabelle's automatic tools. The full proof script, over 210 commands, executes in under 45 seconds.

3.5 Events and Intruder Knowledge

A trace is a list of events, each of the form $\mathsf{Says}\, A\, B\, X$ or $\mathsf{Notes}\, A\, X$. Isabelle/HOL provides lists, while events are trivial to declare as a datatype.

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

Otway-Rees assumes a symmetric-key environment. Every agent A has a long-term key, $\mathsf{shrK}\, A$, shared with the server. The spy has such a key (shrK Spy) and there is even the redundant shrK S. Function initState specifies agents' initial knowledge. The spy knows the long-term keys of the agents in the set bad.

$$\mathsf{initState}\, \mathsf{S} \stackrel{\mathrm{def}}{=} \text{all long-term keys}$$
$$\mathsf{initState}(\mathsf{Friend}\, i) \stackrel{\mathrm{def}}{=} \{\mathsf{Key}(\mathsf{shrK}(\mathsf{Friend}\, i))\}$$
$$\mathsf{initState}\, \mathsf{Spy} \stackrel{\mathrm{def}}{=} \{\mathsf{Key}(\mathsf{shrK}(A)) \mid A \in \mathsf{bad}\}$$

The function spies models the set of messages the spy can see in a trace. He sees all messages sent across the network. He even sees the internal notes of the bad agents, who can be regarded as being under his control. From the empty trace, he sees only his initial state. Recall that $ev\#evs$ is the list

consisting of ev prefixed to the list evs.

$$\mathsf{spies}\,[] \stackrel{\text{def}}{=} \mathsf{initState}\,\mathsf{Spy}$$

$$\mathsf{spies}\,((\mathsf{Says}\,A\,B\,X)\,\#\,evs) \stackrel{\text{def}}{=} \{X\} \cup \mathsf{spies}\,evs$$

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

The function spies describes the spy's view of traffic in order to formalize message spoofing. For other agents, the formal protocol rules mention previous messages directly.

The set $\mathsf{used}\,evs$ formalizes the notion of freshness. The set includes $\mathsf{parts}(\mathsf{spies}\,evs)$ as well as the parts of all messages held privately by any agent. For example, if $\mathsf{Key}\,K \notin \mathsf{used}\,evs$, then K is fresh (in evs) and differs from all long-term keys.

$$\mathsf{used}\,[] \stackrel{\text{def}}{=} \bigcup B.\ \mathsf{parts}(\mathsf{initState}\,B)$$

$$\mathsf{used}\,((\mathsf{Says}\,A\,B\,X)\,\#\,evs) \stackrel{\text{def}}{=} \mathsf{parts}\{X\} \cup \mathsf{used}\,evs$$

$$\mathsf{used}\,((\mathsf{Notes}\,A\,X)\,\#\,evs) \stackrel{\text{def}}{=} \mathsf{parts}\{X\} \cup \mathsf{used}\,evs$$

4 A Shared-Key Protocol: Otway-Rees

Section 2.4 discussed the modelling of a protocol informally, though in detail. Now, let us consider the specification supplied to the theorem prover (Fig. 1).

The identifiers at the far left name the rules: Nil for the empty trace, Fake for fraudulent messages, OR1–4 for protocol steps, and Oops for the accidental loss of a session key. The set of traces is the constant otway.

The Nil rule is trivial, so let us examine Fake. The condition $evs \in \mathsf{otway}$ states that evs is an existing trace. Now

$$X \in \mathsf{synth}(\mathsf{analz}(\mathsf{spies}\,evs))$$

denotes any message that could be forged from what the spy could decrypt from the trace; recall that he holds the bad agents' private keys. The spy can send forged messages to any other agent B, including the server. All rules have additional conditions, here $B \neq \mathsf{Spy}$, to ensure that agents send no messages to themselves; this trivial fact eliminates some impossible cases in proofs.

Rule OR1 formalizes step 1 of Otway-Rees. List $evs1$ is the current trace. (Calling it $evs1$ instead of simply evs tells the user which subgoals have arisen from this rule during an inductive proof, even after case-splitting,

```
Nil  [] ∈ otway

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

OR1 [| evs1 ∈ otway;  A ≠ B;  B ≠ Server;  Nonce NA ∉ used evs1 |]
     ⟹ Says A B {|Nonce NA, Agent A, Agent B,
                   Crypt (shrK A) {|Nonce NA, Agent A, Agent B|} |}
          # evs1  ∈ otway

OR2 [| evs2 ∈ otway;  B ≠ Server;  Nonce NB ∉ used evs2;
        Says A' B {|Nonce NA, Agent A, Agent B, X|} ∈ set evs2 |]
     ⟹ Says B Server
            {|Nonce NA, Agent A, Agent B, X, Nonce NB,
              Crypt (shrK B) {|Nonce NA, Agent A, Agent B|}|}
          # evs2  ∈ otway

OR3 [| evs3 ∈ otway;  B ≠ Server;  Key KAB ∉ used evs3;
        Says B' Server
             {|Nonce NA, Agent A, Agent B,
               Crypt (shrK A) {|Nonce NA, Agent A, Agent B|},
               Nonce NB,
               Crypt (shrK B) {|Nonce NA, Agent A, Agent B|}|}
          ∈ set evs3 |]
     ⟹ Says Server B
            {|Nonce NA,
              Crypt (shrK A) {|Nonce NA, Key KAB|},
              Crypt (shrK B) {|Nonce NB, Key KAB|}|}
          # evs3  ∈ otway

OR4 [| evs4 ∈ otway;  A ≠ B;
        Says B Server {|Nonce NA, Agent A, Agent B, X', Nonce NB,
                        Crypt (shrK B) {|Nonce NA, Agent A, Agent B|}|}
          ∈ set evs4;
        Says S' B {|Nonce NA, X, Crypt (shrK B) {|Nonce NB, Key K|}|}
          ∈ set evs4 |]
     ⟹ Says B A {|Nonce NA, X|} # evs4  ∈ otway

Oops [| evso ∈ otway;  B ≠ Spy;
        Says Server B {|Nonce NA, X, Crypt (shrK B) {|Nonce NB, Key K|}|}
          ∈ set evso |]
     ⟹ Notes Spy {|Nonce NA, Nonce NB, Key K|} # evso  ∈ otway
```

Figure 1: Specifying the Otway-Rees Protocol

etc.) The nonce Na must be fresh: not contained in $\mathsf{used}\, evs1$. An agent has no sure means of generating fresh nonces, but can do so with a high probability by choosing enough random bytes.

In rule OR2, $\mathsf{set}\, evs2$ denotes the set of all events, stripped of their temporal order. Agent B responds to a past message, no matter how old it is. We could restrict the rule to ensure that B never responds to a given message more than once. Current proofs do not require this restriction, however, and it might prevent the detection of replay attacks.

There is nothing else in the rules that was not already discussed in §2.4 above. Translating informal protocol notation into Isabelle format is perhaps sufficiently straightforward to be automated.

4.1 Proving Possibility Properties

The first theorems to prove of any protocol description are some *possibility properties*. These do not assure liveness, merely that message formats agree from one step to the next. We cannot prove that anything must happen; agents are never forced to act. But if the protocol can never proceed from the first message to the last, then it must have been transcribed incorrectly.

Here is a possibility property for Otway-Rees. For all agents A and B, distinct from themselves and from the server, there is a key K, nonce N and a trace such that the final message $B \to A : Na, \{\!|Na, Kab|\!\}_{Ka}$ is sent. This theorem is proved by joining up the protocol rules in order and showing that all their preconditions can be met.

4.2 Proving Forwarding Lemmas

Some results are proved for reasoning about steps in which an agent forwards an unknown item. Here is a rule for OR2:

$$\frac{\mathsf{Says}\, A'\, B\, \{\!|N, \mathsf{Agent}\, A, \mathsf{Agent}\, B, X|\!\} \in \mathsf{set}\, evs}{X \in \mathsf{analz}(\mathsf{spies}\, evs)}$$

The proof is trivial. The spy sees the whole of the message; since X is transmitted in clear, analz will find it. The spy can learn nothing new by seeing X again when B responds to this message.

Sometimes the forwarding party removes a layer of encryption, perhaps revealing something to the spy. Then the forwarding lemma is weaker: it is stated using parts instead of analz, and is useful only for those theorems ('regularity lemmas') that can be stated using parts. Otway-Rees has no nested encryption, but the Oops rule removes a layer of encryption: it takes K from the server's message and gives it to the spy. Its forwarding lemma states that this act does not add new keys to $\mathsf{parts}(\mathsf{spies}\, evs)$.

$$\frac{\mathsf{Says}\, \mathsf{S}\, B\, \{\!|Na, X, \mathsf{Crypt}\ K'\, \{\!|Nb, K|\!\}|\!\} \in \mathsf{set}\, evs}{K \in \mathsf{parts}(\mathsf{spies}\, evs)}$$

4.3 Proving Regularity Lemmas

Statements of the form $X \in \mathsf{parts}(\mathsf{spies}\, evs) \longrightarrow \cdots$ impose conditions on the appearance of X in any message. Many such lemmas can be proved in the same way.

1. Apply induction, generating cases for each protocol step and Nil, Fake, Oops.
2. For each step that forwards part of a message, apply the corresponding forwarding lemma, using $\mathsf{analz}\, H \subseteq \mathsf{parts}\, H$ if needed to express the conclusion in terms of parts.
3. Prove the trivial Nil case using a standard automatic tactic.
4. Simplify all remaining cases.

In Isabelle (or any programmable tool), the user can define a tactic to perform these tasks and return any remaining subgoals. Usually, the Fake case can then be proved automatically.

A basic regularity law states that secret keys remain secret. If $evs \in \mathsf{otway}$ (meaning, evs is a trace) then

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

Two commands generate the proof.

4.4 Proving Unicity Theorems

Fresh session keys and nonces uniquely identify their message of origin. But we must exclude the possibility of spoof messages, and this can be done in two different ways. In the case of session keys, a typical formulation refers to an event and names the server as the sender (for $evs \in \mathsf{otway}$):

$$\begin{aligned}\exists B'\ Na'\ Nb'\ X'.\quad \forall B\ Na\ Nb\ X.\\ \mathsf{Says\,S}\ B\ \{\!|Na, X, \mathsf{Crypt}(\mathsf{shrK}\, B)\{\!|Nb, K|\!\}|\!\} \in \mathsf{set}\, evs\\ \longrightarrow B = B' \wedge Na = Na' \wedge Nb = Nb' \wedge X = X'.\end{aligned}$$

The free occurrence of K in the event uniquely determines the other four components shown. To apply such a theorem requires proof that the message in question really originated with the server.

An alternative formulation, here for nonces, presumes the existence of a message encrypted with a secure key:

$$\begin{aligned}\exists B'.\forall B.\ \mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|Na, \mathsf{Agent}\, A, \mathsf{Agent}\, B|\!\} \in \mathsf{parts}(\mathsf{spies}\, evs)\\ \longrightarrow B = B'.\end{aligned}$$

Here evs is some trace and, crucially, $A \notin \mathsf{bad}$. The spy could not have performed the encryption because he lacks A's key. The free occurrence of Na in the message determines the identity of B.

As in the BAN logic, we obtain guarantees from encryption by keys known to be secret. However, such guarantees are not built into the logic: they are proved. Both formulations of unicity may be regarded as regularity lemmas. Their proofs are not hard to generate.

4.5 Proving Secrecy Theorems

Section 2.8 discussed the session key compromise theorem. If K can be obtained from a set of session keys and messages, then either it is one of those keys, or it can be obtained from the messages alone. The theorem is formulated as follows, for an arbitrary trace evs ($evs \in \mathsf{otway}$).

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

Here $\mathcal{K}$ is an arbitrary set of session keys, not necessarily present in the trace. The right hand side of the equivalence is a simplification of

$$K \in \mathcal{K} \cup \mathsf{analz}(\mathsf{spies}\, evs)$$

Replacing analz by parts, which distributes over union, would render the theorem trivial. The right-to-left direction is trivial anyway.

To prove such a theorem can be a daunting task. However, there are techniques that make proving secrecy theorems almost routine.

1. Apply induction.

2. For each step that forwards part of a message, apply the corresponding forwarding lemma, if its conclusion is expressed in terms of analz.

3. Simplify all cases, using rewrite rules to evaluate analz symbolically: pulling out agent names, nonces and compound messages and performing automatic case splits on encrypted messages.

The Fake case usually survives, but it can be proved by a standard argument involving the properties of synth and analz. This argument can be programmed as a tactic, which works for all protocols investigated. For the session key compromise theorem, no further effort is needed. Other secrecy theorems require a detailed argument. Chief among these is proving that nonce Nb of the Yahalom protocol [10] remains secret, which requires establishing a correspondence between nonces and keys [36].

4.6 Proving the Session Key Secrecy Theorem

This theorem states that the protocol is correct from the server's viewpoint. Let $evs \in \mathsf{otway}$ and A, $B \notin \mathsf{bad}$. Suppose that the server issues key K to A and B:

$$\mathsf{Says\,S}\,B\,\{\!|Na, \mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|Na, K|\!\}, \\ \mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|Nb, K|\!\}|\!\} \in \mathsf{set}\,evs$$

Suppose also that the key is not lost in an Oops event involving the same nonces:

$$\mathsf{Notes\,Spy}\,\{\!|Na, Nb, K|\!\} \notin \mathsf{set}\,evs$$

Then we have $K \notin \mathsf{analz}(\mathsf{spies}\,evs)$; the key is never available to the spy.

This secrecy theorem is slightly harder to prove than the previous one. In the step 3 case, there are two possibilities. If the new message is the very one mentioned in the theorem statement then the session key is not fresh, contradiction; otherwise, the induction hypothesis yields the needed result. Isabelle can prove the step 3 case automatically. The Oops case is also nontrivial; showing that any Oops message involving K must also involve Na and Nb requires unicity of session keys, a theorem discussed in the previous section. The full proof script consists of seven commands and executes in eight seconds, generating a proof of over 4000 steps.[2]

4.7 Proving Authenticity Guarantees

The session key secrecy theorem described above is worthless on its own. It holds of a protocol variant that can be attacked (§2.9). In the correct protocol, if A or B receive the expected nonce, then the server has sent message 3 in precisely the right form. Agents need guarantees (subject to conditions they can check) confirming that their certificates are authentic. Proving such guarantees for A and B completes the security argument, via an appeal to the session key secrecy theorem.

The correct protocol differs in message 2, which now encrypts Nb:

1. $A \to B : Na, A, B, \{\!|Na, A, B|\!\}_{Ka}$
2. $B \to S : Na, A, B, \{\!|Na, A, B|\!\}_{Ka}, \{\!|Na, Nb, A, B|\!\}_{Kb}$
3. $S \to B : Na, \{\!|Na, Kab|\!\}_{Ka}, \{\!|Nb, Kab|\!\}_{Kb}$
4. $B \to A : Na, \{\!|Na, Kab|\!\}_{Ka}$

After receiving the step 3 message, B can inspect the certificate that is encrypted with his key, but not the one he forwards to A.

[2] All runtimes were measured on a 300MHz Pentium II. A human could probably generate a much shorter proof by omitting irrelevant steps.

B's guarantee states that if a trace contains an event of the form

$$\mathsf{Says}\, S'\, B\, \{\!|Na, X, \mathsf{Crypt}(\mathsf{shrK}\, B)\{\!|Nb, \mathsf{Key}\, K|\!\}|\!\}$$

and if B is uncompromised and has previously sent message 2,

$$\begin{aligned}\mathsf{Says}\, B\, \mathsf{S}\, \{\!|&Na, \mathsf{Agent}\, A, \mathsf{Agent}\, B, X',\\ &\mathsf{Crypt}(\mathsf{shrK}\, B)\{\!|Na, Nb, \mathsf{Agent}\, A, \mathsf{Agent}\, B|\!\}|\!\}\end{aligned}$$

then the server has sent a correct instance of step 3. The theorem does not establish $S' = \mathsf{S}$ or even that the X component is correct: the message may have been tampered with. But the session key secrecy theorem can be applied. Checking his nonce assures B that K is a good key for talking to A, subject to the conditions of the secrecy theorem.

B's guarantee follows from a lemma proved by induction. It resembles a regularity lemma. Its main premise is that B's certificate has appeared,

$$\mathsf{Crypt}(\mathsf{shrK}\, B)\{\!|Nb, \mathsf{Key}\, K|\!\} \in \mathsf{parts}(\mathsf{spies}\, evs),$$

with other premises and conclusion as in the guarantee itself. Its proof is complex, requiring several subsidiary lemmas:

- If the encrypted part of message 2 appears, then a suitable version of message 2 was actually sent.
- The nonce Nb uniquely identifies the other components of message 2's encrypted part. This was discussed above (§4.4).
- A nonce cannot be used both as Na and as Nb in two protocol runs. If $A \notin \mathsf{bad}$ then the elements

  $$\begin{aligned}&\mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|Na, \mathsf{Agent}\, A, \mathsf{Agent}\, B|\!\}\\ &\mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|Na', Na, \mathsf{Agent}\, A', \mathsf{Agent}\, A|\!\}\end{aligned}$$

  cannot both be in $\mathsf{parts}(\mathsf{spies}\, evs)$.

The proof complexity arises from the use of nonces for binding and because the two encrypted messages in step 3 have identical formats.

Now consider what A (if uncompromised) can safely conclude upon receiving message 4. If a trace contains a message of the form

$$\mathsf{Says}\, B'\, A\, \{\!|Na, \mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|Na, \mathsf{Key}\, K|\!\}|\!\}$$

and if A recalls sending message 1,

$$\begin{aligned}\mathsf{Says}\, A\, B\, \{\!|&Na, \mathsf{Agent}\, A, \mathsf{Agent}\, B,\\ &\mathsf{Crypt}(\mathsf{shrK}\, A)\{\!|Na, \mathsf{Agent}\, A, \mathsf{Agent}\, B|\!\}|\!\}\end{aligned}$$

then the server has sent a message of the correct form, for some Nb. There are many similarities, in both statement and proof, with B's guarantee. A message, purportedly from B, is considered as A would see it. Nonces are compared with those from another message sent from A to B. The proof again requires our proving by induction a lemma whose main premise is

$$\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|Na, \mathsf{Key}\,K|\!\} \in \mathsf{parts}(\mathsf{spies}\,evs),$$

with a detailed consideration of how nonces can be used.

4.8 Proving a Simplified Protocol

Abadi and Needham [1] suggest simplifying Otway-Rees by eliminating the encryption in the first two messages. Nonces serve only for freshness, not for binding. Message 3 explicitly names the intended recipients.

1. $A \to B : A, B, Na$
2. $B \to S : A, B, Na, Nb$
3. $S \to B : Na, \{\!|Na, A, B, Kab|\!\}_{Ka}, \{\!|Nb, A, B, Kab|\!\}_{Kb}$
4. $B \to A : Na, \{\!|Na, A, B, Kab|\!\}_{Ka}$

The authors claim [1, page 11], 'The protocol is not only more efficient but conceptually simpler after this modification.' The machine proofs support their claims. The vital guarantees to B and A, from the last two messages, become almost trivial to prove. Nonces do not need to be unique and no facts need to be proved about them. The new proof script is smaller and runs faster.[3]

The new protocol is slightly weaker than the original. The lack of encryption in message 2 allows an intruder to masquerade as B, though without learning the session key. The original Otway-Rees protocol assures A that B is present (I have proved this using Isabelle), but the new protocol does not. However, the original version never assured B that A was present; anybody could replay message 1, as Burrows et al. have noted [10, page 247].

5 A Public-Key Protocol: Needham-Schroeder

Needham-Schroeder is the obvious choice for demonstrating the inductive method on public-key protocols. Many researchers have investigated it, and Lowe has discovered a subtle flaw [24].

5.1 The Protocol and Lowe's Attack

The full Needham-Schroeder protocol consists of seven steps, four of which are devoted to distributing public keys. Burrows et al. [10] identified a flaw

[3] From 82 to 40 seconds, and from 88 proof commands to 53.

in this part of the protocol: there was no guarantee that the public keys were fresh. Assuming public keys to be universally known reduces the protocol to three steps:

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

Message 2 assures A of B's presence, since only B could have decrypted $\{\!|Na, A|\!\}_{Kb}$ to extract the freshly-invented nonce Na. Similarly, message 3 assures B of A's presence. Burrows et al. claimed a further property, namely that Na and Nb become known only to A and B. (Such shared secrets might be used to compute a session key.) Lowe refuted this claim, noting that if A ran the protocol with an enemy C, then C could start a new run with any agent B, masquerading as A [24].

One might argue that this is no attack at all. An agent who is careless enough to talk to the enemy cannot expect any guarantees. The mechanized analysis presented below reveals that the protocol's guarantees for A are adequate. However, those for B are not: they rely upon A's being careful, which is a stronger assumption than mere honesty. Moreover, the attack can also occur if A talks to an honest agent whose private key has been compromised. Lowe suggests a simple fix that provides good guarantees for both A and B.

5.2 Modelling the Protocol

In the public-key model, an agent A has a public key $\mathsf{pubK}\,A$, known to all agents, and a private key $\mathsf{priK}\,A$. The spy knows the bad agents' private keys. No private key coincides with any public key. In other respects, the model resembles the shared-key one described above (§3.5).

Let us start with the original, flawed, Needham-Schroeder. Figure 2 presents the inductive definition. There are five rules: three for the protocol steps and two standard ones, identical to those in Fig. 1. There is no Oops message because the protocol does not distribute session keys. However, one could ask—as has Meadows [29]—what might happen if one of the nonces is compromised.

More precisely, the protocol steps are as follows:

1. If, in the current trace, Na is a fresh nonce and B is an agent distinct from A, then we may add the event

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

2. If the current trace contains an event of the form

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

```
Nil  [] ∈ ns_public

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

NS1  [| evs1 ∈ ns_public;  A≠B;  Nonce NA ∉ used evs1 |]
     ⟹ Says A B (Crypt (pubK B) {|Nonce NA, Agent A|})
          # evs1  ∈  ns_public

NS2  [| evs2 ∈ ns_public;  A≠B;  Nonce NB ∉ used evs2;
        Says A' B (Crypt (pubK B) {|Nonce NA, Agent A|}) ∈ set evs2 |]
     ⟹ Says B A (Crypt (pubK A) {|Nonce NA, Nonce NB|})
          # evs2  ∈  ns_public

NS3  [| evs3 ∈ ns_public;
        Says A  B (Crypt (pubK B) {|Nonce NA, Agent A|}) ∈ set evs3;
        Says B' A (Crypt (pubK A) {|Nonce NA, Nonce NB|}) ∈ set evs3 |]
     ⟹ Says A B (Crypt (pubK B) (Nonce NB))
          # evs3  ∈  ns_public
```

Figure 2: Specifying the Needham-Schroeder Protocol

and Nb is a fresh nonce and $A \neq B$, then we may add the event

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

Writing the sender as A' means that B does not know who sent the message.

3. If the current trace contains the two events

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

then we may add the event

$$\mathsf{Says}\, A\, B\, (\mathsf{Crypt}(\mathsf{pubK}\, B)\{\!|Nb|\!\}).$$

A decrypts the message and checks that Na agrees with the nonce she previously sent to B. She replies to B's challenge by sending back Nb.

As mentioned in §2.4, we could model the implicit fourth step, in which B inspects the message arriving from A. But it suffices to prove theorems stating what B can infer from such an inspection, such as that A is present.

5.3 Proving Guarantees for A

The guarantees for A are that her nonce remains secret—from the spy—and that B is present. The latter follows from the former, for if the spy does

not know Na then he could not have sent message 2. The proofs require, as lemmas, unicity properties for Na saying that Na is only used once.

- No value is ever used both as Na and as Nb, even in separate runs.
- In any message of the form $\mathsf{Crypt}(\mathsf{pubK}\,B)\{\!|Na, A|\!\}$, the value of nonce Na uniquely determines the agents A and B, over all traffic.

Both lemmas assume Na to be secret and form part of an inductive proof that Na really is secret. They hold because honest agents are specified to choose unpredictable nonces with a negligible probability of collision.

The guarantee for A after step 2 is that the message indeed originated with B, provided it contains the expected nonce. The guarantee is consistent with Lowe's attack because, as always, it considers runs between two uncompromised principals. If A runs the protocol with the spy then her guarantee is void. Lowe himself found no problem with the protocol from A's viewpoint [24, §3.2]; his attack concerns the guarantee for B.

5.4 Proving Guarantees for B

The situation as seen by B is almost symmetrical to that seen by A. Proving by induction that Nb remains secret would authenticate A. Most of the Isabelle proof scripts for A's theorems also work for B with trivial alterations. It is easy to prove that, if Nb is secret, then its value in any message of the form $\mathsf{Crypt}(\mathsf{pubK}\,A)\{\!|Na, Nb|\!\}$ uniquely determines A and Na.

Unfortunately, Nb does not remain secret. The attempt to prove its secrecy fails, leaving a subgoal that contains (as a past event) A's sending message 1 to a compromised agent. The subgoal describes a consistent set of circumstances: Lowe's attack. Details appear in §5.5 below.

Weaker properties can be proved. If A never sends Nb to anybody in step 3 of the protocol, then Nb remains secret. In consequence, if B receives Nb in step 3 then A has sent it, and is therefore present. However, A may have sent it to anybody.

The proof follows the usual argument (based on A's proofs), but assumes that A says no messages of the form $\mathsf{Crypt}(\mathsf{pubK}\,C)Nb$ for any C. With this additional assumption, Nb does remain secret; it then follows that if B sends $\mathsf{Crypt}(\mathsf{pubK}\,A)\{\!|Na, Nb|\!\}$ as step 2 and receives $\mathsf{Crypt}(\mathsf{pubK}\,B)Nb$, then this reply came from A. Since this conclusion contradicts the assumption, B cannot receive $\mathsf{Crypt}(\mathsf{pubK}\,B)Nb$.

The result above has the form $\neg Q$ implies $\neg P$, which is equivalent to P implies Q. If B does receive $\mathsf{Crypt}(\mathsf{pubK}\,B)Nb$, then A has indeed sent the message $\mathsf{Crypt}(\mathsf{pubK}\,C)Nb$ for some C.

This example suggests a general strategy to prove that decrypting a message of the form $\mathsf{Crypt}(\mathsf{pubK}\,A)X$ indicates A's presence. Prove that if A never performs the step in which that message is decrypted, then some

```
1  [| Crypt (pubK B) {|Nonce NA, Agent A|} ∈ parts(spies evs);
      Nonce NA ∉ analz(spies evs);
      evs ∈ ns_public |]
   ⟹ Crypt (pubK C) {|NA', Nonce NA|} ∉ parts(spies evs)

2  [| Nonce NA ∉ analz(spies evs);  evs ∈ ns_public |]
   ⟹ ∃A' B'. ∀A B.
         Crypt (pubK B) {|Nonce NA, Agent A|} ∈ parts(spies evs)
           ⟶ A=A' & B=B'

3  [| Crypt(pubK B)  {|Nonce NA, Agent A|} ∈ parts(spies evs);
      Crypt(pubK B') {|Nonce NA, Agent A'|} ∈ parts(spies evs);
      Nonce NA ∉ analz(spies evs);
      evs ∈ ns_public |]
   ⟹ A=A' & B=B'

4  [| Says A B (Crypt (pubK B) {|Nonce NA, Agent A|}) ∈ set evs;
      A ∉ bad;  B ∉ bad;  evs ∈ ns_public |]
   ⟹ Nonce NA ∉ analz(spies evs)

5  [| Says A  B (Crypt (pubK B) {|Nonce NA, Agent A|}) ∈ set evs;
      Says B' A (Crypt (pubK A) {|Nonce NA, Nonce NB|}) ∈ set evs;
      A ∉ bad;  B ∉ bad;  evs ∈ ns_public |]
   ⟹ Says B A (Crypt(pubK A) {|Nonce NA, Nonce NB|}) ∈ set evs
```

Figure 3: The Guarantees for A in Isabelle/HOL Notation

item in X remains secret. Conclude that if X is revealed then A must have performed the decryption.

This roundabout procedure is necessary because the mere act of decryption gives weaker guarantees than exhibiting a signed message. Consider the following protocol:

1. $A \to B : Na, A$
2. $B \to A : \{\!|Na|\!\}_{Kb^{-1}}, Nb$
3. $A \to B : \{\!|Nb|\!\}_{Ka^{-1}}$

The nonces are broadcast to the world, but the signatures obviously assure A and B of the other's presence.

5.5 A Glimpse at the Machine Proofs

To give an impression of the Isabelle formalization, Fig. 3 presents the theorems providing guarantees for A. They are numbered as follows.

1. This unicity lemma states that Na (if secret) is not also used as Nb. It is proved by induction.

2. This unicity lemma states that, if Na is secret, then its appearance in any instance of message 1 determines the other components. It too follows by induction, with a standard proof script.

3. This corollary of the previous lemma has a trivial proof.

   These unicity lemmas refer to the presence of encrypted messages anywhere in past traffic. The remaining theorems refer to events of the form $\mathsf{Says}\, A\, B\, X$ involving such encrypted messages.

4. This crucial theorem guarantees the secrecy of Na. The conditions $A \notin \mathsf{bad}$ and $B \notin \mathsf{bad}$ express that both A and B are uncompromised. The proof is by induction; it relies on the previous three lemmas, which assume the secrecy of Na as an induction hypothesis.

5. This theorem is A's final guarantee. If A has used Na to start a run with B and receives the message $\mathsf{Crypt}(\mathsf{pubK}\, A)\{\!|Na, Nb|\!\}$, then B has sent that message. It is subject to both agents' being uncompromised. The proof is by induction and relies on the secrecy and unicity of Na.

The proof script for all five theorems comprises 27 commands (tactic invocations) and executes in ten seconds, or two seconds per theorem.

What about the guarantees for B? Attempting to prove the secrecy of Nb leads to a subgoal that appears to have no proof.

```
[| A∉bad;  B∉bad;  C∈bad;
   evs3∈ns_public;
   Says A  C (Crypt (pubK C) {|Nonce NA, Agent A|})  ∈ set evs3;
   Says B' A (Crypt (pubK A) {|Nonce NA, Nonce NB|}) ∈ set evs3;
   Says B  A (Crypt (pubK A) {|Nonce NA, Nonce NB|}) ∈ set evs3;
   Nonce NB∉analz (spies evs3) |]
==> False
```

This situation might arise when the last event is an instance of step 3, as we can tell because the trace is called $evs3$. Agents A and B are uncompromised and A has used Na to start a run with a compromised agent, C. Somebody has sent the message $\mathsf{Crypt}(\mathsf{pubK}\, A)\{\!|Na, Nb|\!\}$. We must show that these circumstances are contradictory, since the conclusion is just False. The conclusion is the simplified form of the claim that Nb remains secret even after A has sent the step 3 message $\mathsf{Crypt}(\mathsf{pubK}\, C)\{\!|Nb|\!\}$, but this message reveals Nb to the spy.

Such proof states can be hard to interpret. Does the induction formula require strengthening? Must additional lemmas be proved? But, in this case, we easily recognize Lowe's attack. The assumptions describe events that could actually occur: Nb need not remain secret.

5.6 Analyzing the Strengthened Protocol

Lowe [24] suggests improving the Needham-Schroeder protocol by adding explicitness. In step 2, agent B includes his identity:

1. $A \to B : \{\!|Na, A|\!\}_{Kb}$
2. $B \to A : \{\!|Na, Nb, B|\!\}_{Ka}$
3. $A \to B : \{\!|Nb|\!\}_{Kb}$

The previous proof scripts, by and large, still work for this version. Thanks to Isabelle's high level of automation, minor changes such as that above seldom interfere with existing proofs. The guarantees for A are proved precisely as before.

In proving guarantees for B, we naturally seek to strengthen them. The unicity property for Nb states that, if Nb is secret, then its presence in step 2 uniquely determines all other message components (recall §5.4). Step 2 now has the form

$$\mathsf{Crypt}(\mathsf{pubK}\, A)\{\!|Na, Nb, B|\!\}.$$

Nonce Nb determines not only A and Na but also B. This additional fact lets us prove the secrecy of Nb. Recall the subgoal presented in §5.5. With the new version of the protocol, somebody has sent the message

$$\mathsf{Crypt}(\mathsf{pubK}\, A)\{\!|Na, Nb, C|\!\}.$$

Also, agent B has sent the message

$$\mathsf{Crypt}(\mathsf{pubK}\, A)\{\!|Na, Nb, B|\!\}.$$

The unicity theorem for Nb implies $B = C$, a contradiction because C is compromised and B is not.

6 A Recursive Protocol

This protocol [9] generalizes Otway-Rees to an arbitrary number of parties. First, A contacts B. If B then contacts the authentication server then the run resembles Otway-Rees. But B may choose to contact some other agent C, and so forth; a chain of arbitrary length may form. During each such round, an agent adds its name and a fresh nonce to an ever-growing request message.

For the sake of discussion, suppose that C does not extend the chain but instead contacts the authentication server. The server generates fresh session keys Kab and Kbc—in the general case, one key for each pair of agents adjacent in the chain. It encloses each session key in two certificates,

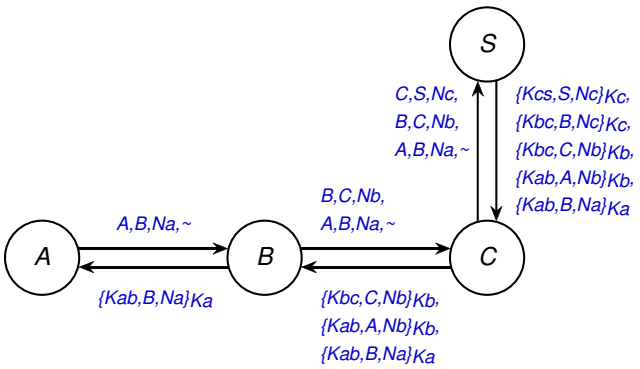

Figure 4: The Recursive Authentication Protocol with Three Clients

one for each party, and gives the bundle to C. Each agent removes two certificates and forwards the rest to its predecessor in the chain. Finally, A receives one certificate, containing Kab.

Such a protocol is hard to specify, let alone analyze. The number of steps, the number of parties and the number of session keys can vary. The server's response to the agents' accumulated requests must be given as a recursive program.

Even which properties to prove are not obvious. One might simplify the protocol to distribute a single session key, common to all the agents in the chain. But then, security between A and B would depend upon the honesty of C, an agent possibly not known to A. There may be applications where such a weak guarantee might be acceptable, but it seems better to give a separate session key to each adjacent pair.

I have proved a general guarantee for each participant. If it receives a certificate containing a session key and the name of another agent, then the spy will never know the key. The Isabelle proofs are modest in scale. Fewer than 30 results are proved, using under 130 commands; they run in about three minutes.

6.1 The Recursive Authentication Protocol

The protocol was invented by John Bull of APM Ltd. In the description below, let $\mathsf{Hash}\, X$ be the hash of X and $\mathsf{Hash}_X\, Y$ the pair $\{\!|\mathsf{Hash}\{\!|X, Y|\!\}, Y|\!\}$. Typically, X is an agent's long-term shared key and $\mathsf{Hash}\{\!|X, Y|\!\}$ is a message digest, enabling the server to check that Y originated with that agent. Figure 4 shows a typical run, omitting the hashing.

Agent A starts a run by sending B a request:

$$1.\ A \to B : \mathsf{Hash}_{Ka}\{\!|A, B, Na, -|\!\}$$

Here Ka is A's long-term shared key, Na is a fresh nonce, and $(-)$ is a placeholder indicating that this message started the run. In response, B sends something similar but with A's message in the last position:

$$2.\ B \to C : \mathsf{Hash}_{Kb}\{B, C, Nb, \overbrace{\mathsf{Hash}_{Ka}\{A, B, Na, -\}}^{\text{from } A}\}$$

Step 2 may be repeated as many times as desired. Each time, new components are added to the message and a new message digest is prefixed. The recursion terminates when some agent performs step 2 with the server as the destination.

In step 3, the server prepares session keys for each caller-callee pair. It traverses the accumulated requests to build up its response. If (as in §6.1) the callers were A, B and C in that order, then the final request is

$$\mathsf{Hash}_{Kc}\{C, S, Nc, \mathsf{Hash}_{Kb}\{B, C, Nb, \mathsf{Hash}_{Ka}\{A, B, Na, -\}\}\}.$$
$$\uparrow \qquad\qquad\qquad \uparrow$$

The arrows point to the occurrences of C, which appear in the outer two levels. C has called S (the server) and was called by B. The server generates session keys Kcs and Kbc and prepares the certificates $\{\!|Kcs, S, Nc|\!\}_{Kc}$ and $\{\!|Kbc, B, Nc|\!\}_{Kc}$. The session key Kcs is redundant because C already shares Kc with the server. Including it allows the last agent in the chain to be treated like all other agents except the first: the initiator receives only one session key.

Having dealt with C's request, the server discards it. Looking at the remaining outer two levels, the request message is

$$\mathsf{Hash}_{Kb}\{B, C, Nb, \mathsf{Hash}_{Ka}\{A, B, Na, -\}\}.$$
$$\uparrow \qquad\qquad\qquad \uparrow$$

The server now prepares two certificates for B, namely $\{\!|Kbc, C, Nb|\!\}_{Kb}$ and $\{\!|Kab, A, Nb|\!\}_{Kb}$. Note that Kbc appears in two certificates, one intended for C (containing nonce Nc and encrypted with key Kc) and one for B.

At the last iteration, the request message contains only one level:

$$\mathsf{Hash}_{Ka}\{A, B, Na, -\}.$$
$$\uparrow$$

The $(-)$ token indicates the end of the requests. The server generates one session key and certificate for A, namely $\{\!|Kab, B, Na|\!\}_{Ka}$.

Having processed the request message, the server returning a bundle of certificates. In our example, it would return five certificates to C.

$$\begin{aligned} 3.\ S \to C : {} & \{\!|Kcs, S, Nc|\!\}_{Kc}, \{\!|Kbc, B, Nc|\!\}_{Kc}, \\ & \{\!|Kbc, C, Nb|\!\}_{Kb}, \{\!|Kab, A, Nb|\!\}_{Kb}, \\ & \{\!|Kab, B, Na|\!\}_{Ka} \end{aligned}$$

In step 4, an agent accepts the first two certificates and forwards the rest to its predecessor in the chain. Every agent performs this step except the one who started the run.

$$\begin{aligned} 4.\ C \to B : & \{\!|Kbc, C, Nb|\!\}_{Kb}, \{\!|Kab, A, Nb|\!\}_{Kb}, \\ & \{\!|Kab, B, Na|\!\}_{Ka} \\ 4'.\ B \to A : & \{\!|Kab, B, Na|\!\}_{Ka} \end{aligned}$$

The description above describes a special case: a protocol run with three clients. The conventional protocol notation cannot cope with arbitrary numbers of participants, let alone recursive processing of nested messages. Section §6.3 below will specify the protocol as an inductive definition.

6.2 Deviations from the Protocol

I have corrected a flaw in the original protocol. My formalization of the protocol differs from the original in some other respects, as well.

In the original protocol, an agent's two certificates are distinguished only by their order of arrival; an intruder could easily exchange them. To correct this flaw, I have added the other party's name to each certificate. Such explicitness is good engineering practice [1]. It also simplifies the proofs (recall §4.8). Bull and Otway have accepted my change to their protocol [9].

The dummy session key *Kcs* avoids having to treat the last agent as a special case. All agents except the first take two certificates. An implementation can safely omit the dummy certificate. Removing information from the system makes less information available to an intruder.

The original protocol implements encryption using exclusive-or (XOR) and hashing. For verification purposes, encryption should be taken as primitive. Correctness of the protocol does not depend upon the precise form of encryption, provided it is implemented properly; the original use of XOR was flawed (see §6.6).

Protocol certificates are accompanied by agent names sent in clear. It is safe to simplify the specification by omitting these names.

6.3 Modelling the Protocol

Requests in the protocol have the form $\mathsf{Hash}_X\, Y$, where Y may contain another request. The $\mathsf{Hash}_X\, Y$ notation for message digests is trivially defined in Isabelle:

$$\mathsf{Hash}[X]Y \equiv \{\!|\mathsf{Hash}\{\!|X, Y|\!\}, Y|\!\}.$$

Most proofs do not apply this definition directly. The default rewrite rules apply only if $\mathsf{Hash}_X\, Y$ appears, say, in the argument of parts, where the expression can be simplified. Such rules help prevent exponential blowup.

A further law is subject to $X \notin \mathsf{synth}(\mathsf{analz}\, H)$, as when X is an uncompromised long-term key:

$$\mathsf{Hash}_X\, Y \in \mathsf{synth}(\mathsf{analz}\, H) \iff \mathsf{Hash}\{\!|X, Y|\!\} \in \mathsf{analz}\, H \wedge Y \in \mathsf{synth}(\mathsf{analz}\, H)$$

The message $\mathsf{Hash}_X\, Y$ can be spoofed iff Y can be and a suitable message digest is available (an unlikely circumstance).

```
Nil  [] ∈ recur

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

RA1 [| evs1 ∈ recur;  A≠B;  A≠Server;  Nonce NA ∉ used evs1 |]
     ⟹ Says A B
          (Hash[Key(shrK A)]
           {|Agent A, Agent B, Nonce NA, Agent Server|})
         # evs1 ∈ recur

RA2 [| evs2 ∈ recur;  B≠C;  B≠Server;  Nonce NB ∉ used evs2;
        Says A' B PA ∈ set evs2 |]
     ⟹ Says B C (Hash[Key(shrK B)]{|Agent B, Agent C, Nonce NB, PA|})
         # evs2 ∈ recur

RA3 [| evs3 ∈ recur;  B≠Server;
        Says B' Server PB ∈ set evs3;
        (PB,RB,K) ∈ respond evs3 |]
     ⟹ Says Server B RB # evs3 ∈ recur

RA4 [| evs4 ∈ recur;  A≠B;
        Says B  C {|XH, Agent B, Agent C, Nonce NB,
                    XA, Agent A, Agent B, Nonce NA, P|}
          ∈ set evs4;
        Says C' B {|Crypt (shrK B) {|Key KBC, Agent C, Nonce NB|},
                    Crypt (shrK B) {|Key KAB, Agent A, Nonce NB|},
                    RA|}
          ∈ set evs4 |]
     ⟹ Says B A RA # evs4 ∈ recur
```

Figure 5: Specifying the Recursive Protocol

For the most part, the protocol is modelled just like the fixed-length protocols discussed above. Figure 5 presents the inductive definition. The rules for the empty trace and the spy are standard. The other rules can be paraphrased as follows:

1. If, in the current trace, Na is a fresh nonce and B is an agent distinct from A and S, then we may add the event

$$\mathsf{Says}\, A\, B\, (\mathsf{Hash}_{\mathsf{shrK}\, A}\{\!|A, B, Na, -|\!\}).$$

A's long-term key is written $\mathsf{shrK}\,A$. For the token $(-)$ I used the name S, but any fixed message would do as well.

2. If the current trace contains the event $\mathsf{Says}\,A'\,B\,Pa$, where $Pa = \{\!|Xa, A, B, Na, P|\!\}$, and Nb is a fresh nonce and $B \neq C$, then we may add the event

$$\mathsf{Says}\,B\,C\,(\mathsf{Hash}_{\mathsf{shrK}\,B}\{\!|B, C, Nb, Pa|\!\}).$$

The variable Xa is how B sees A's hash value; he does not have the key needed to verify it. Component P is $(-)$ if A started the run or might have the same form as Pa, nested to any depth. Agent C might be the server or anybody else.

The specification actually omits the equation defining Pa. It appears to be unnecessary, and its omission simplifies the proofs. They therefore hold of a weaker protocol in which any agent may react to any message by sending an instance of step 2. Ill-formed requests may result, but the server will ignore them.

3. If the current trace contains the event $\mathsf{Says}\,B'\,\mathsf{S}\,Pb$, and $B \neq \mathsf{S}$, and if the server can build from request Pb a response Rb, then we may add the event

$$\mathsf{Says}\,\mathsf{S}\,B\,Rb.$$

The construction of Rb includes verifying the integrity of Pb; this process is itself defined inductively, as we shall see. The rule does not constrain the agent B, allowing the server to send the response to anybody. We could get the right value of B from Pb, but the proofs do not require such details.

4. If the current trace contains the two events

$$\begin{aligned}&\mathsf{Says}\ B\ C\,(\mathsf{Hash}_{\mathsf{shrK}\,B}\{\!|B, C, Nb, Pa|\!\})\\ &\mathsf{Says}\ C'\,B\,\{\!|\mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|Kbc, C, Nb|\!\},\\ &\qquad\qquad \mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|Kab, A, Nb|\!\},\ R|\!\}\end{aligned}$$

and $A \neq B$, then we may add the event

$$\mathsf{Says}\,B\,A\,R.$$

B decrypts the two certificates, compares their nonces with the value of Nb he used, and forwards the remaining certificates (R).

The final step of the protocol is the initiator's acceptance of the last certificate, $\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|Kab, B, Na|\!\}$. This implicit step need not be modelled; all certificates will be proved to be authentic.

There is no Oops message (recall §2.5). It cannot easily be expressed for the recursive authentication protocol because a key never appears together with both its nonces. The spy can still get hold of session keys using the long-term keys of compromised agents.

6.4 Modelling the Server

The server creates the list of certificates according to another inductive definition. It defines not a set of traces but a set of triples (P, R, K) where P is a request, R is a response and K is a session key. Such triples belong to the set respond evs, where evs (the current trace) is supplied to prevent the reuse of old session keys. Component K returns the newest session key to the caller for inclusion in a second certificate.

```
One  [| A ≠ Server;  Key KAB ∉ used evs |]
     ⟹ (Hash[Key(shrK A)] {|Agent A, Agent B, Nonce NA, Agent Server|},
         {|Crypt(shrK A) {|Key KAB, Agent B, Nonce NA|}, Agent Server|},
         KAB)   ∈ respond evs

Cons [| (PA, RA, KAB) ∈ respond evs;
        Key KBC ∉ used evs;  Key KBC ∉ parts {RA};
        PA = Hash[Key(shrK A)] {|Agent A, Agent B, Nonce NA, P|};
        B ≠ Server |]
     ⟹ (Hash[Key(shrK B)] {|Agent B, Agent C, Nonce NB, PA|},
         {|Crypt (shrK B) {|Key KBC, Agent C, Nonce NB|},
           Crypt (shrK B) {|Key KAB, Agent A, Nonce NB|},
           RA|},
         KBC)   ∈ respond evs
```

Figure 6: Specifying the Server

The occurrences of Hash in the definition ensure that the server accepts requests only if he can verify the hashes using his knowledge of the long-term keys. The inductive definition (Fig. 6) consists of two cases.

1. If Kab is a fresh key (that is, not used in evs) then

$$\begin{array}{l}(\,\mathsf{Hash}_{\mathsf{shrK}\,A}\{\!|A, B, Na, -|\!\},\\ \{\!|\mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|Kab, B, Na|\!\}, -|\!\}\\ Kab) \quad \in \quad \mathsf{respond}\,evs.\end{array}$$

   This base case handles the end of the request list, where A seeks a session key with B.

2. If $(Pa, Ra, Kab) \in \mathsf{respond}\,evs$ and Kbc is fresh (not used in evs or Ra) and

$$Pa = \mathsf{Hash}_{\mathsf{shrK}\,A}\{\!|A, B, Na, P|\!\}$$

then

$$\begin{array}{l}(\,\mathsf{Hash}_{\mathsf{shrK}\,B}\{B, C, Nb, Pa\},\\ \ \{\mathsf{Crypt}(\mathsf{shrK}\,B)\{Kbc, C, Nb\},\\ \ \ \mathsf{Crypt}(\mathsf{shrK}\,B)\{Kab, A, Nb\},\ Ra\},\\ \ Kbc) \quad \in \quad \mathsf{respond}\,evs.\end{array}$$

The recursive case handles a request list where B seeks a session key with C and has himself been contacted by A. The respond relation is best understood as a pure Prolog program. Argument Pa of (Pa, Ra, Kab) is the input, while Ra and Kab are outputs. Key Kab has been included in the response Ra and must be included in one of B's certificates too.

An inductive definition can serve as a logic program. Because the concept is Turing powerful, it can express the most complex behaviours. Such programs are easy to reason about.

A Coarser Model of the Server

For some purposes, respond is needlessly complicated. Its input is a list of n requests, for $n > 0$, and its output is a list of $2n+1$ certificates. Many routine lemmas hold for any list of certificates of the form $\mathsf{Crypt}(\mathsf{shrK}\,B)\{K, A, N\}$. The inductive relation responses generates the set of all such lists. It contains all possible server responses and many impossible ones.

The base case is simply $(-) \in \mathsf{responses}\,evs$ and the recursive case is

$$\{\mathsf{Crypt}(\mathsf{shrK}\,B)\{K, A, N\},\ R\} \in \mathsf{responses}\,evs$$

if $R \in \mathsf{responses}\,evs$ and K is not used in evs.

In secrecy theorems (those expressed in terms of analz), each occurrence of Crypt can cause a case split, resulting in a substantial blowup after simplification. Induction over responses introduces only one Crypt, but induction over respond introduces three. Because responses includes invalid outputs, some theorems can only be proved for respond.

6.5 Main Results Proved

For the most part, the analysis resembles that of the Otway-Rees protocol. Possibility properties are proved first, then regularity lemmas. Secrecy theorems govern the use of session keys, leading to the session key secrecy theorem: if the certificate $\mathsf{Crypt}(\mathsf{shrK}\,A)\{Kab, B, Na\}$ appears as part of any traffic, where A and B are uncompromised, then Kab will never reach the spy. Another theorem guarantees that such certificates originate only with the server.

Possibility properties are logically trivial. All they tell us is that the rules' message formats are compatible. However, their machine proofs require significant effort (or computation) due to the complexity of the terms that arise and the number of choices available. I proved cases corresponding to runs with up to three agents plus the server and spy. General theorems for n agents could be proved by induction on n, but the necessary effort hardly seems justified.

A typical regularity lemma states that the long-term keys of uncompromised agents never form part of any message. They do form part of hashed messages, however; recall the discussion in §3.2 above.

Security properties are proved, as always, by induction over the protocol definition. For this protocol, the main inductive set (recur) is defined in terms of another (respond). All but the most trivial proofs require induction over both definitions.

An easily-proved result lets us reduce responses to respond, justifying the use of induction over responses:

$$\frac{(Pa, Rb, Kab) \in \mathsf{respond}\, evs}{Rb \in \mathsf{responses}\, evs}$$

Most results are no harder to prove than for a fixed-length protocol. Proving a theorem requires four commands on average, of which two are quite predictable: induction and simplification. The outer induction yields six subgoals: one for each protocol step, plus the base and fake cases. The inner induction replaces the step 3 case by two subgoals: the server's base case and inductive step. Few of these seven subgoals survive simplification. Only the theorems described below have difficult proofs.

Nonces generated in requests are unique. There can be at most one hashed value containing the key of an uncompromised agent ($A \notin \mathsf{bad}$) and any specified nonce value, Na.

$$\exists B'\, P'.\, \forall B\, P.\\ \mathsf{Hash}\{\mathsf{Key}(\mathsf{shrK}\, A), \mathsf{Agent}\, A, \mathsf{Agent}\, B, Na, P\} \in \mathsf{parts}(\mathsf{spies}\, evs)\\ \longrightarrow B = B' \wedge P = P'.$$

Although it is not used in later proofs, this theorem is important. It lets agents identify runs by their nonces. The theorem applies to all requests, whether generated in step 1 or step 2. For the Otway-Rees protocol, each of the two steps requires its own theorem. The reasoning here is similar, but one theorem does the work of two, thanks to the protocol's symmetry. The nesting of requests does not affect the reasoning.

The session key compromise theorem is formulated just as for Otway-Rees (see §4.5), but its proof is much more difficult. The inner induction over respond leads to excessive case splits. It was to simplify this proof that I defined the set responses.

Unicity for session keys is unusually complicated because each key appears in two certificates. Moreover, the certificates are created in different iterations of respond. The unicity theorem states that, for any K, if there is a certificate of the form

$$\mathsf{Crypt}(\mathsf{shrK}\,A)\{K, B, Na\}$$

(where A and B are uncompromised) then the only other certificate containing K must have the form

$$\mathsf{Crypt}(\mathsf{shrK}\,B)\{K, A, Nb\},$$

for some Nb. If $(PB, RB, K) \in \mathsf{respond}\,evs$ then

$$\begin{aligned}&\exists A'\,B'.\,\forall A\,B\,N.\\ &\qquad \mathsf{Crypt}(\mathsf{shrK}\,A)\{\mathsf{Key}\,K, \mathsf{Agent}\,B, N\} \in \mathsf{parts}\{RB\}\\ &\qquad\qquad \longrightarrow (A' = A \wedge B' = B) \vee (A' = B \wedge B' = A).\end{aligned}$$

This theorem seems quite strong. An agent who receives a certificate immediately learns which other agent can receive its mate, subject to the security of both agents' long-term keys. One might hope that the session key secrecy theorem would follow without further ado. The only messages containing session keys contain them as part of such certificates, and thus the keys are safe from the spy. But such reasoning amounts to another induction over all possible messages in the protocol. The theorem must be stated (stipulating A, $A' \notin \mathsf{bad}$) and proved:

$$\begin{aligned}&\mathsf{Crypt}(\mathsf{shrK}\,A)\{\mathsf{Key}\,K, \mathsf{Agent}\,A', N\} \in \mathsf{parts}(\mathsf{spies}\,evs)\\ &\qquad\qquad \longrightarrow \mathsf{Key}\,K \notin \mathsf{analz}(\mathsf{spies}\,evs)\end{aligned}$$

The induction is largely straightforward except for the step 3 case. The inner induction over respond leads to such complications that it must be proved beforehand as a lemma. If $(PB, RB, Kab) \in \mathsf{respond}\,evs$ then

$$\begin{aligned}&\forall A\,A'\,N.\,A \notin \mathsf{bad} \wedge A' \notin \mathsf{bad}\\ &\qquad \longrightarrow \mathsf{Crypt}(\mathsf{shrK}\,A)\{\mathsf{Key}\,K, \mathsf{Agent}\,A', N\} \in \mathsf{parts}\{RB\}\\ &\qquad\qquad \longrightarrow \mathsf{Key}\,K \notin \mathsf{analz}(\{RB\} \cup \mathsf{spies}\,evs)\end{aligned}$$

Although each session key appears in two certificates, they both have the same format. A single set of proofs applies to all certificates. Once again, the protocol's symmetry halves the effort compared with Otway-Rees.

It may be instructive to see some theorems in Isabelle syntax. Here is the session key compromise theorem:

```
[| evs ∈ recur;  KAB ∉ range shrK |]
⟹ (Key K ∈ analz (insert (Key KAB) (spies evs))) =
    (K = KAB ∨ Key K ∈ analz (spies evs))
```

And here is the session key secrecy theorem:

```
[| Crypt (shrK A) {|Key K, Agent A', N|} ∈ parts (spies evs);
   A ∉ bad;  A' ∉ bad;  evs ∈ recur |]
⟹ Key K ∉ analz (spies evs)
```

6.6 Potential Attacks

All proofs are subject to the assumptions implicit in the model. Attacks against the protocol or implementations of it can still be expected. One 'attack' is obvious: in step 2, agent B does not know whether A's message is recent; at the conclusion of the run, B still has no evidence that A is present. The spy can masquerade as A by replaying an old message of hers, but cannot read the resulting certificate without her long-term key.

Allowing type confusion (such as passing a nonce as a key) often admits attacks [27, 29] in which one form of certificate is mistaken for another. The recursive authentication protocol is safe from such attacks because it has only one form of certificate. However, encryption must be secure.

In the original protocol, each session key was encrypted by forming its XOR with a hash value, used as a one-time pad. Unfortunately, each hash value was used twice: B's session keys Kab and Kbc were encrypted as

$$Kab \oplus \mathsf{Hash}\{\!|Kb, Nb|\!\} \quad \text{and} \quad Kbc \oplus \mathsf{Hash}\{\!|Kb, Nb|\!\}.$$

By forming their XOR, an eavesdropper could immediately obtain $Kab \oplus Kbc$, $Kbc \oplus Kcd$, etc. Compromise of any one session key would reveal all the others [41].

7 Related Work

Several other researchers are using inductive or trace models. Verification is done using general-purpose theorem provers or model checkers, or by hand.

In early work, Kemmerer [23] analyzed a protocol in the Ina Jo specification language, which is based on first-order logic. Using an animation tool, he identified two weaknesses in the protocol. He modelled the system as an automaton, defining the initial state and the state transformations, and specifying security goals as invariants. Proving that state transformations preserve an invariant is the same style of reasoning as induction. Gray and McLean [19] also establish a security invariant by induction, though their work is based on temporal logic and their proofs are done by hand.

Bolignano's work [5] is based on the Coq proof checker. His $X \mathsf{known_in} H$ is equivalent to my $X \in \mathsf{synth}(\mathsf{analz}\, H)$ and models fraudulent messages. Instead of formalizing traces, he models the states of the four agents A, B, S and the spy. He and Ménissier-Morain [6] have formalized the Otway-Rees protocol. In their model, the server uses the function KeyAB to choose session keys: whenever A and B participate in a run, the server issues $\mathsf{KeyAB}(\mathsf{A}, \mathsf{B})$. They have proved three properties. The first resembles the forwarding lemmas described in §4.2. The second states that the secret keys Ka and Kb remain secret; it is similar to a regularity lemma described in §4.3. The third property states that $\mathsf{KeyAB}(\mathsf{A}, \mathsf{B})$ cannot be decrypted from traffic even with the help of all other session keys.

Their model is somewhat restrictive. The constants A and B are fixed in the roles of initiator and responder, respectively. The spy starts off holding no keys, which gives him no prospect of impersonating honest agents or decrypting messages. Also, note that the attack of §2.9 works not by giving $\mathsf{KeyAB}(\mathsf{A}, \mathsf{B})$ to the spy, but by getting A to accept $\mathsf{KeyAB}(\mathsf{C}, \mathsf{A})$ as a good key for talking to B.

Lowe's approach is based on traces, which are specified using the process calculus CSP [21] and examined using the model-checker FDR. His work originates in that of Roscoe [39]. Like Bolignano, he models the four agents A, B, S and the spy. However, his model is more realistic. A and B may engage in concurrent runs, playing either role; the spy has an identity and a long-term key. Lowe has discovered numerous attacks, some of which are serious [24, 25, 26]. I have found his papers most useful in developing the Isabelle model.

Meadows's paper on Needham-Schroeder [29] makes direct comparisons with Lowe's. She examines the same variants of the protocol and discusses differences in speed between the NRL Protocol Analyzer and FDR (the latter is faster). She reports many other experiments, for example on the possibility of nonces being compromised. Not all of these attempts are successful: sometimes the state space becomes too large.

Her Protocol Analyzer performs an unusual combination of search and proof. Ostensibly based on brute-force state enumeration, it can also prove by induction that infinite sets of states are unreachable. A full analysis carries the same assurance as a formal proof. The precise relationship between Meadows's and my uses of induction needs to be examined.

Schneider [42], like Lowe, bases his work on CSP [21]. But instead of using a model checker, he applies the laws of CSP in proofs. A rank function is used to describe how an undesirable event is prevented. Proving certain theorems about the rank function establishes the property in question. Schneider has published detailed hand analyses of both the original protocol and Lowe's version. He considers a number of authentication properties in increasingly general settings, ultimately allowing concurrent runs. Recently, Dutertre and Schneider [16] have mechanized these hand proofs, revealing

many errors in them. Bryans and Schneider [8] have proved some simple properties for a single run of the recursive authentication protocol.

Schneider has considered the consequences of allowing messages to satisfy equational laws on messages. Many protocols—and attacks!—exploit algebraic properties of encryption method, particularly RSA [38].

8 Conclusions

The inductive method is simple and general. We have seen how it handles three versions of Otway-Rees, two versions of Needham-Schroeder (with public keys), and a recursive protocol. The analysis of Needham-Schroeder reveals Lowe's attack, and I have discovered a new attack in a variant of Otway-Rees. In addition to the protocols discussed above, I have analyzed two variants of Yahalom [36], a simplified version of Woo-Lam [1] and the shared-key version of Needham-Schroeder. Bella and I have looked at Kerberos, which is based on timestamps; its use of session keys to encrypt other keys complicates its analysis [4]. I have modelled part of the Internet protocol TLS [15, 35] in which secret nonces are exchanged, then used to compute session keys.

Proofs are highly automated. One Isabelle command can generate thousands of inferences. Small changes to protocols involve only small changes to proof scripts.[4] Analyzing Needham-Schroeder took only 30 hours of my time, the recursive protocol two weeks. These figures include time spent extending the model with public-key encryption and hashing. Adherence to design principles such as explicitness [1] simplifies proofs.

Model checking is an effective means of finding attacks [24, 25, 26], but it cannot replace theorem proving. It copes with only finitely many states, and the failure to find an attack says nothing about how a protocol works.

An inductive proof is a symbolic examination of the protocol. Each step is analyzed in turn. The reasoning can be explained informally, letting us understand how the protocol copes with various circumstances. When a protocol is modified, the proof scripts for the old version form the starting point for its analysis. These scripts take only a few minutes to run, which is competitive with model checking.

The two methods complement each other. A protocol designer might use model checking for a quick inspection and apply the inductive approach to investigate deeper properties.

Formal methods cannot guarantee security. Theorems can easily be misinterpreted. Needham-Schroeder is correct from A's point of view, but not from B's. The flawed version of Otway-Rees is correct from the server's point of view, but the other participants cannot detect tampering (recall

[4] Proof scripts are distributed with Isabelle, which can be obtained from URL `http://www.cl.cam.ac.uk/Research/HVG/Isabelle/dist/`; see subdirectory `HOL/Auth`.

§4.7). A protocol proof must contain a separate guarantee—under reasonable assumptions—for each participant.

The attack on the recursive protocol [41] is a sobering reminder of the limitations of formal methods. Models idealize the real world: here, by assuming strong encryption. Making the model more detailed makes reasoning harder and, eventually, infeasible. A compositional approach seems necessary: different levels of abstraction, such as protocol messages, cryptographic algorithms, and transport protocols, should be verified separately. Devising such an approach will be a challenge.

Acknowledgement R. Needham gave much valuable advice. Thanks are due to J. Bull and D. Otway for explaining their protocol to me. A. Gordon suggested a simplification to the treatment of freshness. Conversations with M. Abadi, D. Bolignano, G. Huet, P. Ryan and K. Wagner were helpful. G. Bella, B. Graham, G. Lowe, F. Massacci, V. Matyas, M. Staples, M. VanInwegen and several anonymous referees commented on drafts of this article. The research was funded by the EPSRC grants GR/K77051 'Authentication Logics' and GR/K57381 'Mechanizing Temporal Reasoning', and by the ESPRIT working group 21900 'Types'.

References

[1] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[2] Peter Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland, 1977.

[3] Ross Anderson and Roger Needham. Programming Satan's computer. In Jan van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, LNCS 1000, pages 426–440. Springer, 1995.

[4] Giampaolo Bella and Lawrence C. Paulson. Using Isabelle to prove properties of the Kerberos authentication system. In Orman and Meadows [30].

[5] Dominique Bolignano. An approach to the formal verification of cryptographic protocols. In *Third ACM Conference on Computer and Communications Security*, pages 106–118. ACM Press, 1996.

[6] Dominique Bolignano and Valérie Ménissier-Morain. Formal verification of cryptographic protocols using Coq. Technical report, INRIA-Rocquencourt, 1996.

[7] Stephen H. Brackin. A HOL extension of GNY for automatically analyzing cryptographic protocols. In Computer Security Foundations Workshop [13], pages 62–75.

[8] Jeremy Bryans and Steve Schneider. CSP, PVS and a recursive authentication protocol. In Orman and Meadows [30].

[9] John A. Bull and David J. Otway. The authentication protocol. Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/0.5b, Defence Research Agency, Malvern, UK, 1997. In press.

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

[11] John Clark and Jeremy Jacob. On the security of recent protocols. *Information Processing Letters*, 56(3):151–155, 1995.

[12] *8th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1995.

[13] *9th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[14] *10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[15] Tim Dierks and Christopher Allen. The TLS protocol: Version 1.0, November 1997. Internet-draft `draft-ietf-tls-protocol-05.txt`.

[16] Bruno Dutertre and Steve Schneider. Using a PVS embedding of CSP to verify authentication protocols. In Elsa Gunter, editor, *Theorem Proving in Higher Order Logics: TPHOLs '97*, LNCS, pages 121–136, 1997. In press.

[17] Dieter Gollmann. What do we mean by entity authentication? In *Symposium on Security and Privacy*, pages 46–54. IEEE Computer Society, 1996.

[18] M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[19] James W. Gray, III and John McLean. Using temporal logic to specify and verify cryptographic protocols. In Computer Security Foundations Workshop [12], pages 108–116.

[20] Matthew Hennessy. *The Semantics of Programming Languages: An Elementary Introduction Using Structural Operational Semantics.* Wiley, 1990.

[21] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[22] Richard Kemmerer, Catherine Meadows, and Jonathan Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[23] Richard A. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, May 1989.

[24] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems. Second International Workshop, TACAS '96*, LNCS 1055, pages 147–166, 1996.

[25] Gavin Lowe. Some new attacks upon security protocols. In Computer Security Foundations Workshop [13], pages 162–169.

[26] Gavin Lowe. SPLICE/AS: A case study in using CSP to detect errors in security protocols. Technical report, Oxford University Computing Laboratory, 1996.

[27] Gavin Lowe. Casper: A compiler for the analysis of security protocols. In Computer Security Foundations Workshop [14], pages 18–30.

[28] Wenbo Mao and Colin Boyd. Towards formal analysis of security protocols. In *Computer Security Foundations Workshop VI*, pages 147–158. IEEE Computer Society Press, 1993.

[29] Catherine A. Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Computer Security — ESORICS 96*, LNCS 1146, pages 351–364. Springer, 1996.

[30] Hilarie Orman and Catherine Meadows, editors. *Workshop on Design and Formal Verification of Security Protocols.* DIMACS, September 1997.

[31] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

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

[33] Lawrence C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 2nd edition, 1996.

[34] Lawrence C. Paulson. Generic automatic proof tools. In Robert Veroff, editor, *Automated Reasoning and its Applications: Essays in Honor of Larry Wos*, chapter 3. MIT Press, 1997.

[35] Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. Technical Report 440, Computer Laboratory, University of Cambridge, December 1997.

[36] Lawrence C. Paulson. On two formal analyses of the Yahalom protocol. Technical Report 432, Computer Laboratory, University of Cambridge, July 1997.

[37] Lawrence C. Paulson. Tool support for logics of programs. In Manfred Broy, editor, *Mathematical Methods in Program Development: Summer School Marktoberdorf 1996*, NATO ASI Series F, pages 461–498. Springer, Published 1997.

[38] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[39] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In Computer Security Foundations Workshop [12], pages 98–107.

[40] Peter Y. A. Ryan. The design and verification of security protocols. Technical Report DRA/CIS3/SISG/CR/96/1.0, Defence Research Agency, May 1996.

[41] Peter Y. A. Ryan and S. A. Schneider. An attack on a recursive authentication protocol: A cautionary tale. Technical Report TR97245, Defence Research Agency, 1997. In Press.

[42] Steve Schneider. Verifying authentication protocols with CSP. In Computer Security Foundations Workshop [14], pages 3–17.