Proving Properties of Security Protocols by Induction[1]

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

lcp@cl.cam.ac.uk

[1]An abridged version of this paper appeared in the Proceedings of the 10th Computer Security Foundations Workshop, 10–12 June 1997, pages 70–83.

Abstract

Informal justifications of security protocols involve arguing backwards that various events are impossible. *Inductive definitions* can make such arguments rigorous. The resulting proofs are complicated, but can be generated reasonably quickly using the proof tool Isabelle/HOL. There is no restriction to finite-state systems and the approach is not based on belief logics.

Protocols are inductively defined as sets of traces, which may involve many interleaved protocol runs. Protocol descriptions model accidental key losses as well as attacks. The model spy can send spoof messages made up of components decrypted from previous traffic.

Several key distribution protocols have been studied, including Needham-Schroeder, Yahalom and Otway-Rees. The method applies to both symmetric-key and public-key protocols. A new attack has been discovered in a variant of Otway-Rees (already broken by Mao and Boyd). Assertions concerning secrecy and authenticity have been proved.

Contents

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*: agents can identify themselves to others, but nobody can masquerade as somebody else. 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 four messages, but are surprisingly hard to get right. Anderson and Needham's excellent tutorial [5] presents several examples and defines the notation and terminology used below.

Many researchers are using formal methods to analyze security protocols. Two popular approaches are *state exploration* and *belief logics*.

- State exploration methods [22] model the protocol as a finite state system and verify by exhaustive search that all reachable states are safe. Lowe, for example, models protocols in CSP [13] and applies a model-checker to explore their behaviour [15, 17]. The Interrogator [14] is another finite-state tool. Such methods can find attacks quickly, but keeping the state space small requires drastic simplifying assumptions.

- Belief logics formally express what an agent may infer from messages received. The original BAN logic [9] allows short proofs expressed at a high level of abstraction. It has identified weaknesses in some protocols, but it has also failed to identify serious weaknesses in others. The field remains promising; researchers such as Mao and Boyd [18] are developing new belief logics. However, such logics do not address secrecy, and some of them are hard to put on a scientific footing.

We can fruitfully borrow elements 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 protocol message. Protocols are formalized simply as the set of all possible traces of events. 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. We can also model accidents and attacks.

Properties are proved by induction on traces. Proofs are much too long to carry out on paper, but the theorem prover Isabelle [20] allows partial automation of the proofs. 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. A complete Isabelle proof script executes in a few minutes. Laws and proof techniques developed for one protocol are often applicable to others.

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. [9, page 247]. At the time, I was unaware of Mao and Boyd's earlier attack [18], and had been assuming the protocol to be correct.

The paper goes on to explain the role of induction (§2) and present an overview of the method (§3). Then it presents the logical details more closely: the theory of messages and shared-key protocols (§4) and the formal proofs for three variants of Otway-Rees (§5). Some conclusions follow (§6).

2 Why an Inductive Approach?

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* [3]. 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. This approach is highly general: it has long been used to specify the semantics of programming languages [12] and can be applied to many nondeterministic processes.

Analyzing security protocols requires modelling the capabilities of an attacker. Only those capabilities included in the model can be considered in proofs. Contrast with the Spi calculus [1], where protocols are proved correct with respect to an arbitrary environment.

Several inductively-defined operators are useful in specifications. One (parts) merely enumerates all the components of a set of messages. Another (analz) models the decryption of past traffic using available keys. Another (synth) models the generation of spoof 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.

Each protocol is itself described by a further inductive definition. It incorporates the attacker as well as the behaviour of honest agents faithfully executing protocol steps. 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 [8] is to eliminate human error. In contrast, inductive verification of protocols involves long and highly 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.* Afraid you've got me there.

3 Overview

The method includes a theory of message analysis and a theory describing standard features of protocols. Individual protocol descriptions rest on this common base.

The traditional notation used for describing protocols conflicts somewhat with the need for mechanization. Expressing concatenation using a comma, as in A, B, would be ambiguous. Enclosing it in braces, as in $\{A, B\}$, invites confusion with finite sets. I have used fat braces to express concatenation, as in $\{\!|A, B|\!\}$. Informal protocol descriptions revert partly to the traditional notation, omitting outer-level braces and indicating encryption by a notation such as $\{\!|Na, Kab|\!\}_{Ka}$.

3.1 Messages

Message items may include

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

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

The formalization assumes that an encrypted message can neither be altered nor read without the appropriate key. The structure of each message is explicit, and different types of components cannot be confused.

Implementors must take care. Some published attacks involve accepting a nonce as a key [16] or regarding one component as being two [10]. Many real-world failures are the fault of slipshod human procedures [4]. Attempting to model the countless forms of carelessness would be futile.

3.2 Message Analysis

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.) We might regard $\mathsf{parts}\,H$ as what God could decrypt from H. Proving $X \notin \mathsf{parts}\,H$ establishes that X does not occur in H at all. Here are some laws about parts:

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

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 maximum knowledge that a determined spy could gain by analyzing intercepted traffic but not breaking ciphers. Proving $K \notin \mathsf{analz}\,H$ should assure us that nobody could learn K by listening to H. Here are some laws about analz:

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

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 some laws about synth:

$$
\begin{aligned}
X \in \mathsf{synth}\, H,\ K \in H &\Longrightarrow \mathsf{Crypt}\, K X \in \mathsf{synth}\, H \\
K \in \mathsf{synth}\, H &\Longrightarrow K \in H \\
\mathsf{synth}\, G \cup \mathsf{synth}\, H &\subseteq \mathsf{synth}(G \cup H) \\
\mathsf{synth}(\mathsf{synth}\, H) &= \mathsf{synth}\, H.
\end{aligned}
$$

3.3 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: there are traces where agents ignore some messages. The formalized protocol describes what can happen, but nobody is forced to do anything, just as in the real world where machines crash and communication links break.

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. Typically, the model gives the spy access to an unspecified set of "lost" long-term keys. An "oops" message in protocol descriptions models 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.

One might imagine that proving anything might be difficult in the presence of so powerful a spy. But standard protocol steps can be just as difficult to analyze. Sometimes an agent forwards an encrypted part of a message (which might be anything) to somebody else. It is a great help that all protocols assume the same sort of spy. A common body of laws and proof techniques can be developed for use in all proofs.

3.4 Modelling a Protocol

Each event in a trace has the form $\mathsf{Says}\, A\, B\, X$, meaning A says message X to B. Other events could be envisaged, such as the changing of a long-term key. Each agent's state is represented by its initial knowledge (typically its key, shared with the server) and what it can scan from the list of events. Apart from the spy, agents only read messages addressed to themselves.

Consider a variant of the Otway-Rees protocol [9, page 247]:[1]

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. $B \to A : Na, \{\!|Na, Kab|\!\}_{Ka}$

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 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

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

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

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

[1] Here is an informal description of this protocol.

1. A asks B to set up a secure conversation, generating Na to identify the run.
2. B forwards A's message to the authentication server, adding a nonce of his own.
3. S generates a new session key Kab and packages it separately for A and B.
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 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

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

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

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

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 rule does not require the message from S' to be more recent than that from B; this holds by the freshness of Nb.

An agent may participate in several protocol runs concurrently; the trace represents his state in all those runs. Agents may respond to past events, no matter how old they are. They 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 and complicate the analysis. Excluding some traces as ill-formed weakens theorems proved about all traces.

A protocol description usually requires three additional rules:

- The empty list, [], is a trace.
- 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 as well as the spy's initial state, which holds the long-term keys (supposedly shared with the server) of an arbitrary set of "lost" agents. Thus, the spy may say anything he plausibly could say, and can masquerade as any of the lost agents.
- If evs is a trace and S distributed key K to A, and nonces Na and Nb were used, then evs may be extended with the event

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

This strange-looking rule, the "oops" rule, models the accidental loss of session keys. Even if A and B safeguard their long-term keys Ka and Kb, they may have carelessly revealed an old session key. We need an assurance that such old keys cannot compromise future runs. The "oops" message includes nonces in order to identify the protocol run. This distinguishes between the recent loss of a key and one in the past.

A normal protocol rule would never demand that a message originated from a particular agent (here S), because the recipient cannot know who the originator is. The oops rule, however, is not intended to model A's knowledge. Insisting that the originator is S ensures that the oops message $\{\!|Na, Nb, K|\!\}$ specifies the true values of the nonces.

3.5 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.

3.6 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.

It is easy to prove that the spy never gets hold of any agent's long-term key except those presumed lost at the outset. 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 for the lost keys).

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 generate these numbers using the means provided in the model, then it is straightforward to prove that the key (or nonce) part of a message determines the values of the other parts. A standard proof script works for all the protocols considered.

3.7 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 say 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 involves secrecy of session keys: 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 $\mathsf{Crypt}\, Kab \,\{\!|\ldots Kcd \ldots|\!\}$. Unfortunately, the spy himself can either send such messages directly or cause other agents to send them. To work such mischief, he must already possess Kcd, so he does not thereby learn new session keys.

The discussion above suggests the precise form of the theorem. If K can be analyzed with the help of a session key K' and previous traffic, then either $K = K'$ or K can be analyzed 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.

The first time I proved this theorem it seemed complicated and difficult, but the script is now down to six commands, and essentially the same script works for all the protocols investigated. Proving this theorem, even for a new protocol, is now routine. 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 [11, §2.1]).

A crucial secrecy property states that if the server distributes a session key Kab to A and B, then no other agent gets this key. It is necessary to assume that A and B have not lost their secret keys to the spy, 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.

This property can usually be proved with fewer than a dozen commands. A specialized proof procedure deals with unknown message components, both fraudulent messages and forwarded parts of previous messages. Such mechanical procedures are called *tactics*; a typical command invokes one tactic.

Proving the first result of this kind took weeks, partly spent building libraries of tactics and relevant laws. 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.

3.8 An Attack

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 (§3.4), 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 issue him a session key Kca and also to package Na with this key. He then masquerades as B.

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

Replacing nonce Na' by A's original nonce Na (in message $2''$) eventually causes A to accept key K_{ca} as a key for talking with B. This attack is more serious than that discovered by Mao and Boyd [18], 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.

As Abadi and Needham discuss [2], the Otway-Rees protocol uses nonces not merely to assure freshness, but for binding: to identify the principals. 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 clash. 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.

This concludes the overview of how protocols are analyzed. Let us now examine the theory and its mechanization in more detail.

4 A Mechanized Theory of Messages

The approach has been mechanized using Isabelle/HOL, an instantiation of the generic theorem prover Isabelle [19, 20] 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, but some mathematical symbols have been substituted for their ASCII equivalents to improve legibility.

Higher-order logic is a typed formalism. An item of type agent can never be used where something of type msg is expected. Isabelle uses type inference to eliminate the need to specify types in expressions and to allow polymorphic theories such as those of lists and sets.

4.1 Agents and Messages

There are three kinds of agents: the server S, the friendly (or honest) agents, and the spy. Friendly agents have the form $\mathsf{Friend}\, i$, where i is a natural number. The following declaration specifies type agent to Isabelle. (The type of natural numbers is nat.)

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

This, like any datatype declaration, introduces axioms to say that the various kinds of agent are distinct from each other. Thus we have $\mathsf{S} \neq \mathsf{Friend}\, i$, $\mathsf{S} \neq \mathsf{Spy}$, $\mathsf{Spy} \neq \mathsf{Friend}\, i$, and also $\mathsf{Friend}\, i = \mathsf{Friend}\, j$ only if $i = j$.

The five kinds of message items (discussed above, §3.1) are declared similarly. Observe the reference to 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
             | Nonce nat
             | Key   key
             | MPair msg msg
             | Crypt key msg
```

Again, the various kinds of message are distinct. We have $\mathsf{Agent}\, A \neq \mathsf{Nonce}\, N$, and $\mathsf{Nonce}\, N = \mathsf{Nonce}\, N' \Longrightarrow N = N'$, etc. Less expected is the law

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

This is easily violated in practice if the plaintext is just a nonce, say, but not if it contains redundancy.

4.2 Message Analysis

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. The following rules express the definition precisely.

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

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

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

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

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

The set $\mathsf{synth}\,H$ is the least that includes H and all agent names, and is closed under pairing and encryption.

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

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

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.

4.3 Laws Governing the Operators

Section 3.2 presented a few of the laws proved for the operators, but protocol verification requires many more. Let us examine them systematically.

Below, $\mathsf{ins}\,X\;H$ stands for the set $\{X\} \cup H$, whose elements are X and those of H.

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

$$\begin{aligned} \mathsf{parts}\,G &\subseteq \mathsf{parts}\,H \\ \mathsf{analz}\,G &\subseteq \mathsf{analz}\,H \\ \mathsf{synth}\,G &\subseteq \mathsf{synth}\,H. \end{aligned}$$

They are idempotent:

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

Similarly, we have the equations

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

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 $\mathsf{synth}(\mathsf{analz}\,H)$, are irreducible. The latter one models the fraudulent messages that a spy could derive from H. We can still prove useful laws such as

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

More generally, we have a rule that bounds what the enemy can say:

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

In a typical proof, G is the set of all messages sent during a trace, while H is some known set of messages. (Often, $G = H$ or $G \subseteq H$.) The rule lets us eliminate the fraudulent message X. It yields an upper bound on $\mathsf{parts}(\mathsf{ins}\,X\;H)$ expressed in terms of $\mathsf{parts}\,H$, which itself is bounded by the induction hypothesis. An analogous rule is useful for proving theorems expressed in terms of analz.

4.4 Symbolic Evaluation

Applying rewrite rules to a term such as

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

can transform it to the equivalent term

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

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

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

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

$$\begin{aligned}&\{\!\!\{\,\mathsf{Nonce}\,Na, \mathsf{Agent}\,A, \mathsf{Agent}\,B,\\&\;\;\mathsf{Crypt}\,Ka\{\!\!\{\mathsf{Nonce}\,Na, \mathsf{Agent}\,A, \mathsf{Agent}\,B\}\!\!\}\}\!\!\}.\end{aligned}$$

Now $\mathsf{parts}(\mathsf{ins}\,X\;H)$ 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 $\mathsf{Key}\,K \notin \mathsf{parts}(\mathsf{ins}\,X\;H)$ 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 element were present, then the subgoal would still simplify to a manageable formula, $K \neq K' \wedge \mathsf{Key}\,K \notin \mathsf{parts}\,H$.

The rules that perform this symbolic evaluation are fairly obvious:

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

Symbolic evaluation of analz is more difficult. A key can only be pulled out if there are no messages encrypted using its inverse. A message encrypted using K can only be pulled out if K^{-1} is not available. We need an operator to denote the set of keys that can be used to decrypt messages in H:

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

Now we can prove that a key can be pulled out provided it is not required for decryption.

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

The rule for pulling out encrypted messages involves case analysis:

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

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 still be enormous, but symbolic evaluation at least expresses $\mathsf{analz}(\mathsf{ins}\, X\; H)$ in terms of $\mathsf{analz}\, H$.

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 spoof. There is a similar rule for $\{\!|X, Y|\!\} \in \mathsf{synth}\, H$.

In all, over ninety theorems are proved about parts, analz, synth and keysFor. Most of them are stored in such a way that Isabelle can apply them automatically for simplification. Most are proved by induction with respect to the relevant inductive definition. Logically speaking, some of these proofs are complex, but they need (on average) under three commands each, thanks to Isabelle's automatic tools. The full proof script, over 200 commands, executes in under ninety seconds.

4.5 Events and Agent Knowledge

A trace is a list of events. Isabelle/HOL provides lists, while events are trivial to declare as a datatype:

```
datatype event = Says agent agent msg
```

Triples of the form (A, B, X) might be used, but the datatype gives a nicer notation and accommodates future extensions to other sorts of events.

The function sees models the set of messages an agent receives from a trace. Its first argument, *lost*, is a set of agents whose keys (supposedly shared only with the server) have been lost to the spy. Honest agents only see messages intended for themselves, but the spy sees all traffic. (Recall that $ev\#evs$ is the list consisting of ev prefixed to the list evs.)

$$\mathsf{sees}\, lost\, A\, ((\mathsf{Says}\, A'\, B\, X) \# evs) = \begin{cases} \mathsf{ins}\, X (\mathsf{sees}\, lost\, A\, evs) & A \in \{B, \mathsf{Spy}\} \\ \mathsf{sees}\, lost\, A\, evs & \text{otherwise} \end{cases}$$

From the empty trace, an agent sees his initial state:

$$\mathsf{sees}\, lost\, A\, [] = \mathsf{initState}\, lost\, A$$

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

$$\begin{aligned} \mathsf{initState}\, lost\, \mathsf{S} &= \text{all long-term keys} \\ \mathsf{initState}\, lost\, (\mathsf{Friend}\, i) &= \{\mathsf{Key}(\mathsf{shrK}(\mathsf{Friend}\, i))\} \\ \mathsf{initState}\, lost\, \mathsf{Spy} &= \{\mathsf{Key}(\mathsf{shrK}(A)) \mid A \in lost\} \end{aligned}$$

A peculiarity is that the spy does not see his own key unless $\mathsf{Spy} \in lost$, which some theorems assume explicitly. We hardly need to consider sees and initState except when formalizing the creation of spoof messages. The descriptions of the legitimate messages directly model what the agents may be presumed to see.

The file that develops the simple theory outlined above also defines proof tactics common to all protocols. It provides specialized rewriting for analz and a tactic for proving cases involving fraudulent messages.

4.6 The Formal Protocol Specification

Section 3.4 discussed the modelling of a protocol informally, though in detail. Now, let us consider the specification as 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 called otway. (Giving

```
Nil  [] ∈ otway

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

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

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

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

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

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

Figure 1: Specifying a Protocol

each formal protocol a different name allows reasoning about several of them at once, though I have not tried this.)

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

$$X \in \mathsf{synth}(\mathsf{analz}(\mathsf{sees}\, lost\, \mathsf{Spy}\, evs))$$

denotes anything that could be faked from what the spy could decrypt with the help of the lost keys. The spy can send any such message 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 condition eliminates some impossible cases in proofs.

Rule OR1 formalizes step 1 of Otway-Rees. List evs is the current trace. The nonce Na may be anything not contained in $\mathsf{used}\, evs$; this set contains all message components that are accessible to some agent. An agent has no sure means of generating fresh nonces, but can do so with a high probability by using a random process or a reliable clock. The spy may use rule OR1, although a successful attack is more likely to involve the replay of an old nonce through rule Fake.

In rule OR2, $\mathsf{set\_of\_list}\, evs$ 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 above (§3.4). Translating informal protocol notation into Isabelle format is perhaps sufficiently straightforward to be automated.

5 Protocol Proofs

Section 3 outlined the form of induction used, the key theorems proved about the Otway-Rees variant, the attack found, and further guarantees proved of a corrected protocol. Let us now take a closer look, though without discussing particular proof commands.

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.

The assertion that nobody sends messages to themselves is proved by induction followed by use of the auto-tactic, which performs simplification and simple logical reasoning. If $evs \in \mathsf{otway}$ then

$$\forall A X.\, \mathsf{Says}\, A\, A\, X \notin \mathsf{set\_of\_list}\, evs.$$

5.1 Forwarding Lemmas

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

$$\frac{\mathsf{Says}\, A'\, B\, \{\!|N, \mathsf{Agent}\, A, \mathsf{Agent}\, B, X|\!\} \in \mathsf{set\_of\_list}\, evs}{X \in \mathsf{analz}(\mathsf{sees}\, lost\, \mathsf{Spy}\, evs)}$$

The proof is trivial. The spy sees the whole of the message; since X is transmitted in clear, analz will find it. So 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 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 case removes a layer of encryption. Here is its forwarding lemma:

$$\begin{array}{l}\mathsf{Says}\, \mathsf{S}\, B\, \{\!|Na, X, \mathsf{Crypt}\;\; K'\, \{\!|Nb, K|\!\}|\!\} \in \mathsf{set\_of\_list}\, evs \\ \qquad\qquad\qquad\qquad\qquad \Longrightarrow K \in \mathsf{parts}(\mathsf{sees}\, lost\, \mathsf{Spy}\, evs)\end{array}$$

These lemmas are trivial, but they are essential to the mechanization. Most inductive proofs require applying them in the appropriate cases.

5.2 Proving Regularity Lemmas

Statements of the form

$$X \in \mathsf{parts}(\mathsf{sees}\, lost\, \mathsf{Spy}\, evs) \longrightarrow \cdots$$

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. Try to prove the Fake case using laws proved for this purpose (§4.3).

5. Simplify all remaining cases.

A predefined tactic performs these tasks and returns any remaining subgoals.

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

$$\mathsf{Key}(\mathsf{shrK}\, A) \in \mathsf{parts}(\mathsf{sees}\, lost\, \mathsf{Spy}\, evs) \iff A \in lost.$$

The predefined tactic proves all but one case, and that is proved in one command by calling another tactic.

5.3 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 $evs \in \mathsf{otway}$):

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

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:

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

Here evs is some trace and, crucially, $A \notin lost$. 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.

5.4 Proving Secrecy Theorems

Section 3.7 discussed a theorem stating that the compromise of some session keys does not immediately compromise other keys: if K can be analyzed from a set of session keys and messages, then either it is one of those keys, or it can be analyzed from the messages alone. The precise statement of this theorem requires some explanation. As always, evs is some trace ($evs \in$ otway).

$$K \in \mathsf{analz}(\mathcal{K} \cup \mathsf{sees}\, lost\, \mathsf{Spy}\, evs)) \iff K \in \mathcal{K} \lor K \in \mathsf{analz}(\mathsf{sees}\, lost\, \mathsf{Spy}\, evs)$$

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

$$K \in \mathcal{K} \cup \mathsf{analz}(\mathsf{sees}\, lost\, \mathsf{Spy}\, evs)$$

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, I have discovered a number of 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 for the easy cases of symbolic evaluation of analz (pulling out agent names, nonces and compound messages) and performing automatic case splits on encrypted messages.

A tactic developed for the Fake case can often prove subgoals involving message forwarding. For the theorem mentioned above (concerning session keys), the remaining subgoals fall to the classical reasoner, one of Isabelle's automatic tools. Some other secrecy theorems require an explicit and detailed argument. Chief among these is proving that nonce *Nb* of the Yahalom protocol [9] remains secret, which requires establishing a correspondence between nonces and keys.

Common patterns of reasoning observed in these long proofs can often be packaged to form new proof tactics, shortening the script and facilitating future proofs.

A crucial secrecy theorem states that the protocol's step 3 distributes the session key only to A and B. It takes the following form. Let $evs \in$ otway and A, $B \notin lost$. Suppose that the server issues key K to A and B:

$$\begin{aligned}&\mathsf{Says\,S}\,B\,\{\!|Na, \mathsf{Crypt}(\mathsf{shrK}\,A)\{\!|Na, K|\!\},\\ &\qquad\qquad \mathsf{Crypt}(\mathsf{shrK}\,B)\{\!|Nb, K|\!\}|\!\} \in \mathsf{set\_of\_list}\,evs\end{aligned}$$

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

$$\mathsf{Says}\,B\,\mathsf{Spy}\,\{\!|Na, Nb, K|\!\} \notin \mathsf{set\_of\_list}\,evs$$

Then we have $K \notin \mathsf{analz}(\mathsf{sees}\,lost\,\mathsf{Spy}\,evs)$; the key is never available to the spy.[2]

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 then the session key is not fresh, contradiction; otherwise, the induction hypothesis yields the needed result. A single call to the classical reasoner executes this argument. 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 six commands and executes in about thirty seconds, generating a 1466-step proof.[3]

5.5 Proving Further Guarantees

The "step 3" secrecy theorem described above is worthless on its own. It holds of a protocol variant that can be attacked (§3.8). In the correct protocol, if A or B receive the expected nonce, then the server has indeed sent the critical step 3. How can we prove such guarantees for agents?

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. $B \to A : Na, \{\!|Na, Kab|\!\}_{Ka}$

Given step 3, B can only read the third component, which is encrypted with his key. He does not know what he is sending as step 4 or who will receive it.

[2]To show that K is also unavailable to any other agent C, simply apply the theorem putting $\{C\} \cup lost$ in place of $lost$: we can without loss of generality assume $C \in lost$, and the spy can decrypt everything intended for the agents in $lost$. This little corollary is the only reason why the set $lost$ is given as an argument to sees and otway. Declaring $lost$ as a constant would sacrifice the corollary but simplify the notation.

[3]All runtimes were measured on a Sun SuperSPARC model 61. A human could probably generate a much shorter proof by omitting irrelevant steps.

B's guarantee is as follows. If $B \notin \mathit{lost}$ and $B \neq \mathsf{Spy}$, and 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 recalls sending 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 "step 3" secrecy theorem can be applied. Checking his nonce assures B that K will be secure, subject to the premises of the secrecy theorem.

B's guarantee follows from a lemma proved by induction. It resembles a regularity lemma. Its main premise is

$$\mathsf{Crypt}(\mathsf{shrK}\, B)\{\!|Nb, \mathsf{Key}\, K|\!\} \in \mathsf{parts}(\mathsf{sees}\, \mathit{lost}\, \mathsf{Spy}\, \mathit{evs})$$

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 (§5.3).
- A nonce cannot be used both as Na and as Nb in two protocol runs. If $A \notin \mathit{lost}$ then the elements

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

  cannot both be in $\mathsf{parts}(\mathsf{sees}\, \mathit{lost}\, \mathsf{Spy}\, \mathit{evs})$.

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 can safely conclude upon receiving message 4. Suppose $A \notin \mathit{lost}$ and $A \neq \mathsf{Spy}$. 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

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

with a detailed consideration of how nonces can be used.

5.6 Proving a Simplified Protocol

Abadi and Needham [2] 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 [2, 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 half the size and runs in half the time.[4]

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 [9, page 247]. To Lowe [16], this represents a failure of authentication. The more refined analysis of Gollmann [11] lets us decide for ourselves whether such a limitation matters.

6 Conclusions

I have applied the approach to three versions of the Otway-Rees protocol:

1. a flawed version from Burrows et al. [9, page 247]

2. the same protocol, corrected by restoring the encryption of Nb

[4]From 348 to 142 seconds, and from 88 proof commands to 57.

3. a version that replaces much encryption by explicitness [2]

All three versions satisfy a crucial secrecy property: a key distributed by the server to two agents remains secure. They differ in their authenticity properties.

Version 1 is so weak as to be useless; the participants are not assured that the server has distributed the keys to the correct agents. Attempting to prove this assurance suggested a new attack (§3.8). Version 2 does assure its participants that keys are distributed correctly; the proof requires many lemmas that nonces uniquely identify certain messages. Version 3 gives its participants the same assurance, and the proof is much shorter, requiring no reasoning about unicity. However, version 2 satisfies a further property: it assures A that B is present.

Implementors must recognize the limitations of formal proofs. The model assumes strong encryption: the attacker can neither read nor modify messages. The model does not consider confusion between items of different types, say between keys and nonces. If distinct encrypted parts might be confused (because, say, their lengths are identical in bits), then labels must be inserted to distinguish them [2]. Redundancy may also be necessary to prevent attackers from exploiting algebraic properties of the encryption method. Proofs certify designs, but many attacks are directed against implementation flaws.

I have applied the inductive method to the three variants of the Otway-Rees protocol described here, to a recursive generalization of it, to two variants of Yahalom, to a simplified version of Woo-Lam [2], and to Needham-Schroeder (both shared- and public-key [21] versions).[5] Proofs are highly automated: one command can generate tens or hundreds of inferences, and small changes to protocols involve only small changes to proof scripts.

Bolignano [6] is developing a similar method. The similarity is most obvious in the realm of message analysis. His reduction relations for sets of messages are plainly related to my operators parts, analz and synth. Instead of formalizing traces, he precisely models the states of A, B and the spy, though the effect is similar. He has proved theorems concerning the Otway-Rees protocol using Coq [7].

The inductive approach is a valuable addition to the protocol analyzer's toolkit. A combination of tools may yield the best results. Using a belief logic during the design phase helps ensure freshness properties. Using a model-checker can find simple attacks quickly. Finally, the inductive approach allows deeper properties to be investigated with a modest amount of effort.

[5] Proof scripts are distributed with Isabelle, which can be obtained by ftp from URL `ftp://ftp.cl.cam.ac.uk/ml/index.html`; see subdirectory `HOL/Auth`.

Acknowledgement Conversations with D. Bolignano, A. Gordon, G. Huet, K. Wagner and especially R. Needham were invaluable. M. Abadi, G. Bella, B. Graham, G. Lowe, F. Massacci, V. Matyas, M. VanInwegen and the referees commented on drafts of this article. The research was funded by the EPSRC grant GR/K77051 "Authentication Logics" and by the ESPRIT working group 21900 "Types".

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*. ACM Press, 1997. In press.

[2] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Softw. Eng.*, 22(1):6–15, Jan. 1996.

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

[4] R. Anderson. Why cryptosystems fail. *Commun. ACM*, 37(11):32–40, Nov. 1994.

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

[6] D. 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.

[7] D. Bolignano. Formal verification of cryptographic protocols using Coq. Technical report, INRIA-Rocquencourt, 1996.

[8] S. H. Brackin. A HOL extension of GNY for automatically analyzing cryptographic protocols. In *9th Computer Security Foundation Workshop*, pages 62–75. IEEE Press, 1996.

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

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

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

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

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

[14] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

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

[16] G. Lowe. Some new attacks upon security protocols. In *9th Computer Security Foundations Workshop*, pages 162–169. IEEE Computer Society Press, 1996.

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

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

[19] L. C. Paulson. Tool support for logics of programs. In M. Broy, editor, *Mathematical Methods in Program Development: Summer School Marktoberdorf 1996*, NATO ASI Series F. Springer. In press.

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

[21] L. C. Paulson. Mechanized proofs of security protocols: Needham-Schroeder with public keys. Technical Report 413, Computer Laboratory, University of Cambridge, Jan. 1997.

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