Relations Between Secrets: Two Formal Analyses of the Yahalom Protocol

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

lcp@cl.cam.ac.uk

July 1997
Revised, June 1998 and November 1998
Minor revisions, January 2001

Abstract

The Yahalom protocol is one of those analyzed by Burrows et al. [5]. Based upon their analysis, they have proposed modifications to make the protocol easier to understand and to analyze. Both versions of Yahalom have now been analyzed using Isabelle/HOL. Modified Yahalom satisfies strong security goals, and the original version is adequate. The mathematical reasoning behind these machine proofs is presented informally. An appendix gives extracts from a formal proof.

Yahalom presents special difficulties because the compromise of one session key compromises other secrets. The proofs show that the resulting losses are limited. They rely on a new proof technique, which involves reasoning about the relationship between keys and the secrets encrypted by them. This technique is applicable to other difficult protocols, such as Kerberos IV [2].

The new proofs do not rely on a belief logic. They use a fundamentally different formal model: the inductive method. They confirm the BAN analysis and the advantages of the proposed modifications. The new proof methods detect more flaws than BAN and analyze protocols in finer detail, while remaining broadly consistent with the BAN principles. In particular, the proofs confirm the explicitness principle of Abadi and Needham [1]. The proofs also suggest that any realistic model of security must admit that secrets can become compromised over time.

Contents

1 Introduction

Many methods have been developed for mechanically analyzing cryptographic protocols. Some involve enumerating reachable states [7, 8], while others involve formal proof using general-purpose logics and tools [3, 13]. Authentication logics, designed specifically for security applications, have been popular. After the seminal BAN paper [5], numerous variants and extensions were put forward, some reaching commercial application [4]. But the BAN logic attracted criticism too—Mao and Boyd [10] is one example—and now appears to be losing favour. Below, we shall examine the two versions of the Yahalom protocol discussed in the original paper. Mechanized proofs performed using my inductive method confirm the BAN analysis.

The present case study is intended to approach realistic complexity. Although the Yahalom protocol has only four messages, it operates in an unusually subtle way. Moreover, the formal model is of a severely compromised network. Like most such models, it includes a spy who is in control of all communications. But it goes further: this intruder has taken control of some agents and can occasionally get hold of session keys. Even under these conditions, Yahalom provides adequate guarantees for the uncompromised parts of the system. These claims have been verified using the interactive theorem prover Isabelle [12].

What makes Yahalom subtle? Protocols such as Otway-Rees [11] distribute certificates, signed by a trusted authority, to their principals. Each principal typically receives a session key packaged with a nonce to ensure freshness. But in Yahalom, principal *B* receives two certificates. One contains a key but no evidence of freshness, while the other is signed using the same doubtful key. To accept the latter certificate as evidence of freshness for the key requires a convoluted argument. It relies on the secrecy of the nonce *Nb*, which is encrypted using the very key in question; that it still works is surprising.

The Yahalom protocol is largely of academic interest, but equally awkward protocols have been deployed. Kerberos version IV [2] uses session keys to encrypt other session keys. If one session key is compromised, many others could be lost. Despite this vulnerability, the protocol can be analyzed using essentially the same technique that proves the secrecy of *Nb* in Yahalom.

Burrows et al. [5] pointed out these features of Yahalom. They suggested that the protocol could be improved by including *B*'s nonce in the first certificate. The protocol becomes stronger, easier to analyze, and even more efficient, for some encryption can be removed.

The paper reviews the inductive approach (§2) and the Yahalom protocol (§3). It formalizes the protocol and discusses proofs of its basic properties (§§4–5). Establishing *B*'s guarantee requires several stages of reasoning (§§6–9). For the modified protocol, this guarantee is trivial is prove (§10). The paper examines authentication properties (§11) and concludes (§12). An appendix presents extracts from an Isabelle proof session.

2 Inductive Protocol Verification

Induction is the natural way to reason about security protocols. Such protocols work by preserving certain secrets and using them to establish new secrets. Expressed formally, reasoning of this sort is induction. Inductive definitions can be regarded as an abstract programming language that can easily express the actions of agents, whether honest or not.

The inductive approach models a protocol as the set of traces that could arise over time. Agents drawn from an infinite population may engage, playing various roles, in any number of possibly interleaved protocol runs. The formal definition resembles informal protocol notation, but contains additional rules to allow the empty trace, enemy action and accidental security breaches. Properties are proved by induction over this definition. If the inductive argument appears not to hold, one can easily identify the offending rule and the circumstances under which the desired property fails. Then one must generalize the induction formula, prove further lemmas to bridge the gap in the reasoning, or look for a weakness in the protocol.

The method benefits from mechanical support. Large expressions can arise in the course of a proof. However, the mechanical proofs are ideally suited to the tools in Isabelle. Protocols previously analyzed include versions of Bull's recursive authentication protocol [13], Kerberos [2] and the Internet protocol TLS [14].

Messages may contain agent names, nonces and keys. They may be built up using concatenation and encryption.

- Agent A, Agent B, ...
- Nonce Na, Nonce Nb, ...
- Key Ka, Key Kb, Key Kab ...
- $\{\!|X, X'|\!\}$ (concatenation)
- Crypt $K\,X$

Ordinary braces are reserved for set notation, so fat braces indicate nesting structure in messages. In informal descriptions, I often omit the tags Agent, Nonce and Key and write encryption as $\{\!|X|\!\}_K$. In the model, encrypted messages cannot be read without using the corresponding key. Encryption includes enough redundancy to ensure that $\{\!|X|\!\}_K = \{\!|X'|\!\}_{K'}$ implies $X = X'$ and $K = K'$, even if the plaintexts are just nonces; with modern cryptosystems, this can be done without compromising secrecy.

Assertions often concern sets $G, H, \ldots$ of messages, typically histories of past traffic. Operators over sets of message include

- parts H, the components of messages in H that could be obtained by breaking every encryption
- analz H, the components of messages in H that could be decrypted using only keys that can (recursively) be extracted from H
- synth H, the set of all messages that could be built up using messages in H as components

The Yahalom protocol uses symmetric-key encryption and a trusted server, S. Each agent A shares a long-term key with the server, written Ka or (more formally) $\mathsf{shrK}\ A$. Assertions typically involve the constant bad, denoting the set of agents who are controlled by the spy. Few guarantees can be expected of protocol runs with compromised agents: the spy can both read their traffic and sign messages in their name.

An *event* has the form $\mathsf{Says}\ A\ B\ X$ and represents an attempt by A to send B the message X. But B is not guaranteed to receive it and cannot know who sent it. If evs is a list of events then the set of message bodies seen by the spy is written $\mathsf{spies}\ evs$. This set includes the long-term keys of the agents in bad. The set of messages that the spy could invent, having monitored the traffic in evs, is

$$\mathsf{synth}(\mathsf{analz}(\mathsf{spies}\ evs)).$$

Confidentiality of the key or nonce M from the spy is stated as

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

The impossibility of M's appearing in traffic at all, even encrypted, is stated as

$$M \notin \mathsf{parts}(\mathsf{spies}\ evs).$$

More details of the inductive approach to protocol analysis are available elsewhere [13].

3 The Yahalom Protocol

This protocol, described by Burrows et al. [5, page 257], distributes a session key Kab to parties A and B with the help of a trusted authentication server. At the end of a run, each party can be sure that the other was recently present.

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

Now we can see in detail why Yahalom is problematical. When B receives the fourth message, he obtains a session key from the certificate $\{A, Kab\}_{Kb}$, but it does not mention Nb and could therefore be a replay of an old message. Freshness evidence comes from $\{Nb\}_{Kab}$, but why should B trust a certificate that is signed with an old, possibly compromised key?

The protocol is correct because Nb is kept secret. Only A could have formed $\{Nb\}_{Kab}$. So A associates Kab with the fresh nonce. Moreover, B learns that A has been active recently, a stronger outcome than with Otway-Rees.

Proving that Nb remains secret is harder than it looks. In an ideal model, one could prove that Kab always remains secret, and the secrecy of Nb would follow immediately for all runs between uncompromised agents. Such reasoning is faulty. It could 'verify' a version of Yahalom that sent Nb in clear in message 2:

2. $B \rightarrow \mathsf{S} : B, Nb, \{A, Na\}_{Kb}$

But this version can be attacked. Suppose an intruder I has managed to crack one of B's old certificates $\{A, K\}_{Kb}$, extracting the session key K. He can then masquerade as A, using *Nb* to forge message 4:

$$\begin{array}{ll} 1. & I_A \to B : A, Nc \\ 2. & B \to I_S : B, Nb, \{A, Nc\}_{Kb} \\ 4. & I_A \to B : \{A, K\}_{Kb}, \{Nb\}_K \end{array}$$

We must be realistic. Old session keys or nonces do sometimes leak out. The inductive model admits such accidents through the Oops rule (see below). In order to analyze the protocol, we must then examine the associations between session keys and nonces in the runs of a Yahalom trace. If a particular key *Kab* is secret then *Nb* is secret too, even if other keys or nonces are compromised. The proof is long and detailed—as it must be in a non-trivial model.

4 Formalization of the Protocol

Figure 1 displays the protocol definition in Isabelle syntax. The formalization follows the usual conventions of the inductive method. Traces are built in reverse order, and # is the list constructor. For example, `Says Spy B X # evs` is the trace whose last event is the spy's sending message X to B. The function used returns the set of all items present in a trace, so $N \notin \mathsf{used}\, evs$ asserts that N is fresh in the trace evs. The function set converts a list to the set of its elements, so $ev \in \mathsf{set}\, evs$ means that event ev has occurred at some point in trace evs.

The Fake rule is standard to all protocol descriptions. The spy may send any message he can generate to anybody. Such forged messages could ultimately cause security breaches.

The Oops rule hands the triple $\{Na, Nb, Kab\}$ to the spy. The precondition of this rule is that the server has actually used these nonces and key together in a run. The conclusion uses an event of the form $\mathsf{Notes\,Spy}\, X$, which makes X accessible to the spy. Oops is intended to model compromise of the session key by any means, from brute-force cracking to burglary. Including the nonces in the message identifies the run associated with the key. With Yahalom, however, the loss of *Nb* is itself a breach of security. Our guarantee to B will say that—provided the run involving *Nb* has not been compromised in this way—the session key is both fresh and secure.

The other rules concern messages 1 to 4. Calling the current trace `evs`i in the rule for message i identifies the subgoals in inductive proofs. As some disappear and others split into subcases, it is convenient to know which protocol rule relates to any subgoal.

5 Proving Basic Properties

Many of the protocol's properties are expressed and proved almost exactly like the analogous properties of Otway-Rees [13]. In particular, we must prove the *session key compromise theorem*. This conditional equation serves as a rewrite rule; it extracts

empty trace
```
[] ∈ yahalom
```

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

Message 1
```
[| evs1 ∈ yahalom;  A ≠ B;  Nonce NA ∉ used evs1 |]
⟹ Says A B {|Agent A, Nonce NA|} # evs1 ∈ yahalom
```

Message 2
```
[| evs2 ∈ yahalom;  B ≠ Server;  Nonce NB ∉ used evs2;
   Says A′ B {|Agent A, Nonce NA|} ∈ set evs2 |]
⟹ Says B Server
        {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|}
      # evs2 ∈ yahalom
```

Message 3
```
[| evs3 ∈ yahalom;  A ≠ Server;  Key KAB ∉ used evs3;
   Says B′ Server
        {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|}
     ∈ set evs3 |]
⟹ Says Server A
         {|Crypt (shrK A) {|Agent B, Key KAB, Nonce NA, Nonce NB|},
           Crypt (shrK B) {|Agent A, Key KAB|}|}
      # evs3 ∈ yahalom
```

Message 4
```
[| evs4 ∈ yahalom;  A ≠ Server;
   Says S A {|Crypt (shrK A) {|Agent B, Key K, Nonce NA, Nonce NB|},
              X|}  ∈ set evs4;
   Says A B {|Agent A, Nonce NA|} ∈ set evs4 |]
⟹ Says A B {|X, Crypt K (Nonce NB)|} # evs4 ∈ yahalom
```

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

Figure 1: Specifying the Yahalom Protocol

session keys from the scope of the analz operator. If evs is a trace and Kab is a session key[1] then

$$K \in \mathsf{analz}(\{Kab\} \cup \mathsf{spies}\, evs) \iff K = Kab \lor K \in \mathsf{analz}(\mathsf{spies}\, evs). \qquad (1)$$

The loss of one session key does not compromise other keys. This theorem is clearly of the greatest importance.

Proving it by induction requires first generalizing it from a single session key to an arbitrary set of them. If $\mathcal{K}$ is a set of session keys then

$$K \in \mathsf{analz}(\mathcal{K} \cup \mathsf{spies}\, evs) \iff K \in \mathcal{K} \lor K \in \mathsf{analz}(\mathsf{spies}\, evs). \qquad (2)$$

Some cases of the induction introduce new keys, so $\mathcal{K}$ cannot be kept constant during the proof.

Later we shall consider the security of Nb, where it appears necessary to prove an analogous but more complicated theorem about nonces. A similar theorem is needed to analyze Kerberos IV, and its corollaries include three special cases of the session key compromise theorem. Since Kerberos encrypts session keys using other session keys, the key compromise theorem does not hold in general [2].

The *session key secrecy theorem* states that the protocol is correct from the server's point of view. The session key given out in step 3 reaches only the agents named in that message. The theorem is proved in the usual way. Subgoals arising from induction are simplified with the help of theorem (1). Remaining subgoals are proved by routine reasoning about freshness, etc.

But this guarantee is not enough for A and B. In order to take advantage of it, they require assurance that the certificates they receive originated in a recent and correct server message. Only then can they trust the session key, for otherwise there could be attacks involving reuse of certificates [13].

The relevant guarantee for A states that if A is uncompromised and $\{B, Kab, Na, Nb\}_{Ka}$ occurs in a trace then that certificate originated with the server. The proof is a trivial induction. The argument is that only the server could have issued the certificate (by inspection of the protocol, if you like). The certificate contains all the information A needs: it confirms the name of the other party in the run, namely B, and the presence of nonce Na assures freshness. So A can trust the session key to be fresh and shared only with B.

6 Proving Guarantees for B

Half of B's guarantee resembles A's and is nearly as easy to prove. It states that if B is uncompromised and $\{A, Kab\}_{Kb}$ appears in a trace then the certificate originated in a server message of the form

$$\{B, Kab, Na', Nb'\}_{Ka}, \{A, Kab\}_{Kb}$$

[1] In the model, this merely means that Kab is not anybody's long-term shared key. No type distinction is made between long-term keys and session keys, even though the latter might be shorter in a real system.

for some Na' and Nb'. While A's certificate mentions the nonces used by the server, B's certificate mentions no nonces and conveys no information about how old it is.

Freshness guarantees can only come from B's other certificate. If $\{Nb\}_{Kab}$ appears in a trace then the server said something of the form

$$\{B', Kab, Na', Nb\}_{Ka}, \{A', Kab\}_{Kb}$$

for some A', B' and Na'—*provided* that Nb is secure from the spy. Thanks to its strong assumption about Nb, the guarantee is not hard to prove. Most of the cases of the induction are trivial. The spy cannot create $\{Nb\}_{Kab}$ because he does not know Nb, and honest agents do not try to. But this reasoning does not cover the message 4 case.

In this case, the most recent event—extending a trace evs—is an instance of message 4. Some agent A' sends a message containing $\{Nb'\}_{Kab'}$ as a component. By assumption, the critical message (namely $\{Nb\}_{Kab}$) is present in the extended trace. It may have been present in the original trace evs, in which case the induction hypothesis applies. Or, it may be the new element in the trace; we then have $Nb' = Nb$ and $Kab' = Kab$. Message 4 is only sent in response to a message containing what appears to be a valid certificate from the server. Since the certificate mentions Nb, and this nonce is secret from the spy, A' is uncompromised (otherwise the spy would be able to read the certificate). Therefore, the guarantee for A' holds, assuring that the certificate originated with the server.

The argument above is typical of proofs in the inductive method. The induction formula concerns one or two messages of certain forms. Call these the *critical messages*. Applying induction splits the formula into cases, one for the empty trace and others in which a trace evs is extended. The new event is an instance of a protocol message or Fake or Oops, and the induction hypothesis states that the formula holds for the original trace evs. If the latest event does not involve a critical message, then simplification can prove that case automatically. The Fake case is often trivial too, provided the induction formula contains enough assumptions to prevent the spy's forging the critical messages.

If a protocol step creates one of the critical messages, then explicit reasoning may be called for. Simplification sometimes reduces this case to the subcase where the critical message occurs in the latest event. Then we can use other facts to show that the desired property is preserved. Nonces and session keys play a major role in the reasoning. If a nonce has been assumed to be fresh then it must differ from the nonces present in old messages. Nonces created by honest agents identify message components uniquely. Such arguments become more complicated when the property refers to several critical messages. The Oops case can be the hardest of all; the loss of secrets particularly complicates proofs of confidentiality.

Now let us return to the Yahalom proof. Combining the guarantees for the two certificates meets B's requirements. The server never issues the same session key twice, so the two server messages involving Kab are identical, fixing the values of A, B and Nb. The nonce Na remains unconstrained, but that does not matter to B.

Figure 2 states the two main theorems of this section in Isabelle syntax, making the assumptions explicit and giving an impression of the formalism. Isabelle displays formulas using mathematical symbols similar to those shown, although ASCII equivalents

```
[| Crypt (shrK B) {|Agent A, Key K|} ∈ parts (spies evs);
   B ∉ bad;  evs ∈ yahalom |]
⟹ ∃NA NB. Says Server A
                  {|Crypt (shrK A) {|Agent B, Key K,
                                     Nonce NA, Nonce NB|},
                    Crypt (shrK B) {|Agent A, Key K|}|}
              ∈ set evs

[| Crypt K (Nonce NB) ∈ parts (spies evs);
   evs ∈ yahalom;
   Nonce NB ∉ analz (spies evs) |]
⟹ ∃A B NA. Says Server A
                   {|Crypt (shrK A) {|Agent B, Key K,
                                     Nonce NA, Nonce NB|},
                     Crypt (shrK B) {|Agent A, Key K|}|}
                 ∈ set evs
```

Figure 2: *B*'s Guarantees in Isabelle Syntax

appear in the proof scripts.[2] Both theorems are intended to be assurances for *B*. The first one refers to the first part of message 4, asserting that that the server distributed the key for *A* and *B*, but with unknown nonces. The second theorem uses the second part of *A*'s message to conclude that the server distributed the key quoting nonce *Nb*, assuming that *Nb* is secret from the spy. Satisfying this assumption turns out to be a major task.

7 The Associations Between Keys and Nonces

Proving that *Nb* remains secret may seem little different from proving that *Kab* remains secret—and that would be hard enough—but one detail makes it much harder. The direct analogues of laws (1) and (2) do not hold. Those laws essentially say that session keys are not normally used to encrypt other session keys. One might hope to prove that session keys are not normally used to encrypt nonces, but Yahalom's fourth message does precisely that.

A qualified form of (1) does hold. If the server has sent the message $\{B, Kab, Na, Nb'\}_{Ka}$ (thereby associating Nb' with *Kab*) and $Nb \neq Nb'$ then

$$Nb \in \mathsf{analz}(\{Kab\} \cup \mathsf{spies}\, evs) \iff Nb \in \mathsf{analz}(\mathsf{spies}\, evs).$$

The loss of *Kab* compromises at most one nonce, namely Nb'.

As before, proving this theorem by induction requires first generalizing it to an arbitrary set $\mathcal{K}$ of session keys. Its premise takes on a negative form. If the server says *no* message of the form $\{B, K, Na, Nb\}_{Ka}$ for $K \in \mathcal{K}$ and any *A*, *B* and *Na*, then

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

[2]Proof scripts for many protocols are distributed with Isabelle, which can be obtained from URL `http://www.cl.cam.ac.uk/Research/HVG/Isabelle/dist/`. See subdirectory `src/HOL/Auth`.

The right-to-left direction of the equivalence is trivial because analz is monotonic, so only the left-to-right direction is of any interest. But expressing the law as an equivalence (rather than as an implication) makes it available as a rewrite rule, which increases the degree of automation in the proof. Its script is still nine steps long. Message 4, which encrypts nonces, requires some attention; the guarantee for A is used to show that the server associated *Kab* with *Nb*. The Oops case involves similar reasoning, but matters can be arranged such that simplification proves it automatically.

In order to abbreviate some of the assertions, I have introduced a relation symbol for the association of K with *Nb* by some event in trace evs.

$$\mathsf{KeyWithNonce}\ K\ Nb\ evs \equiv \exists A\ B\ Na\ X.\ \mathsf{Says}\ \mathsf{S}\ A\ \{\!|\{\!|B, K, Na, Nb|\!\}_{Ka}, X|\!\} \in \mathsf{set}\ evs$$

We typically must show that $\neg$ KeyWithNonce $K\ Nb\ evs$ holds. It suffices either that the key is fresh or that the server has already associated the key with some other nonce. These two lemmas (shown in Isabelle syntax) can each be proved with a single command.

```
Key K ∉ used evs ⟹ ¬KeyWithNonce K NB evs

[| Says Server A
        {|Crypt (shrK A) {|Agent B, Key K, na, Nonce NB'|},   X|}
     ∈ set evs;
   NB ≠ NB';  evs ∈ yahalom |]
⟹ ¬KeyWithNonce K NB evs
```

Here are the Isabelle forms of this section's main theorems. The variable KK was written as $\mathcal{K}$ above. The premise `KK ⊆ -(range shrK)` expresses that KK is a set of session keys: it contains no long-term keys. Also, insert x A denotes $\{x\} \cup A$.

```
[| evs ∈ yahalom;
   KK ⊆ -(range shrK);
   ∀K∈ KK. ¬KeyWithNonce K NB evs |]
⟹ (Nonce NB ∈ analz (Key''KK ∪ (spies evs))) =
    (Nonce NB ∈ analz (spies evs)))

[| Says Server A
        {|Crypt (shrK A) {|Agent B, Key KAB, na, Nonce NB'|}, X|}
     ∈ set evs;
   NB ≠ NB';  KAB ∉ range shrK;  evs ∈ yahalom |]
⟹ (Nonce NB ∈ analz (insert (Key KAB) (spies evs))) =
    (Nonce NB ∈ analz (spies evs))
```

8 Proving Secrecy of *Nb*

We are finally in a position to tackle the secrecy theorem for *Nb*. Suppose a trace contains an instance of message 2:

$$2.\quad B \rightarrow \mathsf{S} : B, \{\!|A, Na, Nb|\!\}_{Kb}$$

If A and B are uncompromised then *Nb* is secure from the spy—*provided* no Oops event involving *Na* and *Nb* occurs in the trace. We must exclude Oops events for all possible keys because B does not know which session key will be generated. B's final guarantee also requires this strong assumption.

The proof script is long by Isabelle standards (19 commands). Some cases of the induction are straightforward. The Fake case is trivial: if the spy does not have Nb already, nothing he says can get it immediately. The message 1 case is also easy: the only nonce in it is fresh, and cannot therefore be the nonce we are concerned about. The message 2 case splits into three subcases: either the nonce is fresh, or it is an instance of Na (which the spy would have seen already), or the message does not mention our Nb. The message 3 case concerns a new server message containing nonces Na' and Nb', and we must deal with the possibilities $Nb = Na'$ and $Nb = Nb'$.

- $Nb = Na'$ is impossible because honest agents choose fresh nonces. Since Na' was chosen in an instance of message 1, and Nb was chosen in an instance of message 2, they were chosen at different times. If these nonces are equal then freshness has not been observed.

- $Nb = Nb'$ implies (because B never reuses nonces), that the server message arose from the very instance of message 2 mentioned in the theorem statement. This coincidence allows an appeal to the induction hypothesis.

The message 4 case is more difficult. Here, a message of the form $\{Nb'\}_K$ is sent by some agent, A'. We must contend with the possibility that $Nb = Nb'$. A long chain of reasoning ensues. Since Nb' has been kept secure up to now, agent A' must be honest. The initiator's guarantee tells us that the server must have issued Na, Nb and K together. (Again, part of the reasoning is that an honest agent uses a nonce at most once.) By the assumption that no Oops events have occurred involving Na and Nb, we may appeal to the secrecy theorem for session keys: the key is secure and so the new message $\{Nb\}_K$ does not betray Nb.

The Oops case involves more intricate reasoning of this sort. A session key and its associated nonces are lost. There are two cases. If the Oops event betrays the particular Nb in question, then it must involve the particular Na also, but the theorem statement assumes such events not to occur. If it betrays a different value of Nb, then the loss of the session key is irrelevant. Oops is the only case to require the theorem about nonces elaborately proved in §7; we again see that Oops is essential to a realistic treatment of Yahalom.

Reverting to Isabelle notation, here are the two 'unicity lemmas' used in the proof above. The nonce Nb uniquely identifies the ciphertext in which it appears, provided the encrypting agent is honest. A nonce value is never used both as Na and as Nb, provided it is unknown to the spy.

```
[| Crypt (shrK B) {|Agent A, Nonce NA, nb|} ∈ parts(spies evs);
   Crypt (shrK B') {|Agent A', Nonce NA', nb|} ∈ parts(spies evs);
   evs ∈ yahalom;  B ∉ bad;  B' ∉ bad |]
⟹ NA = NA' & A = A' & B = B'

[| Crypt (shrK B') {|Agent A', Nonce NB, nb'|} ∈ parts(spies evs);
   evs ∈ yahalom;  Nonce NB ∉ analz(spies evs) |]
⟹ Crypt (shrK B) {|Agent A, na, Nonce NB|} ∉ parts(spies evs);
```

And here is the main theorem of this section, that nonce `NB` remains safe from the spy.

```
[| (∀k. Notes Spy {|Nonce NA, Nonce NB, k|} ∉ set evs);
   Says B Server
        {|Agent B, Crypt (shrK B){|Agent A, Nonce NA, Nonce NB|}|}
     ∈ set evs;
     A ∉ bad;  B ∉ bad;  evs ∈ yahalom |]
⟹ Nonce NB ∉ analz (spies evs)
```

9 A Session Key Theorem for *B*

Now, all the pieces fit together. Recall the three lemmas proved for *B*:

- The session key found in the first certificate originated with the server. (But how long ago?)
- Provided *Nb* has been kept secret from the spy, the session key is recent: the server bound it together with *Nb*.
- Under certain conditions, nonce *Nb* indeed remains secret.

Combining these results yields *B*'s overall guarantee for the session key. If *B* has issued message 2 mentioning *A*, *Na* and *Nb*, and receives $\{A, K\}_{Kb}$ and $\{Nb\}_K$ at the end of a run, and *A* and *B* are uncompromised, and no Oops event involving *Na* and *Nb* has occurred, then the server has issued a valid instance of message 3 involving the particular *A*, *B*, *Na*, *Nb* and *K*. Giving this conclusion to the session key secrecy theorem tells us that *K* is known only to *A* and *B*.

Here is the full statement of the theorem, using Isabelle's syntax.

```
[| Says B Server
     {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|}
     ∈ set evs;
   Says A' B {|Crypt (shrK B) {|Agent A, Key K|},
               Crypt K (Nonce NB)|} ∈ set evs;
   ∀k. Notes Spy {|Nonce NA, Nonce NB, k|} ∉ set evs;
   A ∉ bad;  B ∉ bad;  evs ∈ yahalom |]
⟹ Says Server A
            {|Crypt (shrK A) {|Agent B, Key K,
                      Nonce NA, Nonce NB|},
              Crypt (shrK B) {|Agent A, Key K|}|} ∈ set evs
```

Guarantees for other protocols typically have a weaker Oops assumption, ruling out the single Oops event $\{Na, Nb, K\}$. But the guarantee above cannot be so strengthened. Suppose the Oops event did occur for $\{Na, Nb, Kab\}$, where *Kab* was the true session key issued by the server. Then the spy, if he possessed an old key *K* and an old certificate $\{A, K\}_{Kb}$ for it, could form $\{Nb\}_K$ and fool *B* into accepting *K* instead of *Kab*. The protocol is thus slightly weaker than one might hope.

10 Analysis of Modified Yahalom

My modification of Yahalom incorporates some suggestions from the BAN paper [5, page 259]. The new protocol is stronger and much easier to reason about. The server includes *Nb* in the first certificate. The nonce is no longer kept secret.

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

An elementary induction now suffices to prove that B may trust Kab. If B is uncompromised and $\{A, B, Kab, Nb\}_{Kb}$ appears in a trace, then that certificate originated with the server. After verifying his nonce (Nb), B may safely conclude that Kab is fresh. It is also secret, subject to the conditions of the session key secrecy theorem, namely that A is uncompromised and the Oops event $\{Na, Nb, Kab\}$ has not occurred. Note that the Oops assumption has been relaxed to refer to one specific session key.

Here is B's guarantee in Isabelle notation. It is almost identical to the first theorem presented in §6 except that B's nonce (NB) now occurs in the assumptions and conclusion.

```
[| Crypt (shrK B) {|Agent A, Agent B, Key K, Nonce NB|}
     ∈ parts (spies evs);
   B ∉ bad;  evs ∈ yahalom |]
⟹ ∃NA. Says Server A
           {|Nonce NB,
             Crypt (shrK A){|Agent B, Key K, Nonce NA|},
             Crypt (shrK B){|Agent A, Agent B, Key K, Nonce NB|}|}
         ∈ set evs
```

The BAN suggestions shorten the proofs by almost half, eliminating the lengthy reasoning about Nb.[3] I kept no records of the human effort required, but original Yahalom required much more work than the modified version.

The inductive method and the BAN logic work in fundamentally different ways. The former models traces using standard predicate logic and set theory; it requires long, detailed proofs. The latter is a specialized logic of belief; it works at a high level of abstraction. For the two methods to agree on the defects of the original Yahalom protocol and the virtues of the modified version suggests that the BAN analysis is broadly correct. There is similar agreement with Abadi and Needham [1] concerning the Otway-Rees protocol: applying their suggestions reduces the Isabelle proof script by 40% [13]. Such findings add support to their explicitness principle.

Shorter proofs are advantageous provided they come from a better design, rather than from an unrealistic formalism. Ease of analysis is obviously important in any technology, and especially in those concerning safety or security. Extra complexity could hide a flaw. With the recursive authentication protocol, replacing encryption by hashing introduced several vulnerabilities [15]. In the case of Yahalom, the same complexity appears in the BAN analysis, the inductive analysis, and at the informal level. We must conclude that this complexity is not an artifact of the analysis method, but is intrinsic to the protocol design.

[3]The full proof script for original Yahalom has 132 commands and runs in 87 seconds on a 300Mhz Pentium Pro. That for modified Yahalom has 68 commands and runs in 53 seconds.

Of course, the BAN logic is not always right. The original paper's advice on Otway-Rees admitted catastrophic breaches of security [10, 13]; an intruder could masquerade as other users at will. The BAN version of Yahalom has a minor security flaw. Its two certificates have identical formats:

$$\{B, Kab, Na\}_{Ka} \quad \{A, Kab, Nb\}_{Kb}$$

A weak middleperson attack, in which the spy passes off one certificate as the other one, lets the spy masquerade as another agent. He does not get hold of the session key but still violates one of the protocol's objectives: to assure each party of the other's presence.[4] Moreover, the possibility of exchanging the certificates seriously complicates the reasoning. To eliminate this danger, I have applied the explicitness principle: my version of the protocol includes B's name in the second certificate.

11 Proving Authentication

Related protocols such as Needham-Schroeder (shared key version) and Otway-Rees merely distribute session keys. Yahalom, however, confirms to each party that the other was present recently. This information is of limited value—a principal can, after all, fail at any time—but it may have specialized uses.

Proving authentication of B to A is easy. If B is uncompromised, then only B can perform the encryption used in message 2. If A receives the certificate $\{B, Kab, Na\}_{Ka}$ then B has created the message $\{A, Na\}_{Kb}$ and thus has been active since Na was invented. This guarantee is expressed in terms of the messages of modified Yahalom, but similar theorems can be proved for the original protocol using similar proof scripts.

The authentication of A to B makes use of A's final message, $\{Nb\}_{Kab}$. Intuitively, B reasons that only A could know Kab and perform this encryption. The inductive proof is not difficult. In the message 3 case, where B's certificate is first created, we note that $\{Nb\}_{Kab}$ could not yet exist: Kab is too fresh. In the message 4 case, where something of the form $\{Nb'\}_{Kab'}$ is created, we can by the induction hypothesis assume $Nb' = Nb$ and $Kab' = Kab$; applying the protocol's guarantees for both parties confirms the identities of A and B.

This inductive argument assumes that Kab is secure from the spy. The resulting lemma is then combined with the session key secrecy theorem to obtain B's guarantee.

12 Conclusions

Yahalom is far from being the most complicated protocol to be analyzed formally. But it shows surprising complexity, particularly when examined in a model that formalizes the possibility of accidents. Using Isabelle/HOL, I have proved that the protocol satisfies its main objectives. The machine proofs can be explained in terms of intuitive arguments about the potential impact of sending a protocol message.

[4] Syverson [16] describes two attacks on BAN-Yahalom. Attack 2 is the one described above. Attack 1 involves passing off the concatenation of two nonces as a single nonce, which is impossible in my model because of the length discrepancy. Unaware of Syverson's paper, I rediscovered attack 2 while looking at failing proofs.

Simple modifications to the protocol eliminate the complexity and allow the proof scripts to be greatly shortened. Advice derived from the BAN logic has thus been confirmed experimentally using an intrinsically different formal method, the inductive approach. The only flaw in the BAN analysis concerns the risk of exchanging the two certificates.

The agreement between the two analyses suggests that the underlying concepts are valid. In a frequently-cited paper, Abadi and Needham present principles for protocol design, commenting ‘We arrived at our principles by noticing some common features among protocols that are difficult to analyze’ [1, page 6]. As new analysis methods supplant old ones, we have reason to expect that their principles will remain stable.

It is not clear whether some of the methods replacing authentication logics support such a detailed analysis. Model-checking [9] is fully automatic, while my methods require much effort. Model-checking is highly successful at finding attacks, and could probably be applied to Yahalom. But the failure to find an attack does not explain why the protocol works, which would be valuable feedback for designers.

Yahalom’s complexity comes from the relation between the two secrets, *Kab* and *Nb*, which are distributed separately to B. The formal proofs for Yahalom demonstrate how to analyze Kerberos [2] and presumably other protocols involving relationships between secrets.

Acknowledgement Discussions with Martín Abadi and Roger Needham were helpful. Fabio Massacci commented extensively on a draft. The research was funded by the EPSRC, grants GR/K77051 ‘Authentication Logics’ and GR/K57381 ‘Mechanizing Temporal Reasoning.’

A Extracts from the Nb Secrecy Proof

To gain an impression of how protocols are proved by machine, let us look at part of the proof that Nb remains secret. The full text of the session is much too large to include in full, so we shall concentrate on the case where the most recent event is an instance of message 4.

We start the proof by giving the goal to Isabelle. To allow the induction to go through, it is expressed slightly differently from the form shown in §8 above.

```
[| A ∉ bad; B ∉ bad; evs ∈ yahalom |]
⟹ Says B Server
     {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|}
    ∈ set evs →
    (∀k. Notes Spy {|Nonce NA, Nonce NB, k|} ∉ set evs) →
    Nonce NB ∉ analz (spies evs)
```

Next, we apply induction, yielding seven subgoals. Only the goal corresponding to message 4 is shown. A message between some agents `Aa` and `Ba` extends some existing trace, `evs4`.

```
[| A ∉ bad; B ∉ bad; evs4 ∈ yahalom;
   Says B Server
    {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|}
   ∈ set evs4 →
   (∀k. Notes Spy {|Nonce NA, Nonce NB, k|} ∉ set evs4) →
   Nonce NB ∉ analz (spies evs4);
   Aa ≠ Server;
   Says S Aa
    {|Crypt (shrK Aa) {|Agent Ba, Key K, Nonce NAa, Nonce NBa|}, X|}
   ∈ set evs4;
   Says Aa Ba {|Agent Aa, Nonce NAa|} ∈ set evs4 |]
⟹ Says B Server
     {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|}
    ∈ set (Says Aa Ba {|X, Crypt K (Nonce NBa)|} # evs4) →
    (∀k. Notes Spy {|Nonce NA, Nonce NB, k|}
          ∉ set (Says Aa Ba {|X, Crypt K (Nonce NBa)|} # evs4)) →
    Nonce NB
    ∉ analz (spies (Says Aa Ba {|X, Crypt K (Nonce NBa)|} # evs4))
```

The next steps are the application of forwarding lemmas [13] and rewriting. Then the classical reasoner (`blast_tac`) is invoked with suitable lemmas, automatically proving the Fake case and the cases for messages 1–3. The only cases left are those for Oops and message 4, which after further simplification is

```
[| A ∉ bad; B ∉ bad; evs4 ∈ yahalom; Aa ≠ Server;
   Says S Aa
    {|Crypt (shrK Aa) {|Agent Ba, Key K, Nonce NAa, Nonce NB|}, X|}
   ∈ set evs4;
   Says Aa Ba {|Agent Aa, Nonce NAa|} ∈ set evs4;
   X ∈ analz (spies evs4);
   Says B Server
    {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|}
   ∈ set evs4;
   ∀k. Notes Spy {|Nonce NA, Nonce NB, k|} ∉ set evs4;
   Key K ∈ analz (spies evs4); Nonce NB ∉ analz (spies evs4) |]
⟹ False
```

Let's examine the assumptions at this point. Agents `A` and `B` are uncompromised. The agent, `Aa`, who is distinct from the `Server`, has received what appears to be a certificate containing the session key `K`. The last two assumptions state that `K` is known to the spy but that `NB` is not. (Simplification has automatically dealt with the case where `K` is not known to the spy.)

Observe that agent `Aa` must be uncompromised: `NB` is safe despite having appeared in traffic encrypted by `Aa`'s key. Calling the specialized tactic `not_bad_tac` performs this reasoning, adding a new assumption at the end.

```
[| A ∉ bad; B ∉ bad; evs4 ∈ yahalom; Aa ≠ Server;
   Says S Aa
    {|Crypt (shrK Aa) {|Agent Ba, Key K, Nonce NAa, Nonce NB|}, X|}
   ∈ set evs4;
   Says Aa Ba {|Agent Aa, Nonce NAa|} ∈ set evs4;
   X ∈ analz (spies evs4);
```

```
     Says B Server
      {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|}
     ∈ set evs4;
     ∀k. Notes Spy {|Nonce NA, Nonce NB, k|} ∉ set evs4;
     Key K ∈ analz (spies evs4); Nonce NB ∉ analz (spies evs4);
     Aa ∉ bad |]
  ⟹ False
```

Since Aa is uncompromised, any certificate sealed with Aa's key is authentic. A lemma (proved earlier by a trivial induction) tells us that the certificate originated with the Server.

```
  [| A ∉ bad; B ∉ bad; evs4 ∈ yahalom; Aa ≠ Server;
     Says Aa Ba {|Agent Aa, Nonce NAa|} ∈ set evs4;
     X ∈ analz (spies evs4);
     Says B Server
      {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|}
     ∈ set evs4;
     ∀k. Notes Spy {|Nonce NA, Nonce NB, k|} ∉ set evs4;
     Key K ∈ analz (spies evs4); Nonce NB ∉ analz (spies evs4);
     Aa ∉ bad;
     Says Server Aa
      {|Crypt (shrK Aa) {|Agent Ba, Key K, Nonce NAa, Nonce NB|},
        Crypt (shrK Ba) {|Agent Aa, Key K|}|}
     ∈ set evs4 |]
  ⟹ False
```

Another lemma, also proved by a previous induction, tells us that the Server would only have issued such a message in response to a valid instance of message 2.

```
  [| A ∉ bad; B ∉ bad; evs4 ∈ yahalom; Aa ≠ Server;
     Says Aa Ba {|Agent Aa, Nonce NAa|} ∈ set evs4;
     X ∈ analz (spies evs4);
     Says B Server
      {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|}
     ∈ set evs4;
     ∀k. Notes Spy {|Nonce NA, Nonce NB, k|} ∉ set evs4;
     Key K ∈ analz (spies evs4); Nonce NB ∉ analz (spies evs4);
     Aa ∉ bad;
     Says Server Aa
      {|Crypt (shrK Aa) {|Agent Ba, Key K, Nonce NAa, Nonce NB|},
        Crypt (shrK Ba) {|Agent Aa, Key K|}|}
     ∈ set evs4;
     ∃B'. Says B' Server
            {|Agent Ba,
              Crypt (shrK Ba) {|Agent Aa, Nonce NAa, Nonce NB|}|}
           ∈ set evs4 |]
  ⟹ False
```

We now see that NB appears in two instances of message 2. Since it is unknown to the spy, it must have been created by an honest agent, who would use nonces uniquely. This identifies the message components: agents Aa and A are the same, as are Ba and B. So the message from the Server is to two honest agents. By the session key secrecy theorem, the key it holds is safe from the spy, contradicting the assumption Key K ∈ analz (spies evs4). A call to blast_tac finds this argument, proving the message 4 case.

A further six commands are needed to prove the Oops case. Space limitations prevent our discussing the proof in detail, but a peek at the simplified subgoal is instructive:

```
[| A ∉ bad; B ∉ bad; evso ∈ yahalom;
   Says Server Aa
    {|Crypt (shrK Aa) {|Agent Ba, Key K, Nonce NAa, Nonce NBa|}, X|}
   ∈ set evso;
   Says B Server
    {|Agent B, Crypt (shrK B) {|Agent A, Nonce NA, Nonce NB|}|}
   ∈ set evso;
   NB = NBa → NA ≠ NAa;
   (∀k. Notes Spy {|Nonce NA, Nonce NB, k|} ∉ set evso);
   Nonce NB ∉ analz (spies evso) |]
⟹ NB ≠ NAa &
    NB ≠ NBa & Nonce NB ∉ analz (insert (Key K) (spies evso))
```

To show that `NB` remains secure, we must show that it differs from the two nonces mentioned in the Oops event, namely `NAa` and `NBa`. Finally, we must show that `NB` is not compromised by the loss of the session key `K`. The proof requires the theorem proved in §7.

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] Giampaolo Bella and Lawrence C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In J.-J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann, editors, *Computer Security — ESORICS 98*, LNCS 1485, pages 361–375. Springer, 1998.

[3] Dominique Bolignano. Towards the formal verification of electronic commerce protocols. In Computer Security Foundations Workshop [6], pages 113–147.

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

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

[6] *10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

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

[8] 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. Springer, 1996.

[9] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.

[10] 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.

[11] Dave Otway and Owen Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

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

[13] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[14] Lawrence C. Paulson. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Information and System Security*, in press.

[15] Peter Y. A. Ryan and Steve A. Schneider. An attack on a recursive authentication protocol: A cautionary tale. *Information Processing Letters*, 65(1):7–10, January 1998.

[16] Paul Syverson. A taxonomy of replay attacks. In *7th Computer Security Foundations Workshop*, pages 187–191. IEEE Computer Society Press, 1994.