Kerberos Version IV: Inductive Analysis of the Secrecy Goals

Giampaolo Bella and Lawrence C Paulson

Computer Laboratory – University of Cambridge
New Museums Site, Pembroke Street
Cambridge CB2 3QG (UK)
Giampaolo.Bella@cl.cam.ac.uk, Larry.Paulson@cl.cam.ac.uk

Abstract. An operational model of crypto-protocols is tailored to the detailed analysis of the *secrecy* goals accomplished by Kerberos Version IV. The model is faithful to the specification of the protocol presented by the MIT technical plan [14] — e.g. timestamping, double session key delivery mechanism are included. It allows an eavesdropper to exploit the shared keys of compromised agents, and admits the accidental loss of expired session keys. Confidentiality is expressed from the viewpoint of each party involved in a protocol run, with particular attention to the assumptions the party relies on. If such assumptions are unrealistic, they highlight weaknesses of the protocol. This is particularly so from the viewpoint of the responder: the model suggests and proves a reasonable correction.

Keywords: secrecy, secure key, non-expired timestamp, inductive method, machine proof.

1 Overview

Crypto-protocols are highly error-prone, but formal methods can be used to limit the risks deriving from their execution. Different kinds of methods could be used in combination to achieve the best results.

Burrows et al. [6] developed a *belief logic*, i.e. a modal logic suitable to express the beliefs of parties, and analysed the authenticity properties of several classical protocols. Because of the failure to discover errors found by other methods, many extensions and variations have been proposed (e.g. [5]).

State enumeration deals instead with systems of limited size, and checks that all the states reachable are safe. This has led to several encouraging results (e.g. [8, 11, 12, 20]).

Another approach relying on the simple concept of *mathematical induction* comprises few methods that enhance state enumeration by inductive features (e.g [13]), and a purely inductive one. This method, simply called the "inductive method" in the sequel, deals with systems of infinite size and is based on theorem proving. It has achieved promising results with classical protocols such as Otway-Rees, Needham-Schroeder and Yahalom [17, 18], and with real-world protocols such as the Internet protocol TLS [19] and Kerberos.

The work on Kerberos was started by the authors last summer, and has gone through several stages. Several technical results about the Version IV[1] of the protocol were proven soon [1], but the analysis of its secrecy goals turned out to be a major task because of the mechanism that delivers session keys.

The subsequent analysis [2] of the simpler BAN version of the same protocol (i.e. the version presented by Burrows et al. [6]) allowed the development of suitable proof strategies thanks to which the formal analysis of the secrecy goals of Version IV has recently been achieved. This work shows a strong protection over non-expired session keys even from attacks that exploit expired ones. However, the risk of loss of expired session keys weakens Bob's confidentiality guarantees. To strengthen them, we suggest to add a simple temporal check to the operation of the trusted party of the protocol, and prove it to be an efficacious cure.

It has to be mentioned that Version IV is much harder to analyse than the BAN version. The mechanisation of the former executes in six times the CPU time required by the mechanisation of the latter.

Section 2 sketches the main concepts of the inductive method. Kerberos Version IV is introduced in Section 3 and analysed in Section 4. Some related work is mentioned in Section 5, and Section 6 concludes.

2 Inductive Method Reminder

Crypto-protocols aim at keeping secret certain pieces of information in order to infer the secrecy of new ones. This can be naturally expressed by mathematical induction.

The inductive method models a crypto-protocol as the set of all possible traces of events deriving from its execution by an infinite number of agents. The basic event has the form $\mathsf{Says}\, A\, B\, msg$. In the real world, networks are often threatened by an eavesdropper trying to access resources he is not entitled to get. A dishonest agent *spy* is therefore included in the model. He controls a set bad of compromised agents. Besides, honest agents can leak by accident valuable information such as session keys.

Security properties typically concern sets of messages. Three operators map a given set H of messages in another such set:

$\mathsf{parts}\, H$ yields all information contained in H, components of compound messages, and bodies of all encrypted messages;

$\mathsf{analz}\, H$ yields all information accessible in H, components of compound messages, and bodies of messages encrypted under keys (recursively) extracted from H;

$\mathsf{synth}\, H$ yields all compound messages and encrypted messages that can be built using elements of H as components.

The spy's ability of monitoring the network traffic is formalised by $\mathsf{spies}\, evs$, inductively defined as the set of messages over the trace evs — the traffic over evs

[1] Version IV is the "original" Kerberos. Version V is based on the same message structure.

— plus the shared keys of compromised agents. Therefore, from the observation of the trace *evs*, the spy can send messages belonging to the set

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

Confidentiality of a key K is formally stated in terms of the analz operator

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

while K not appearing in traffic at all, even encrypted, is formalised in terms of parts

$$K \notin \mathsf{parts}(\mathsf{spies}\, evs)$$

Encryption is assumed to be safe, i.e. bodies of encrypted messages can not be read without knowing the corresponding key.

Guarantees are expressed in form of theorems mechanised by Isabelle [16]. The inductive method is described in greater detail elsewhere [17].

3 Kerberos Version IV Overview

The development of Kerberos started during the mid 1980s within project Athena at MIT. After three trial versions, Version IV was released in 1989 [14].

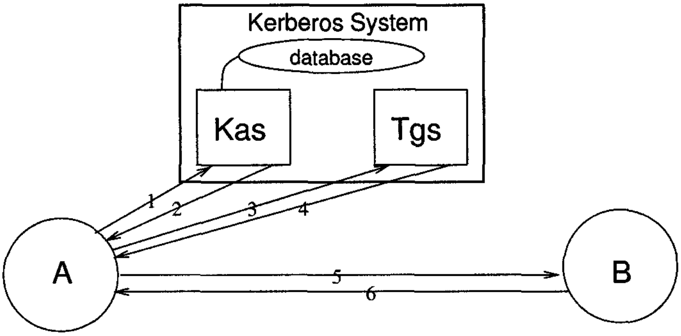

Fig. 1. Basic Kerberos Layout

Kerberos is a shared key protocol based on timestamps. Its basic layout is shown in Fig. 1. Kerberos is meant to provide secure communication over a local area network (LAN) and its use has become more and more widespread during this decade. However, we believe that the actual protocol has not yet been formally analysed extensively (see also Sec. 5). This might be due to the double authentication procedure that the initiator of the protocol has to go through, a feature that seems to provide reassuring guarantees by inspection, but also makes its formal analysis a lot harder.

Shared-key protocols normally rely on a trusted third party: Kerberos relies on two. The first to take part to the protocol execution is the *Kerberos Authentication Server* (abbreviated by Kas), the second is the *Ticket Granting Server* (abbreviated by Tgs). A database contains the shared keys of all network users and only Kas can access it.

The protocol in Fig. 2 is faithfully quoted from the MIT Athena technical plan [14]. Lifetimes are omitted because sending them over the network does not enhance secrecy, as pointed out by Bellovin and Meritt [4]. Kas does not need a shared key of its own, because it uses Alice's shared key looked up in the Kerberos database to communicate with her. Kas knows the shared key of Tgs.

I. AUTHENTICATION

1. $A \rightarrow \mathsf{Kas} : A, \mathsf{Tgs}, Ta1$
2. $\mathsf{Kas} \rightarrow A : \{AuthKey, \mathsf{Tgs}, Tk, \underbrace{\{A, \mathsf{Tgs}, AuthKey, Tk\}_{Ktgs}}_{AuthTicket}\}_{Ka}$

II. AUTHORISATION

3. $A \rightarrow \mathsf{Tgs} : \underbrace{\{A, \mathsf{Tgs}, AuthKey, Tk\}_{Ktgs}}_{AuthTicket}, \underbrace{\{A, Ta2\}_{AuthKey}}_{authenticator}, B$
4. $\mathsf{Tgs} \rightarrow A : \{ServKey, B, Tt, \underbrace{\{A, B, ServKey, Tt\}_{Kb}}_{ServTicket}\}_{AuthKey}$

III. SERVICE

5. $A \rightarrow B : \underbrace{\{A, B, ServKey, Tt\}_{Kb}}_{ServTicket}, \underbrace{\{A, Ta3\}_{ServKey}}_{authenticator}$
6. $B \rightarrow A : \{Ta3 + 1\}_{ServKey}$

Fig. 2. Kerberos Version IV

The AUTHENTICATION phase (messages 1 and 2) sees Alice logging onto a workstation in order to access the network. Alice sends her identifier to Kas and gets in reply a session key and an encrypted ticket, respectively called *authkey* and *authticket* in the sequel, which will be used in the next phase. They are sent encrypted under Alice's shared key that Kas has retrieved from the database. Alice can not decrypt the authticket as this is encrypted under the shared key of Tgs. The authkey has a lifetime of several hours: Alice is automatically logged out when this key expires.

The AUTHORISATION phase (messages 3 and 4) occurs each time Alice wants to access the network resource Bob. Alice presents the authticket to Tgs together with an *authenticator* to show that the authticket was issued to her. Tgs issues her with a new session key and a new ticket, respectively called *servkey* and *servticket* below. The servkey has a short lifetime of few minutes, and the servticket is unintelligible to Alice, being encrypted under Bob's shared key.

The SERVICE phase (messages 5 and 6) follows each authorisation phase. Alice presents the servticket to Bob along with a new authenticator. Bob's reply is borrowed from the Needham-Schroeder protocol.

4 Kerberos Version IV Inductive Analysis

We define the function

$$\mathsf{Ct} : \textit{event list} \longrightarrow \textit{bool}$$

expressing the current time over a given trace, for creating timestamps. It is defined as the *length* of a trace, since traces never shrink.

Four natural numbers formalise the lifetimes Kerberos relies on.

1. AuthLife is the lifetime of the authkey. Tgs checks the authkey against this lifetime before issuing Alice with a servkey.
2. ServLife is the lifetime of the servkey. Bob checks the servkey to have not expired w.r.t. this lifetime before setting up a communication with Alice.
3. AutcLife is the lifetime of any *authenticator*. Both Tgs and Bob check the authenticator they receive to have not expired w.r.t. this lifetime in order to prevent the replay of past authenticators.
4. RespLife is the lifetime of any *server response*. Once Alice has contacted Kas or Tgs, she checks their replies to be not late w.r.t. this lifetime. A late reply would indicate some messages of the communication to be possibly faked.

The first three lifetimes were introduced by the Athena Technical Plan [14]. The fourth is meant to safeguard Alice, and was suggested by the first author and Riccobene [3] from the observation that late server replies could indicate illegal actions to have been performed. Our model does not force agents to act. They could even reply late, but Alice would discard late replies.

Note that temporal checks must involve timestamps. Therefore, saying that a session key has not expired means that the timestamp accompanying it inside the ticket has not expired. Similarly, saying that an authenticator has not expired means that the timestamp inside it has not expired. To enhance readability, we define the predicates

$$\begin{aligned}
&1.\ \mathsf{ExpirAuth}\ \ Tk\ \ evs && \equiv\ (\mathsf{Ct}\,evs) > Tk + \mathsf{AuthLife}\\
&2.\ \mathsf{ExpirServ}\ \ Tt\ \ evs && \equiv\ (\mathsf{Ct}\,evs) > Tt + \mathsf{ServLife}\\
&3.\ \mathsf{ExpirAutc}\ \ Ta\ \ evs && \equiv\ (\mathsf{Ct}\,evs) > Ta + \mathsf{AutcLife}\\
&4.\ \mathsf{RecentResp}\ \ T\ \ T' && \equiv\ T\ \leq\ T' + \mathsf{RespLife}
\end{aligned}$$

The inductive definition of Kerberos is presented in Appendix. Rules K1 to K6 formalise the protocol describing how to build new traces from each message. The other rules express respectively the spy's illegal activity (Fake), the accidental leak of expired authkeys (Oopsl), and of expired servkeys (Oops2). Traces are formed in reverse: the new event is put on the front. The temporal checks follow a pattern that is maintained throughout the protocol.

The current model allows the accidental leakage of session keys that have expired (see Oopsl and Oops2), because there is some risk increasing over time that the spy might get hold of some keys used in past runs. Our analysis

- discovers that non-expired session keys are strongly protected even from the illegal use of expired ones;
- discovers that Bob's guarantees of confidentiality over the servkeys are weakened by the risk of loss of expired authkeys;
- shows how to strengthen Bob's guarantees.

4.1 The Secrecy Goals

The technical results about Kerberos, such as *possibility* and *regularity* properties, are discussed elsewhere [1].

The secrecy goals are typically of two sorts: to limit the damage arising from a *key compromise*, and to keep keys *confidential*. The goals met by Kerberos are expressed by the following theorems.[2]

Key-Compromise Theorems typically state that some keys remain secret even when some session keys have been leaked to the spy. Their proofs can be hard, as they can often require case analysis under the analz operator. They also serve as crucial simplification lemmas for proving confidentiality theorems.

Most shared-key protocols prudently never use session keys to encrypt other session keys. Doing so would let the spy easily exploit the theft of one key to learn others.

Kerberos takes the risk of encrypting the servkey by the authkey in the message sent by Tgs. Therefore, the compromise of the authkey would obviously compromise also the servkey, as confirmed by the following theorem, proven straightforwardly from the definition of analz.

Theorem 1 (Compromise of servkey from compromise of authkey).

```
[| Says Tgs A (Crypt AuthKey {|Key ServKey, Agent B, Number Tt,
                 ServTicket|}) ∈ set evs;
   Key AuthKey ∈ analz (spies evs); evs ∈ kerberos |]
⟹ Key ServKey ∈ analz (spies evs)
```

Despite this weakness, three other important goals are expected to to be met.

1. Both authkeys and shared keys should remain secure from the compromise of any session keys, as session keys never encrypt them.

[2] Proofs are omitted here for the sake of brevity. They are available from the authors at http://www.cl.cam.ac.uk/~gb221/Kerberos/

2. If Tgs encrypts a servkey by an authkey, then the compromise of another authkey should not compromise that servkey, because a servkey should never be encrypted by more than one authkey.
3. All keys should remain secure from the compromise of a servkey, since servkeys never encrypt other keys.

Tackling the first goal required the definition of a set AuthKeys *evs* yielding the authkeys over a trace *evs*, and several lemmas to simplify it. The result was a long and time-consuming proof. The proofs of the remaining goals turned out to be a major task, so different strategies were investigated.

When Tgs sends the fourth message, a stable association between a servkey and an authkey is created, which resembles the association between nonces created by the Yahalom protocol [18]. A servkey is associated with only one authkey, but an authkey can be associated with several servkeys (i.e. an authkey can encrypt several servkeys), and these facts could be proven. Such association is formalised by the predicate KeyCryptKey *authkey servkey evs*, which holds if Tgs sends a fourth message containing *authkey* and *servkey* in the trace *evs*.

Lemma 2 (Protection from a set of compromised session keys).

```
evs ∈ kerberos ⟹
    ( SK ⊆ Compl (range shrK) ⟶
        (K ∈ SK. ¬ KeyCryptKey K SesKey evs) ⟶
            Key SesKey ∈ analz (Key``SK ∪ (spies evs)) )
            ⟺ (SesKey ∈ SK | Key SesKey ∈ analz (spies evs))
```

This lemma states that a set of compromised session keys SK that are never used by Tgs to encrypt the given session key does not help the spy to learn the session key. Precisely, the session key can be analysed from the traffic together with SK if and only if the session key belongs to SK or could be analysed from the traffic alone. The result is very general because the assumption on the predicate KeyCryptKey can be refined to define the type of session key — whether authkey or servkey — as shown below.

The proof consists of 40 Isabelle commands, necessary to apply several simplification lemmas — e.g. KeyCryptKey never holds on shared keys or on session keys not yet appeared on the traffic — structural lemmas — e.g. KeyCryptKey never associates an authkey with another authkey — and unicity lemmas — e.g. KeyCryptKey associates a servkey with one and only one authkey. This lemma is applied by the following three theorems that prove respectively the three expected goals stated above.

Theorem 3 (Protection from compromised session key).

```
[| AuthKey ∈ (AuthKeys evs) ∪ range shrK; SesKey ∉ range shrK;
   evs ∈ kerberos |]
⟹ Key AuthKey ∈ analz (insert (Key SesKey) (spies evs))
    ⟺ (AuthKey = SesKey | Key AuthKey ∈ analz (spies evs))
```

The theorem reads as follows: an authkey can be analysed from the traffic plus a session key if and only if the session key is the authkey or the authkey could be analysed from the traffic alone. This means that the spy can not exploit stolen session keys to learn new authkeys: the authkeys are safe from the compromise of any other session keys. The guarantee is strong and desirable — authkeys are valuable pieces of information, as they have a long lifetime — and also applies to shared keys.

The new proof simply applies the lemma stating that an authkey (or a shared key) is never treated by Tgs as a servkey

```
[| K ∈ AuthKeys evs ∪ range shrK; evs ∈ kerberos |]
⟹ ∀ K'. ¬ KeyCryptKey K' K evs
```

and then concludes by lemma 2: only 2 Isabelle commands.

Theorem 4 (Protection from compromised different authkey).

```
[| KeyCryptKey AuthKey ServKey evs;
   AuthKey ≠ AuthKey'; AuthKey' ∉ range shrK; evs ∈ kerberos |]
⟹ Key ServKey ∈ analz (insert (Key AuthKey') (spies evs))
    ⟺ (ServKey = AuthKey' | Key ServKey ∈ analz (spies evs))
```

If an authkey is associated with a servkey, then the compromise of a different authkey does not help the spy to learn the servkey. The form is the same as that of the previous theorem. The proof uses lemma 2 as well as a unicity lemma about the unique association of a servkey to an authkey:

```
[| KeyCryptKey AuthKey ServKey evs;
   AuthKey' ≠ AuthKey; evs ∈ kerberos |]
⟹ ¬ KeyCryptKey AuthKey' ServKey evs
```

Theorem 5 (Protection from compromised servkey).

```
[| ServKey ∉ (AuthKeys evs); ServKey ∉ (range shrK);
   evs ∈ kerberos |]
⟹ Key K ∈ analz (insert (Key ServKey) (spies evs))
    ⟺ (K = ServKey | Key K ∈ analz (spies evs))
```

The theorem states that no keys can be learned from the compromise of a servkey. The proof applies lemma 2 and a lemma stating that a servkey is never treated by Tgs as an authkey:

```
[| K ∉ AuthKeys evs; K ∉ range shrK; evs ∈ kerberos |]
⟹ ∀ K'. ¬ KeyCryptKey K K' evs
```

Confidentiality Theorems typically express the assumptions upon which each party can infer that a certain session key is secure from the spy.

They have a common feature. Any parties mentioned by the assumptions are required to be uncompromised in order to protect the secrets they know. With Kerberos, an agent has to trust that the agent at the other end of the communication is not conspiring with the spy. However, there are other real-world protocols for situations when nobody trusts anybody else.

Theorem 6 (Confidentiality for Kas).

```
[| Says Kas A (Crypt Ka {|Key AuthKey, Agent Tgs, Number Tk,
                 AuthTicket|}) ∈ set evs;
   ¬ ExpirAuth Tk evs; A ∉ bad; evs ∈ kerberos |]
⟹ Key AuthKey ∉ analz (spies evs)
```

The only session keys administered by Kas are authkeys. This theorem assures Kas that an authkey is safe as long as it has not expired. The proof exploits a unicity lemma stating that Kas never distributes the same authkey to different parties, and applies theorems 3 and 4.

Theorem 7 (Weak confidentiality for Tgs).

```
[| Says Tgs A (Crypt AuthKey {|Key ServKey, Agent B, Number Tt,
                 ServTicket|}) ∈ set evs;
   Key AuthKey ∉ analz (spies evs);
   ¬ ExpirServ Tt evs; A ∉ bad;  B ∉ bad; B ≠ Tgs; evs ∈ kerberos |]
⟹ Key ServKey ∉ analz (spies evs)
```

The theorem is targeted to the confidentiality of a servkey, as Tgs never sees any authkeys. In order to protect the servkey, the theorem assumes the authkey to be confidential (otherwise theorem 1 would apply). This is a weak guarantee: although Tgs can check the freshness of the timestamp, it can not check the confidentiality of the authkey. Theorems 3, 4, and 5 crucially help the proof.

Theorem 8 (Realistic confidentiality for Tgs).

```
[| Says Tgs A (Crypt AuthKey {|Key ServKey, Agent B, Number Tt,
                 ServTicket|}) ∈ set evs;
   Says Kas A (Crypt Ka {|Key AuthKey, Agent Tgs, Number Tk,
                 AuthTicket|}) ∈ set evs;
   ¬ ExpirAuth Tk evs; ¬ ExpirServ Tt evs;
   A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerberos |]
⟹ Key ServKey ∉ analz (spies evs)
```

This version is more realistic because Tgs and Kas are part of the same system, so they can inspect each other's activity: Tgs could check whether Kas has issued the authkey. The proof refines theorem 7 by theorem 6.

Theorem 9 (Confidentiality for Tgs).

```
[| Says Tgs A (Crypt AuthKey {|Key ServKey, Agent B, Number Tt,
                  ServTicket|}) ∈ set evs;
   Crypt (shrK Tgs) {|Agent A, Agent Tgs, Key AuthKey, Number Tk|}
     ∈ parts (spies evs);
   ¬ ExpirAuth Tk evs; ¬ ExpirServ Tt evs;
   A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerberos |]
⟹ Key ServKey ∉ analz (spies evs)
```

This is the most general version of confidentiality for Tgs because, if the involved parties can be trusted, Tgs gets the guarantee as soon as it receives the authticket $\{\mathsf{A}, \mathsf{Tgs}, \mathsf{AuthKey}, \mathsf{Tk}\}_{Ktgs}$ which is encrypted under its own shared key. The proof applies first a lemma stating that the authtickets originate with Kas, and then the previous theorem.

Note that the theorem holds since the authticket mentioned by the second assumption *appears in the traffic*, but becomes useful to Tgs only when Tgs gets hold of the authticket, perhaps later. The same remark applies to other theorems involving parts.

Theorem 10 (Confidentiality over authkey for Alice).

```
[| Crypt (shrK A) {|Key AuthKey, Agent Tgs, Number Tk, AuthTicket|}
     ∈ parts (spies evs);
    ¬ ExpirAuth Tk evs; A ∉ bad; evs ∈ kerberos |]
⟹ Key AuthKey ∉ analz (spies evs)
```

Alice gets a strong guarantee of confidentiality over the authkey from the reception of a non-expired message encrypted under her shared key. The proof is based on a lemma stating that, since Alice is uncompromised, the message encrypted under her shared key has originated with Kas. Theorem 6 is then applied.

Theorem 11 (Confidentiality over servkey for Alice).

```
[| Crypt (shrK A) {|Key AuthKey, Agent Tgs, Number Tk, AuthTicket|}
     ∈ parts (spies evs);
   Crypt AuthKey {|Key ServKey, Agent B, Number Tt, ServTicket|}
     ∈ parts (spies evs);
   ¬ ExpirAuth Tk evs; ¬ ExpirServ Tt evs;
   A ∉ bad;  B ∉ bad; B ≠ Tgs; evs ∈ kerberos |]
⟹ Key ServKey ∉ analz (spies evs)
```

The guarantee for Alice over the servkey is strong too, as she can check the reception of a message of the expected form encrypted under her shared key, then extract the authkey, and hence check again whether she gets an acceptable message encrypted under the authkey. The proof is based on the following sketch. Since Alice is uncompromised, the message mentioned by the first assumption originated with Kas. Theorem 6 then derives that the authkey is confidential, so the message mentioned by the second assumption originated with Tgs. Theorem 8 concludes.

Theorem 12 (Weak confidentiality for Bob).

```
[| Crypt (shrK B) {|Agent A, Agent B, Key ServKey, Number Tt|}
     ∈ parts (spies evs);
   Crypt AuthKey {|Key ServKey, Agent B, Number Tt, ServTicket |}
     ∈ parts (spies evs));
   Crypt (shrK A) {|Key AuthKey, Agent Tgs, Number Tk, AuthTicket|}
     ∈ parts (spies evs);
   ¬ ExpirAuth Tt evs; ¬ ExpirServ Tk evs;
   A ∉ bad;  B ∉ bad; B ≠ Tgs; evs ∈ kerberos |]
⟹ Key ServKey ∉ analz (spies evs)
```

The theorem says that the servkey is secure provided that it has not expired and that the authkey encrypting it has not expired either. This is a weak guarantee because Bob has no role in the AUTHENTICATION phase, never sees any authkeys, and can not check whether the authkey encrypting the servkey he gets has expired or not. However, the assumptions on the authkey are indispensable to the theorem because, should the authkey expire, it could be then leaked by accident and disclose (theorem 1) also the servkey to the spy.

Not only is the weakness highlighted by the previous theorem due to the incautious design of the fourth message, but also to the lack of a connection between the expiring times of the two kinds of session keys. The following scenario could happen: an authkey expires, its user is logged out from the workstation; the authkey is somehow leaked; the servkeys encrypted under that authkey are still non-expired (i.e. can be used) but compromised to the spy.

If Tgs only issued new servkeys when prompted with an authkey still valid for the whole lifetime of the servkeys, then the problem might be fixed. This intuition is confirmed by machine proofs. Rule K4 can be strengthened by the temporal check

$$(\mathsf{Ct}\ \ evs)+\ \mathsf{ServLife} \leq\ Tk+\ \mathsf{AuthLife}$$

In this stronger protocol, when Bob receives a non-expired servkey, he is also assured that the authkey encrypting it has not expired either. This lemma lets us remove from the previous theorem those assumptions about the authkey that could not be checked by Bob, so that his confidentiality guarantee is strengthened as follows.

Theorem 13 (Confidentiality for Bob — fixed model).

```
[| Crypt (shrK B) {|Agent A, Agent B, Key ServKey, Number Tt|}
     ∈ parts (spies evs);
   ¬ ExpirServ Tt evs;
   A ∉ bad;  B ∉ bad; B ≠ Tgs; evs ∈ kerberos |]
⟹ Key ServKey ∉ analz (spies evs)
```

5 Related Work

The approaches that have been tailored to the formal analysis of Kerberos Version IV are surprisingly not many. By contrast, a great number has been applied to the simpler BAN version.

The first author and Riccobene analyse Version IV [3] by *Gurevich's Abstract State Machine* [7]. They use a detailed algebraic model to formalise all possible actions of honest agents, but the eavesdropper's potentialities are finite. Theorems are stated from the viewpoint of the single agent of an infinite set. Proofs are carried out by hand thanks to the little formal overhead, but automated support is under development.

Mitchell et al. [15] model check a highly simplified version of Kerberos Version IV derived from Kohl et al. [10]. Timestamps are not included, and multiple runs are not allowed. They find no attacks on a system of size three — initiator, Kerberos servers and responder — and a "redirection" attack on a system of size four — with two responders — by which Alice might believe to be talking with Bob when in fact she has been redirected to Charlie, who is possibly compromised. They also check that the problem can be fixed according to the directions of RFC-1510 [9] by upgrading the authenticator. However, the official Kerberos Version IV [14] easily solves the problem by quoting Bob's name in message 4.

6 Conclusion

Kerberos Version IV has proven to be a remarkable case study.[3] Its analysis is far more complicated than the analysis of the BAN version [2].

The risks arising from the accidental loss of the different kinds of session keys have been formally tackled. Strong confidentiality assurances have been provided to each uncompromised party involved in the protocol, showing that non-expired session keys are not compromised by the accidental loss of expired ones. However, Kerberos requires the network clocks to be synchronised, which is a well known intrinsic weakness.

Each party has been provided with certain assumptions to check in order to infer valuable guarantees. If a party is not able to check some of these assumptions, then the involved guarantee is weak. This is how it is discovered that, in a realistically hostile environment, Bob gets a weak confidentiality guarantee unless Tgs makes a suitable temporal check, not stated by the Athena Technical Plan [14].

Modelling the possible compromise of session keys has greatly complicated the analysis. However, a deployed protocol such as Kerberos Version IV must be resilient against such losses. A litmus test for any protocol analysis method is whether it addresses such issues.

[3] The full Isabelle proof script executes in approximately 4 minutes on a 300 MHz Pentium Pro, the longest time amongst the protocols analysed thus far.

Acknowledgement. The research was funded by EPSRC, grants GR/K57381 'Mechanizing Temporal Reasoning' and GR/K77051 'Authentication Logics'.

References

1. G. Bella, L. C. Paulson. Using Isabelle to Prove Properties of the Kerberos Authentication System. Proc. of *DIMACS Workshop on Design and Formal Verification of Security Protocols*, Orman and Meadows (eds.), 1997.
2. G. Bella, L. C. Paulson. Mechanising BAN Kerberos by the Inductive Method. Proc. of *Conference on Computer Aided Verification*, Springer, LNCS Series, 1998.
3. G. Bella, E. Riccobene. Formal Analysis of the Kerberos Authentication System. *Journal of Universal Computer Science: Special Issue on Gurevich's Abstract State Machine*, Springer, 1997.
4. S. M. Bellovin, M. Meritt. Limitations of the Kerberos authentication system. *Computer Comm. Review*, 20(5) 119-132, 1990.
5. S. H. Brackin. A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols. Proc. of *Computer Security Foundations Workshop*, IEEE Press, 1996.
6. M. Burrows, M. Abadi, R. M. Needham. A logic of authentication. *Proceedings of the Royal Society of London,* 426:233-271, 1989.
7. Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In *Specification and Validation Methods*, Oxford University Press, E. Börger (ed.), 1995.
8. R. Kemmerer, C. Meadows, J. Millen. Three Systems for Cryptographic Protocol Analysis. *Journal of Cryptology*, 7(2), 79-130, 1994.
9. J. Kohl, B. Neuman. The Kerberos Network Authentication Service (Version V). Internet Request for Comment RFC-1510, 1993.
10. J. Kohl, B. Neuman, T. Ts'o. The Evolution of the Kerberos Authentication Service. IEEE Press, 78-94, 1994.
11. G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. *Tools and Algorithms for the Construction and Analysis of Systems,* Margaria and Steffen (eds.), LNCS1055, Springer Verlag, 147-166, 1996.
12. G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. Oxford University, Computing Laboratory, *Technical Report*, 1996.
13. C. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2), 113-131, 1996.
14. S. P. Miller, J. I. Neuman, J. I. Schiller, J. H. Saltzer. Kerberos authentication and authorisation system. *Project Athena Technical Plan,* Sec. E.2.1, 1-36, MIT, 1989.
15. J. C. Mitchell, M. Mitchell, U. Stern: Automated Analysis of Cryptographic Protocols Using Murphi. Proc. of *IEEE Symposium on Security and Privacy*, 141-151, 1997.
16. L. C. Paulson. *Isabelle: A Generic Theorem Prover.* Springer, 1994. LNCS 828.
17. L. C. Paulson. Proving properties of security protocols by induction. Proc. of *Computer Security Foundations Workshop*, IEEE Press, 1997.
18. L. C. Paulson. On Two Formal Analyses of the Yahalom Protocol. Cambridge University, Computer Laboratory, *Technical Report No. 432*, July 1997.
19. L. C. Paulson. Inductive Analysis of the Internet Protocol TLS. Cambridge University, Computer Laboratory, *Technical Report No. 440*, Dec. 1997.
20. S. Schneider. Verifying Authentication Protocols Using CSP. Proc. of *Computer Security Foundations Workshop*, IEEE Press, 1997.

Appendix. Kerberos Version IV Inductive Definition

```
kerberos :: event list set
inductive kerberos

Base  [] ∈ kerberos

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

K1   [| evs ∈ kerberos; A ≠ Kas |]
     ⟹ Says A Kas {|Agent A, Agent Tgs, Number (Ct evs)|}
           # evs ∈ kerberos


K2   [| evs ∈ kerberos; A ≠ Kas; Key AuthKey ∉ used evs;
        Says A' Kas {|Agent A, Agent B, Number Ta1|} ∈ set evs |]
     ⟹ Says Kas A (Crypt (shrK A)
                       {|Key AuthKey, Agent Tgs, Number (Ct evs),
                         (Crypt (shrK Tgs) {|Agent A, Agent Tgs,
                            Key AuthKey, Number (Ct evs)|}) |})
           # evs ∈ kerberos


K3   [| evs ∈ kerberos; A ≠ Tgs;
        Says A Kas {|Agent A, Agent Tgs, Number Ta1|} ∈ set evs;
        Says Kas' A (Crypt (shrK A) {|Key AuthKey, Agent Tgs,
                       Number Tk, AuthTicket|}) ∈ set evs;
        RecentResp Tk Ta1 |]
     ⟹ Says A Tgs {|AuthTicket,
                      (Crypt AuthKey {|Agent A, Number (Ct evs)|}),
                      Agent B|}
           # evs ∈ kerberos


K4   [| evs ∈ kerberos; A ≠ Tgs; B ≠ Tgs; Key ServKey ∉ used evs;
        Says A' Tgs {|(Crypt (shrK Tgs) {|Agent A, Agent Tgs,
                         Key AuthKey, Number Tk|}),
                      (Crypt AuthKey {|Agent A, Number Ta2|}),
                            Agent B|} ∈ set evs;
        ¬ ExpirAuth Tk evs; ¬ ExpirAutc Ta2 evs |]
     ⟹ Says Tgs A (Crypt AuthKey
                       {|Key ServKey, Agent B, Number (Ct evs),
                         (Crypt (shrK B) {|Agent A, Agent B,
                            Key ServKey, Number (Ct evs)|}) |})
           # evs ∈ kerberos
```

```
K5    [| evs ∈ kerberos; A ≠ B;
         Says A Tgs {|AuthTicket,
                      (Crypt AuthKey {|Agent A, Number Ta2|}),
                         Agent B|} ∈ set evs;
         Says Tgs' A (Crypt AuthKey {|Key ServKey, Agent B, Number Tt,
                        ServTicket|}) ∈ set evs;
         RecentResp Tt Ta2 |]
      ⟹ Says A B {|ServTicket,
                     (Crypt ServKey {|Agent A, Number (Ct evs5)|}) |}
           # evs ∈ kerberos


K6    [| evs ∈ kerberos; A ≠ B;
         Says A' B {|(Crypt (shrK B) {|Agent A, Agent B, Key ServKey,
                        Number Tt|}), (Crypt ServKey {|Agent A,
                                          Number Ta3|}) |} ∈ set evs;
         ¬ ExpirServ Tt evs; ¬ ExpirAutc Ta3 evs |]
      ⟹ Says B A Crypt ServKey (Number (Ta3 + 1))
           # evs ∈ kerberos


Oops1  [| evs ∈ kerberos; A ≠ Spy;
          Says Kas A (Crypt (shrK A) {|Key AuthKey, Agent Tgs,
                        Number Tk, AuthTicket|}) ∈ set evs;
          ExpirAuth Tk evs |]
       ⟹ Says A Spy {|Agent A, Agent Tgs, Number Tk, Key AuthKey|}
            # evs ∈ kerberos


Oops2  [| evs ∈ kerberos; A ≠ Spy;
          Says Tgs A (Crypt AuthKey {|Key ServKey, Agent B,
                        Number Tt, ServTicket|}) ∈ set evs;
          ExpirServ Tk evs |]
       ⟹ Says A Spy {|Agent A, Agent B, Number Tt, Key ServKey|}
            # evs ∈ kerberos
```