Using Isabelle to Prove Properties of the Kerberos Authentication System

Giampaolo Bella* Lawrence C Paulson†

Computer Laboratory – University of Cambridge
New Museums Site, Pembroke Street
Cambridge CB2 3QG (UK)

Abstract

The *Inductive method*, previously used to analyse classical, nonce-based cryptographic protocols, is here tailored to formalise Kerberos, a real-world, timestamp-based protocol. A complete formalisation of the whole protocol is achieved, and several guarantees about its entangled operation are proved using the theorem prover Isabelle.

1 Introduction

Since Needham and Schroeder pioneered in [6] that *protocol errors are unlikely to be detected in normal operations* and that *the need for techniques to verify the correctness of such protocols is great*, a number of methods have been developed to analyse cryptographic protocols. However, none of these methods can claim that a protocol is mathematically secure.

A combination of different methods might yield the best results. For instance, the use of a *belief logic* [3] during the design phase might help ensure freshness properties, and the use of a *state enumeration method* [4] might pinpoint simple flaws quickly. Deeper structural properties might be proved by way of the *inductive method*.

Using such a method, some famous cryptographic protocols of the literature have been analysed with the support of the generic theorem prover *Isabelle*. Several guarantees (nearly two hundred theorems in all) have been established about the following protocols: Needham-Schroeder (both shared-key and public-key), Otway-Rees, Yahalom [8, 9, 10]. A new attack has been discovered in a variant of the Otway-Rees protocol.

The inductive approach has also been tailored to analyse real-world protocols. The first one tackled in this field is Kerberos [5, 1], whose analysis has

*Giampaolo.Bella@cl.cam.ac.uk
†Larry.Paulson@cl.cam.ac.uk

required some effort for the mechanism of broadcast of a session key to the intended recipients which relies on the presence of two different trusted third parties. Kerberos is the first protocol relying on the use of timestamps (instead of nonces) to prevent the replay of messages. This has caused the introduction in the framework of a function yielding the current time and has brought the case for the definition of a new class of theorems stating temporal properties.

Assuming that the reader is familiar with Kerberos, Section 2 sketches the basic operation of the protocol. It also gives the motivations for the choice of Kerberos by going through the little related work. Section 3 then gives a feel about the inductive method, and Section 4 presents the actual analysis of the protocol. Finally, Section 5 sums up the paper. The complete Isabelle specification of Kerberos is given in Appendix.

2 Kerberos

Developed a decade ago as part of the project Athena at MIT [5], Kerberos has gone through many updates aimed at improving its security.

Kerberos relies on the use of timestamps to assure freshness of precious pieces of information as session keys. Kerberos checks twice the identity of Alice before she can get access to the network resource Bob, first by means of Kas (*Kerberos Authentication Server*) and then by means of Tgs (*Ticket Granting Server*). Alice receives by Kas a session key to be used for the communication with Tgs. Then, each time Alice wants to reach Bob, Tgs issues her with another session key which is to be used only for the communication between Alice and Bob.

Although the use of Kerberos as an authentication system for LANs is today widespread, the literature contains little work about the application of formal methods to it. This was the main motivation to our choice. Until a short time ago, the work of Burrows et al. [3] was the only significant attempt, but showed very few properties and has been widely criticised. A complete formalisation pointing out rigorously all the numerous details of the operation of Kerberos has been only recently achieved by means of the *Gurevich's Abstract State Machine* [1], but lacks automation. That work has been the basis to ours, which is, to our knowledge, the first attempt to mechanise such a complex protocol.

3 The Inductive Method

For the entire treatment of the method, we refer to [8]. Only some guidelines are given here.

3.1 Overview

The inductive method rests on the notion of *event*. Each time an agent A sends an agent B a message X, the event

$$\mathsf{Says}\ A\ B\ X$$

occurs in our model. This event suffices to describe all traffic over the network and is the only one envisaged so far.

Cryptographic protocols are formalised as the set of all possible traces, which are lists of events. Our specification defines this set inductively, i.e. it describes how to extend a trace of the set with a new event, according to the protocol operation.

Proving properties on the set formalising a protocol follows the classical induction principle. Once shown that the property holds on the empty set, we simply have to show that, if the property holds on a trace evs of the set, it still holds on the traces that extend evs by means of the rules for defining the set. Proofs would be too long to carry out on paper. This is where the theorem prover Isabelle comes into help.

The framework for shared-key protocols existing before the analysis of Kerberos relied on a *trusted party*, a special agent which has knowledge of all agents' secret keys. An eavesdropper, which attempts to get network resources by faking messages and exploiting accidental key losses, was impersonated by the agent Spy. Nonces were explicitly formalised as legitimate parts of messages. Tackling Kerberos has brought up the need for a few extensions (see Section 4).

3.2 Operators

The model makes use of three main operators that extend a given set of messages H. They are defined inductively, and enjoy several laws.

parts H is built from H by repeatedly adding the components of compound messages and the bodies of encrypted messages. Intuitively, it formalises all the knowledge that can be obtained from H, except for the keys that encrypt messages in H.

analz H is built from H by repeatedly adding the components of compound messages and the bodies of messages encrypted under keys already in analz H. It is a subset of parts H because it doesn't break ciphers.

synth H models the messages that the spy can built up from elements of H, including agent names, timestamps, compound messages, and messages encrypted with keys in H. Agent names are included because they are publicly known, timestamps because they can sometimes be guessed

Another operator models the set of messages an agent A receives from a trace evs:

$$\mathsf{sees}\ A\ evs.$$

It is inductively defined assuming that honest agents see only the messages intended for themselves, while the spy sees all traffic. Therefore, the set

$$\mathsf{synth}(\mathsf{analz}(\mathsf{sees}\ \mathsf{Spy}\ evs))$$

denotes all the fraudulent messages the spy can send by observing the trace *evs*. A spy able to send such a large number of messages has a *potentially infinite behaviour* [8] which goes beyond the limits of formalisations by state-enumeration methods.

3.3 Guarantees

Guarantees about the protocol operation are established in terms of theorems proved by Isabelle. When such proofs fail, they help to point out weaknesses of the protocol and even possible bugs. This is how a new bug in a variant of the Otway-Rees protocol has been discovered in [8].

The guarantees mentioned in [8] belong to five families.

Possibility properties are usually the first to be dealt with. The most important one consists in showing that there are traces that reach the end of the protocol. Although the model does not force agents to act, the lack of a trace proceeding from the first message to the last would indicate the protocol to have been transcribed incorrectly.

Forwarding lemmas are stated each time an agent forwards an item that it can not decrypt. They formally express that the spy will not learn anything new by seeing that item. Their proofs are trivial.

Regularity lemmas state that a certain item X does not belong to the set of messages available to the spy, which means that the spy can not ever get hold of X. They are usually easy to prove because stated in terms of the parts operator.

Authenticity theorems state that some valuable pieces of information as session keys and timestamps uniquely identify their message of origin. They provide guarantees about messages encrypted by keys believed to be secret (in general an encrypted message might have been faked if the key had been lost to the spy).

Secrecy theorems are the most difficult to prove, as they are stated in terms of the analz operator. An important one states that no session key is used to encrypt other session keys, so that the spy can not use a stolen session key to learn others. A crucial one states that if the trusted party distributes a session key to two agents which have not lost their secret keys, and if this session key is not lost to the spy, then no other agent can get hold of it.

4 The Analysis of Kerberos

Modelling Kerberos required the definition of two trusted parties Kas and Tgs, and of timestamps as parts of the messages. These main modifications were quickly applied to the existing framework based on one trusted party and on nonces, thus proving that it easily adapts to different protocol structures. Since Kas and Tgs are the foundation of the entire authentication procedure, our model assumes their private keys to be secure.

The Isabelle specification of Kerberos is presented in Appendix, where some mathematical symbols have taken the place of their ASCII equivalents to improve readability. The base of the induction is achieved by the `Nil` rule, while the power of the spy is formalised by the `Fake` rule. There are two `Oops` rules to model respectively the accidental loss of an *authentication key* (the session key used for the communication between Alice and Tgs) and of a *service key* (the session key used for the communication between Alice and Bob). They are currently omitted to simplify some proofs.

Each timestamp is obviously taken as the current time, which is formalised by the function `CT` mapping the current trace into an integer. Four lifetimes are defined as global constants, since sending the lifetimes inside the messages is a redundancy already addressed in [2].

`AuthLife` is the lifetime of the authentication key and usually lasts several hours; each authentication key can be used within this lifetime between Alice and Tgs.

`ServLife` is the lifetime of the service key and usually lasts a few minutes; each service key can be used within this lifetime between Alice and Bob. It is so short to prevent the re-use of a service key.

`RecentAuth` is the lifetime within which an *authenticator* is considered acceptable (Each time Alice sends a message to anybody, she builds an authenticator containing a timestamp and sends it inside the message).

`RecentResp` is the lifetime within which Alice considers acceptable a server's reply.

The need for the first two is straightforward. The reason for the last two is that, if an authenticator or a server's reply are "old" (i.e. they contain a timestamp older than `RecentAuth` or `RecentResp` respectively), then they are very likely to have been faked.

The timestamps `Ta`, `Ta1`, `Ta2` mark the moments that Alice sends a message to Kas, Tgs, and Bob respectively. Similarly, `Tk` marks the moment of the Kas's reply and `Tt` the moment of the Tgs's reply.

Each rule consists of some assumptions (which are enclosed between `[|` and `|]`) and a conclusion (which appears after the $\Longrightarrow$). If the assumptions hold, then the event mentioned in the conclusion *may* occur. Then, the gist of the protocol should arise easily.

4.1 Guarantees about Kerberos

Proving for Kerberos the possibility properties and the regularity lemmas already established for other protocols in [8] has not required much effort.

New forwarding lemmas have been established. One is proved for the *ticket*[1] that Alice receives from Kas (and then forwards to Tgs):

```
Says Kas' A (Crypt Key_A {|AuthKey, Agent Tgs, Tk, AuthTicket|})
  ∈ set_of_list evs
⟹ AuthTicket ∈ parts(sees Spy evs)
```

being `Key_A` Alice's secret key, and one is proved for the ticket that Alice gets from Tgs (and then forwards to Bob):

```
Says Tgs' A (Crypt AuthKey {|ServKey, Agent B, Tt, ServTicket|})
  ∈ set_of_list evs
⟹ ServTicket ∈ parts(sees Spy evs)
```

In the real world, Alice does not know who the true senders of the messages she gets are. This is why Kas and Tgs are primed.

The proof of such theorems is now down to one Isabelle command thanks to the development of suitable proof tactics.

A crucial authenticity theorem formally states that if a certain encrypted message appears, then it originated with Kas:

```
[|Crypt Key_A {|Key AuthKey, Agent Tgs, Tk, AuthTicket|}
     ∈ parts(sees Spy evs);
   A ∉ lost; evs ∈ kerberos|]
⟹ AuthTicket = Crypt Key_Tgs
                  {|Agent A, Agent Tgs, Key AuthKey, Tk|} &
    Says Kas A (Crypt Key_A {|Key AuthKey, Agent Tgs, Tk,
                    (Crypt Key_Tgs
                    {|Agent A, Agent Tgs, Key AuthKey, Tk|})
                |})
    ∈ set_of_list evs
```

Note the assumption that A has not lost her key to the spy. Similar is the counterpart for Tgs:

```
[|Crypt AuthKey {|Key ServKey, Agent B, Tt, ServTicket|}
     ∈ parts(sees Spy evs);
   Key AuthKey ∉ analz(sees Spy evs);
   B ∉ lost; evs ∈ kerberos|]
⟹ ServTicket = Crypt Key_B
                  {|Agent A, Agent B, Key ServKey, Tt|} &
    Says Tgs A (Crypt AuthKey {|Key ServKey, Agent B, Tt,
                    (Crypt Key_B
                    {|Agent A, Agent B, Key ServKey, Tt|})
                |})
    ∈ set_of_list evs
```

[1]For the reader unfamiliar with Kerberos, a ticket is an encrypted message that Alice uses as a credential with the next interlocutor, although ignoring its content. She receives the *authentication ticket* from Kas, and the *service ticket* from Tgs.

Also this second theorem relies on the encryption keys being secure. This is formalised by `B` $\notin$ `lost` for B's private key, and by `Key AuthKey` $\notin$ `analz(sees Spy evs)` for the session key `AuthKey`.

As for the secrecy theorems, not surprisingly the proof that no session key is used to encrypt other session keys failed. The reason is that the authentication key encrypts the service key; therefore, the theorem could be proved only for service keys. Proof methods recently applied to Yahalom [11] may yield results for Kerberos also.

Since the authentication key has a long lifetime, it is used to encrypt many service keys, so, from learning one authentication key, the spy could obtain a considerable number of other session keys. This addresses a protocol weakness already mentioned in [1].

The temporality of Kerberos suggested the definition of a new class of theorems, called *temporal properties*, expressing relations about the timestamps. They emphasise the key role played by the lifetimes, and can suggest the right values for them. This is of crucial importance since many replay attacks success because of too long lifetimes. For these reasons, more and more theorems of this class are being investigated.

For instance, the following is a theorem expressing that a service key is issued not later than the authentication key has expired:

```
[|Says Kas' A (Crypt Key_A {|AuthKey, Agent Tgs, Tk, AuthTicket|})
   ∈ set_of_list evs;
  Says Tgs' A (Crypt AuthKey {|ServKey, Agent B, Tt, ServTicket|})
   ∈ set_of_list evs|]
  ⟹ Tt ≤ Tk + AuthLife
```

5 Conclusion

A framework for the analysis of cryptographic protocols based on the inductive method has been developed within Isabelle and applied to some case studies [8].

This paper has discussed the analysis of Kerberos by means of that framework. Kerberos was chosen because it is a popular authentication system which was still lacking a mechanised analysis. Hence, while Kerberos has met its first mechanisation, the Isabelle framework has been, in turn, extended to deal with two trusted parties and with timestamps.

This analysis has shown that the method scales up to real-world protocols, and has founded a new class of guarantees, the *temporal properties*, only partially investigated so far, which will be the field for the future research.

Acknowledgement Giampaolo Bella wishes to thank the *Fondazione Bonino-Pulejo* (Messina-ITALY) and *The British Council* (Rome-ITALY) for funding his Ph.D. at the University of Cambridge (UK).

References

[1] 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.

[2] S. M. Bellovin, M. Meritt. Limitations of the Kerberos authentication system. *Computer Comm. Review*, 20(5) 119-132, 1990.

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

[4] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems,* Margaria and Steffen (eds.), volume 1055 of Lecture Notes in Computer Science, Springer Verlag, 147-166, 1996.

[5] 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.

[6] R. M. Needham, M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM,* 21(12), 993-999, 1978.

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

[8] L. C. Paulson. Proving properties of security protocols by induction. Cambridge University, Computer Laboratory, *Technical Report No. 409*, 1996.

[9] L. C. Paulson. Mechanized proofs of security protocols: Needham-Schroeder with public keys. Cambridge University, Computer Laboratory, *Technical Report No. 413*, 1997.

[10] L. C. Paulson. Mechanized proofs for a recursive authentication protocol. Cambridge University, Computer Laboratory, *Technical Report No. 418*, 1997.

[11] L. C. Paulson. On Two Formal Analyses of the Yahalom Protocol. Cambridge University, Computer Laboratory, *Technical Report No. 432*, 1997.

Appendix: Specifying Kerberos in Isabelle

```
kerberos :: event list set
inductive kerberos

Nil   [] ∈ kerberos

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

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

K2    [| evs ∈ kerberos; A≠Kas; Key AuthKey ∉ used evs;
         Says A' Kas {|Agent A, Agent Tgs, Timestamp Ta|}
          ∈ set_of_list evs |]
      ⟹ Says Kas A (Crypt (shrK A)
                   {|Key AuthKey, Agent Tgs, Timestamp (CT evs),
                     (Crypt (shrK Tgs)
                     {|Agent A, Agent Tgs, Key AuthKey,
                       Timestamp (CT evs)|})
                   |})
           # evs ∈ kerberos

K3    [| evs ∈ kerberos; A≠Tgs;
         Says A Kas {|Agent A, Agent Tgs, Timestamp Ta|}
          ∈ set_of_list evs;
         Says Kas' A (Crypt (shrK A) {|Key AuthKey, Agent Tgs,
                                       Timestamp Tk, AuthTicket|})
          ∈ set_of_list evs;
         Tk ≤ Ta + RecentResp |]
      ⟹ Says A Tgs {|AuthTicket, (Crypt AuthKey
                     {|Agent A, Timestamp (CT evs)|}),
                     Agent B |}
           # evs ∈ kerberos

K4    [| evs ∈ kerberos; A≠Tgs; Key ServKey ∉ used evs;
         Says A' Tgs {|(Crypt (shrK Tgs)
                     {|Agent A, Agent Tgs, Key AuthKey,
                       Timestamp Tk|}),
                     (Crypt AuthKey
                     {|Agent A, Timestamp Ta1)|}),
                     Agent B |}
          ∈ set_of_list evs;
         (CT evs) ≤ Tk + AuthLife;
```

```
        (CT evs) ≤ Ta1 + RecentAuth |]
     ⟹ Says Tgs A (Crypt AuthKey
                  {|Key ServKey, Agent B, Timestamp (CT evs),
                    (Crypt (shrK B)
                     {|Agent A, Agent B, Key ServKey,
                       Timestamp (CT evs)|})
                  |})
          # evs ∈ kerberos

K5   [| evs ∈ kerberos; A≠B;
        Says A Tgs {|AuthTicket,
                     (Crypt AuthKey {|Agent A, Timestamp Ta1|}),
                     Agent B|}
          ∈ set_of_list evs;
        Says Tgs' A (Crypt AuthKey {|Key ServKey, Agent B,
                                     Timestamp Tt, ServTicket|})
          ∈ set_of_list evs;
        Tt ≤ Ta1 + RecentResp |]
     ⟹ Says A B {|ServTicket, (Crypt ServKey
                   {|Agent A, Timestamp (CT evs)|}),
                 |}
          # evs ∈ kerberos

K6   [| evs ∈ kerberos; A≠B;
        Says A' B {|(Crypt (shrK B)
                    {|Agent A, Agent B, Key ServKey,
                      Timestamp Tt|}),
                    (Crypt ServKey
                    {|Agent A, Timestamp Ta2|}),
                  |}
          ∈ set_of_list evs;
        (CT evs) ≤ Tt + ServLife;
        (CT evs) ≤ Ta2 + RecentAuth |]
     ⟹ Says B A (Crypt ServKey (Timestamp Ta2))
          # evs ∈ kerberos

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

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