Pre-study of:

Security Protocol Verification using SPIN*

Audun Jøsang
Norwegian Institute of Technology
Trondheim

October 1995

Abstract

SPIN is a model checking tool. This paper presents some ideas on how SPIN/PROMELA can be adapted to cover security protocol verification. Existing methods are usually based on logical deduction and theorem proving, and I briefly describe the strength and weaknesses of BAN logic which is one of the methods most widely used. The goal of using SPIN for security protocol verification is to see whether model checking can be used to find protocol weaknesses which other methods leave undetected. A brief description of a practical realisation is given, and the difficulties, which are the specification and implementation of the model and the state space explosion, are pointed out.

1 Introduction

Security protocols are seemingly easy to design but surprisingly difficult to get right. Many different schemes for digital signature, authentication and other security services are published each year. Security protocols of this kind usually consist of 2-5 messages, which is quite a small number. A human is therefore easily led to believe, just by looking at the specification of the protocol, that it is correct for all its intended purposes. After all, this type of protocol is nothing more than a specification of some kind of program, and even average programmers are usually able to get a program of 3 lines correct the first time. Experience shows however, that flaws are discovered and still exist in accepted and well known security protocols.

This unfortunate situation has come about because we fail to be sufficiently conscious about a number of assumptions which are crucial for the protocol to work as intended, and because pure human intuition is unable to see and analyse all subtle possibilities for malicious manipulation. See e.g. [And94]. As a remedy for this, formal techniques have been applied with a certain degree of success, but some attacks still can not be discovered.

Existing techniques are based on logical deduction and proof, and in this context, a tool which can explore the complete state space of a security protocol would be a new approach to formal verification. It will be based on model checking rather than on theorem proving. If SPIN can be successfully be applied to security protocols, we will have an efficient tool which probably will help us spot other types of protocol attacks in addition to the attacks which already can be discovered by the existing methods.

*For a description of SPIN and PROMELA, see e.g. [Hol91]

2 What is a Security Protocol?

In the field of communication security, a number of security services are defined. The most common are listed below:

- **Authentication** can either be message origin authentication or entity authentication, meaning the corroboration that the source of data or an entity has the claimed identity.
- **Access Control** is the prevention of unauthorised use of a resource.
- **Confidentiality** is the property that information is not made available or disclosed to unauthorised individuals, entities or processes.
- **Integrity** is the property that the data have not been altered or destroyed in an unauthorised manner.
- **Non-repudiation** is the provision of proof of delivery or proof of origin of data. It is used to prevent one of the parties from falsely claiming not having received or originated a message.

A security protocol is typically designed to provide one or more of these services. The particular mechanisms which is implemented in order to provide a service can vary, and the most common mechanisms are *digital signature*, *encipherment*, *hashing* and *authentication exchange*. See e.g. [ISO88] for a detailed description of mechanisms which can be used to provide a particular security service.

As an example on how a security protocol provides security services, we shall study the CCITT X.509 protocol[ITU89] for signed secure communication between two parties, i.e. message exchange with authentication and confidentiality. This can e.g. be used to establish session keys. The protocol consists of only 3 messages.

Message 1 $\quad A \rightarrow B: \quad A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$

Message 2 $\quad B \rightarrow A: \quad B, \{T_b, N_b, A, N_a, X_b, \{Y_b\}_{K_a}\}_{K_b^{-1}}$

Message 3 $\quad A \rightarrow B: \quad A, \{N_b\}_{K_a^{-1}}$

Here, T_a and T_b are time stamps, N_a and N_b are nonces(a random number used once), and X_a, X_b, Y_a and Y_b are user data. Public key cryptography is assumed as mechanism for encipherment and digital signature. K_a and K_a^{-1} represent user A's public and private keys, and K_b and K_b^{-1} likewise for user B. Applying the public key K_b is equivalent with encipherment so that only user B is able to decipher(using a secret decipherment key), and applying the private key K_a^{-1} produces a digital signature of A which can be checked by anyone(using A's public signature checking key). If the RSA algorithm[RSA78] is used as mechanism of both encipherment and digital signature, then these operations are equivalent so that $\{\{X\}_K\}_{K^{-1}} = \{\{X\}_{K^{-1}}\}_K = X$ or in other words, the secret key used for signing is also used for decipherment, and the public key used for encipherment is also used for checking the digital signature. A mechanism with these properties is suggested by the notation of the protocol, but it is perfectly possible to use other mechanisms where digital signature and encipherment are different operations.

The digital signature $\{\}_{K^{-1}}$ is supposed to provide *integrity* and *authentication* of X_a,X_b,Y_a and Y_b, and the encipherment $\{\}_K$ is supposed to provide *confidentiality* of Y_a and Y_b.

Although the protocol can seem straightforward, we shall see that it contains several flaws which make it vulnerable to attacks.

3 Types of attack

3.1 Reuse of Confidential Data

This attack simply consists of stripping off the signature of any intercepted message and reuse the confidential data contained in it. Suppose e.g. that C is able to intercept message 1 of the original protocol from A to B so that B never receives it:

Message 1 $\quad A \rightarrow B$ (intercepted by C): $A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$

C can now send Y_a encrypted and signed as if C originated it:

Message 2 $\quad C \rightarrow B: \; C, \{T_c, N_c, B, X_c, \{Y_a\}_{K_b}\}_{K_c^{-1}}$

Notice that C is unable to read Y_a, but this attack still represents a serious weakness in the protocol. The error in the protocol consists of not following a simple thumb rule: *Never sign encrypted data.* The equivalent paper version of this scheme is intuitively easier to understand: *When sending a letter, sign the letter rather than the envelope.* Signing enciphered data is dangerous because the signer may not know what he is signing. This flaw can fixed by applying the digital signature before encipherment.

3.2 Replay Attack

In this attack an intruder C comes between user A and user B, and manages to make B believe that he is talking to A while in reality he is talking to C. The protocol thus fails in its main purpose, namely authentication.

The attack starts with C capturing an old message sent from A to B. This message is then replayed to B as follows:

Message 1 $\quad C \rightarrow B: \; A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$

B is not supposed to check that the nonce N_a has been used previously, its function is merely to link messages 1 and 2 in the original protocol. If the message is sufficiently recent, the reuse of T_a and the replay of the message will not be discovered by B.

B replies as though the message came from A, and provides a new nonce N_b:

Message 2 $\quad B \rightarrow C: \; B, \{T_b, N_b, A, N_a, X_b, \{Y_b\}_{K_a}\}_{K_b^{-1}}$

At this point, C causes A to initiate authentication with C, by whatever means:

Message 3 $\quad A \rightarrow C: \; A, \{T'_a, N'_a, C, X'_a, \{Y'_a\}_{K_c}\}_{K_a^{-1}}$

C replies to A, providing the nonce N_b which was provided by B in message 2.

Message 4 $\quad C \rightarrow A: \; C, \{T_c, N_b, A, N'_a, X_c, \{Y_c\}_{K_a}\}_{K_c^{-1}}$

A replies to C, signing the exact message needed for C to convince B that the first message was sent recently by A, and is not a replay of an old message.

Message 5 $\quad A \rightarrow C: \; A, \{N_b\}_{K_a^{-1}}$

Message 6 $\quad C \rightarrow B: \; A, \{N_b\}_{K_a^{-1}}$

The authors of the protocol seem to have hoped that the use of nonce N_b would be sufficient to link the third message to the first of the original protocol, since N_b links the last two messages, and N_a links the first two. The error here is that N_b alone does not link the last two messages, and this allows the intruder C to replay one of A's old messages, and thereby breaking the protocol. One possible solution is to include B's name in the last message.

4 Existing tools

The most widely recognised formal technique for security protocol verification is the BAN logic [BAN89]. Although there are a number of other techniques, BAN is the one which has captured the most scalps in this field, and also the one which first spotted the flaws in the X.509 authentication protocol described above. Various extensions to BAN have been proposed, see e.g. [GNY90],[AT91],[vO93],[SvO94] and [Sko94].

Recently, the authors behind BAN have produced a set of thumb rules [And94][AN95] in order to help researchers and designers produce good security protocols without necessarily having to go trough formal verification. By following these rules, an eventual formal verification is also made easier, and assurance may be added to the protocol design.

BAN logic is aimed specifically at formal analysis of *authentication protocols*. The purpose of the logic is to have formal expressions that reflect informal reasoning about authentication protocols. The idea is then to apply the logic to protocols in order to better explain them and understand their differences. There are two main components in this logic. One is the *language* used to describe the beliefs of principals taking part in a protocol. The other is a set of *inference rules* that describe how these beliefs change as result of communication and interpretation of messages. The language contains several constructs that express central concepts in authentication. This includes *good keys* used to determine the originator of a message, *authorities* trusted to generate a correct message contents, and *freshness* of a message, meaning that it is not a replay from a previous run of the protocol.

From a practical viewpoint, the analysis of a protocol is performed as follows:

- The protocol is rewritten or *idealised* using the BAN formalism.
- Assumptions about the *initial state* are written.
- Logical formulas are attached to the statements of the protocol, in the form of *assertions* about the state of the system after each message.
- The logical postulates are applied to the assumptions and assertions, in order to *deduce* the beliefs that the principals should have as a result of running the protocol.

BAN has gained a wide reputation, but it must not be understood as to be perfect. One possible weakness of BAN is the lack of any expression for confidentiality as a property of encryption. This becomes critical when a protocol step depends on the confidentiality of a previous message in order to work as intended. As a result, certain *interpretations* of a protocol specified in BAN formalism are vulnerable to special attacks. These issues have been the cause of some controversy [Nes90][Sne92].

Another point is the failure of BAN logic to address special forms of interleaving attacks involving replay of messages from at least two contemporaneous protocol runs. A such attack is presented in in [Syv93].

5 Purpose of Security Protocol Verification with SPIN

The purpose of applying SPIN[Hol91] to security protocols is to verify their resistance to malicious manipulation. This feature is usually not needed in normal communication protocols, where entities have no will or intent to do neither good nor bad, they just execute the protocol which has been programmed.

From a pure communication point of view, security protocols are usually very simple. The danger of deadlocks, livelocks and non progress cycles can immediately be excluded just by looking at the protocols.

On the other hand we must now consider protocol entities with imagination and will to do all sorts of things which could give them some potential profit or advantage, and this calls for a complete new frame of reasoning.

In principle, a malicious protocol entity could easily cause a deadlock if it wished to do so, by simply not answering any request or repeatedly sending completely erroneous messages, but this is not what we are interested in, because it will quickly be discovered by the other untampered entities. Our goal is to spot manipulations that untampered entities are unable to detect, i.e. false message sequences or messages with false contents which are accepted as genuine.

One result of running a security protocol is the establishment of beliefs at the various entities taking part in the protocol. If the protocol design is very bad, the intended beliefs are not achieved even when the protocol has not been manipulated. It is worth mentioning that this kind of protocol errors can probably not be found with SPIN. BAN logic is particularly good at this, i.e. it can prove that the intended beliefs are established if the protocol is executed *as specified*. The strong side of SPIN on the other hand will be to discover protocol *manipulations*, and this is exactly an area where BAN logic can fail.

6 Requirements and Practical Realisation

Protocol verification with SPIN starts with implementing the protocol entities in PROMELA formalism, including the correctness criteria. This step will be similar for security protocols.

In addition, a malicious entity must be defined and implemented in PROMELA formalism. The main requirement for such an entity is that it shall be able to execute "manipulated" protocols within a certain freedom of choice. To define and implement such a protocol entity is one of the main problems to be solved.

Assuming that a malicious entity can be implemented, it can play two roles. Firstly, we can let it replace an untampered protocol entity and run the protocol with the remaining entities. Secondly, we can let it play the role of an intruder, i.e. come between the other entities in various ways.

To follow the example of the X.509 protocol, entities A and B must be implemented to execute sessions of the specified protocol in different directions in an interleaved fashion. C must be capable of performing manipulations, i.e. a set of variations of the specified protocol, based on message interception or eavesdropping and on inherent intelligence. Figure 1 illustrates the model.

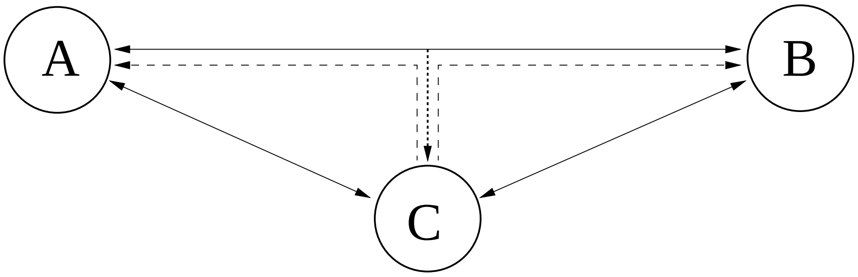

Figure 1: Model used for protocol analysis

Key to the figure:	————————	normal protocol
	– – – – – –	manipulated protocol
	....	eavesdropping/interception

We must establish a valid state transition sequence for each untampered protocol entity. During verification, SPIN must search for alternative protocol sessions which are indistinguishable from a normal protocol session, i.e. leading from a defined start state to an end state recognised as genuine even if it is not. All message sequences with this property must be stored for subsequent inspection either manually or by some other automated tool, to see if they really constitute an attack.

To resume, security protocol verification with SPIN consists of the following steps:

- Specify the model:
  - Define the normal protocol sessions to be executed.
  - Define the manipulated protocol sessions to be executed.
  - Define the choice of selecting messages for the manipulated protocol sessions.
  - Define the freedom of choice of selecting messages and contents of the messages sent by the malicious entity.
- Implement the untampered entities in PROMELA.
- Define a valid state sequence and end state for each entity, and let it be the trigger to mark a suspect protocol.
- Implement the malicious entity according to the model.
- Run the simulation/verification.
- Inspect suspect message sequences.

The intention is that this method shall be able to discover the replay attack of section 3.2. In order to cover the attack described in section 3.1 where confidential data is reused, we would in addition need a mechanism which can describe that confidential data may not be reused.

7 Estimating the state space

Next to the difficulties related to the specification and implementation of the FSM in PROMELA, the state space size can be a problem. I will here try to specify an FSM, and estimate the size of the resulting state space.

We start with assuming that the entities A, B and C run a total number p of protocol sessions with each other. In the case of X.509, each session consists of 3 messages, and without considering the message order we get $(p \cdot 3)!$ possibilities, but observing that for each session, only 1 out of 6 possible sequence orders is allowed, we can divide this number by 6^p. The number of possible interleaved message sequences q is then:

$$q = \frac{(p \cdot 3)!}{6^p}$$

Each time message 1 or 2 is sent from C in a normal protocol, C has the choice of submitting an observed or a completely new nonce. If f messages are affected and the number of choices is given by c, the state space size s can be written as:

$$s = qc^f = \frac{(p \cdot 3)! c^f}{6^p}$$

As a first example, I will specify a model which at least shall be able to discover the replay attack of the X.509 protocol. Because we know what we are looking for, we can say that A shall run only one session with B and one session with C, as indicated on figure 2. In addition C shall run a third manipulated protocol session with B based on *observed* (captured and received) messages from the two other sessions.

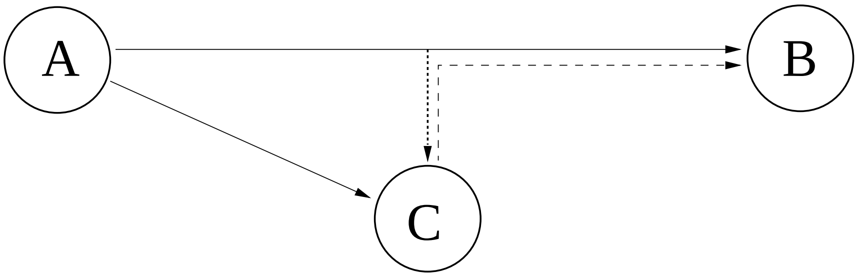

Figure 2: Model for analysis of the example

Key to the figure:		
	————————	normal protocol
	– – – – – –	manipulated protocol
	....	eavesdropping/interception

The number of interleaved sequences is thus obtained with $p = 3$. Now, only message 2 from C to A is affected by the choice of nonces, thus we get $f = 1$. Not all observed nonces are relevant, and assuming some intelligence in the implementation of C we can assume the number of choices to be $c = 3$. This should be enough to give us a rough estimate of the state space s:

$$s = \frac{(3 \cdot 3)!3^1}{6^3} = 5.040$$

Running SPIN on an average workstation gives a state space limit of about 500.000, so this result looks promising, and the model will be adequate to discover the replay attack described in section 3.2, but we must not forget that we in this example knew exactly which sessions were involved in the attack, so that in fact we were cheating. Let's look briefly at how the state space size increases when extending the model. If each entity runs one session with the two other entities, and C in addition runs a manipulated protocol session with both A and B, we get the model of figure 1 with $p = 8$. By investigating the model, we can observe that 4 messages from C will be affected by the choice of nonces, so we get $f = 4$, and c can be estimated to 4. Our state space then explodes and becomes:

$$s = \frac{(8 \cdot 3)!4^4}{6^8} \approx 10^{19}$$

The specifications of this model **is** reasonable, because many attacks are complicated and will involve messages from different interleaved protocol sessions. Knowing that SPIN "only" can handle models with up to 10^7 states, we clearly see that the limitations of a tool like SPIN quickly is reached, and that verification of security protocols often will involve simplifications which could affect the effectiveness of the method.

8 Conclusion

With the existing formal methods for security protocol verification, most types of design flaws can be discovered, albeit not all. SPIN and PROMELA together form a very efficient tool for protocol and FSM verification, and this paper has presented some ideas on how this method can be extended to cover security protocols. The particular area where SPIN can be useful, is investigation of protocol manipulations. The purpose is not to have a method which can do everything, but one that hopefully can spot new types of errors which other methods are unable to discover.

An adaptation of SPIN for this purpose has been roughly described, and the main obstacles which have been pointed out are the implementation of "malicious" protocol entities, and the state space explosion. As a consequence, limitations imposed on the complexity of the models, and thus also on the protocols, may reduce the value of the proposed method.

As a next step, it would be interesting to make a trial realisation. I am afraid that the study presented in this paper is too superficial, and that a more detailed specification of the model is needed before any practical trial can be conducted.

References

[AN95] Ross Anderson and Roger Needham. Robustness principles for public key protocols. In *Advances in Cryptology - CRYPTO'95*, pages 237–247. Springer-Verlag, August 1995.

[And94] Ross J. Anderson. Why cryptosystems fail. *Communication of the ACM*, 37(11):32–40, November 1994.

[AT91] Martín Abadi and Mark Tuttle. A semantics for a logic of authentication. In *Proceedings of the Tenth ACM Symposium on Principals of Distributed Computing*, pages 201–216, August 1991.

[BAN89] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. Technical report, DEC Systems Research Center, February 1989. Research Report 39.

[GNY90] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings og the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 234–248, Los Alamitos, California, 1990. IEEE Computer Society Press.

[Hol91] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[ISO88] ISO. *IS7498-2. Basic Reference Model For Open Systems Interconection - Part 2: Security Architecture*. International Organisation for Standardisation, 1988.

[ITU89] ITU. *X.509, The Directory - Authentication Framework*. CCITT, 1989.

[Nes90] D. M. Nesset. A critique of the burrows, abadi and needham logic. *Operating Systems Review*, 24(2), April 1990.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signature and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Sko94] Åsmund Skomedal. *Formal Analysis of Mechanisms for Access Control in Distributed Computing Systems*. PhD thesis, Norwegian Institute of Technology, 1994.

[Sne92] Einar Snekkenes. Roles in cryptographic protocols. In *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, Canada*. IEEE Computer Society Press, 1992.

[SvO94] Paul F. Syverson and Paul C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 14–28. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[Syv93] Paul F. Syverson. On key distribution protocols for repeated authentication. *Operating Systems Review*, 27(4), October 1993.

[vO93] Paul C. van Oorschot. Extending cryptographic logics of belief to key agreement protocols(extended abstract). In *Proceedings of the first ACM Symposium on Computer & Communication Security*, pages 232–243, November 1993.