Issues in Generating Security Protocols Automatically

Along Lin
Trusted E-Services Laboratory
HP Laboratories Bristol
HPL-2001-240(R.1)
October 19, 2006*

security protocols, protocol specification, security property

Protocol design is a very challenging and knowledge-intensive task. Designing a protocol is as hard as synthesizing a program. In this paper, we will first discuss the issues in our automatic security protocol generator, and then propose an approach to constructing security protocols intelligently, based on a security property specification and an extensible general-purpose library of schemas. Each schema is defined by specifying intended behaviors and is described as a sequence of primitive messages. Finally, some conclusions are drawn.

Issues in Generating Security Protocols Automatically

Along Lin
Hewlett-Packard Laboratories
Filton Road, Stoke Gifford, Bristol BS34 8QZ, U.K.

Abstract

Protocol design is a very challenging and knowledge-intensive task. Designing a protocol is as hard as synthesizing a program. In this paper, we will first discuss the issues in our automatic security protocol generator, and then propose an approach to constructing security protocols intelligently, based on a security property specification and an extensible general-purpose library of schemas. Each schema is defined by specifying intended behaviors and is described as a sequence of primitive messages. Finally, some conclusions are drawn.

Keyword(s): security protocols; protocol specification; security property

1. Introduction

Security protocols are supposed to use cryptography to set up private communication channels on an insecure network. However, many protocols contain flaws, and because security goals are seldom explicitly specified in detail, we cannot be certain what constitutes a flaw. Actually, a widely accepted standard threat model for cryptographic protocols has been proposed [Dolev and Yao 81]. It has the following assumptions about the capabilities of an attacker:

- He can obtain any messages passing through the network;
- He is a legitimate user of the network, and thus in particular can initiate a conversation with any other user;
- He will have the opportunity to be a receiver to any principal;
- He can send messages to any principal by impersonating any other principal.

Thanks to a number of researchers' significant contributions, security protocols can now be analyzed formally [Paulson 99]. After many security protocols have been found flawed, it is realized that it is exceptionally very hard to get security protocols right in the first place. A well-known public-key based authentication protocol, which was first published by Needham and Schroeder [Needham and Schroeder 78], turned out to be flawed 17 years later [Lowe 95].

Much research has been done on protocol verification. Some researchers have proposed a direct and fully automated translation from standard security protocol properties descriptions to rewrite rules [Jacquemard et al 2000], in which the existence of flaws in the protocol can be revealed by the derivation of an inconsistency. Recently, strand spaces based approach seems very promising by its simplicity of the model and the ease of producing intelligible and reliable proofs of protocol correctness even without automated support [Fabrega 98]. The extensions to the strand space model have also been suggested by researchers [Song 2001], who use a new efficient representation and utilize techniques from both model checking and theorem proving approaches. Following their work on efficient automatic checking algorithm for security protocol analysis [Song 99], we try to investigate some new method of generating security protocols practically and intelligently, based on a protocol's specification and a set of seed protocols.

In this paper, we attempt to draw some general lessons on the generation of security protocols by using an approach based on protocol-specific knowledge base and specification, and suggest a schema-based method by using planning techniques.

2. Protocol Representation

In the following, the representation and the types for the keys, nonces, and messages of a cryptographic protocol will be figured out and well refined. It is believed that any well-typed protocol is robustly safe. For convenience, most concepts involved will be defined as structures in PROLOG.

2.1 Principal

A principal can be an entity, an agent, or a user, which can be a computer, a device, a resource, or a service provider. It is represented by a structure *identity(Name)*, where *Name* is its symbolic name. When discussing about a security protocol, we usually need to assume a set of principals involved. Typical principal names are 'A', 'B', and 'S' that normally denotes a trusted server.

2.2 Nonce

A nonce is either a randomly generated number or an expression. For convenience, it is represented by a structure *nonce(Identity, Value, Name, Number, Known)*, where

Identity reflects whom the nonce belongs to.
Value is the real value of the random number.
Name is the nonce's symbolic name.
Number is the serial number of the nonce generated by the principal *Identity*.
Known is a list of identities' symbolic names to which the nonce is known.

Thus, the first nonce generated by principal A can be expressed as follows:
nonce('A', *V*, 'Na', 1, ['A']), where *V* denotes the value of its corresponding random number.

2.3 Data

Data is required in some protocols. It is expressed as *data(Generator, Value, Name, Number, Known)*, where *Generator* is the symbolic name of its generating principal, *Value* is its real value, *Name* is the symbolic name of the data item, *Number* and *Known* are the date's serial number and those principals to whom the data is known initially, respectively.

2.4 Keys

There are three types of keys involved in a security protocol, which are public keys, private keys, and symmetric keys. Public and private keys occur in a public key based cryptographic system whereas symmetric keys are used in a secret key based cryptographic system. Public keys can be made available to every one, but private keys must be only accessible to their owners. A symmetric key is supposed to be shared between two parties. In the following descriptions, *Owner* is the key owner whereas *Name* and *Number* are the symbolic name and serial number of a key, respectively.

- *publicKey*(*Owner*, *Name*, *Number*, *Known*), where *Known* is a list of identity to whom the public key is assumed to be accessible at the beginning.

- *privateKey*(*Owner*, *Name*, *Number*, *Known*), where *Known* is a list of identity to whom the private key is available at the beginning. In most situations, *Known* contains only *Owner*.

- *symmetricKey*([*First*, *Second*], *Name*, *Number*, *Known*), where *First* and *Second* are the names of the principals sharing the symmetric key, and *Known* is a list of identity to which the symmetric key is available initially.

Alternatively, all these types of keys can be expressed as a uniformed structure: *key*(*Type*, *Owners*, *Name*, *Number*, *Known*), where *Type* is *public*, *private*, *and symmetric* and *Owners* is a list of principal symbolic names.

2.5 Simple Message

A simple message is defined as one of the following structures:

- *data(Generator, Value*, *Name*, *Number*, *Known*)
- *key*(*Type*, *Owners*, *Name*, *Number*, *Known*)
- *nonce(Identity*, *Value*, *Name*, *Number*, *Known*)
- *identity*(*Name*)
- others such as message digest and time-stamps

2.6 Message

A message can now be defined as one of the followings:

- A simple message
- An encrypted message *encrypt*(M, K), where M is a message and K is a key.
- A concatenated message [M_1, … , M_n], where M_i (i = 1, …, n) is a message.

2.7 Event

An event is a message exchange between two participating principals or entities, which is normally the sending of a message from one entity to another. It can be expressed by a structure *event*(*From*, *Message*, *To*), where *From* and *To* are the sending and receiving principal's symbolic names, respectively. *Message* contains the information to be exchanged between two entities specified by *From* and *To*.

2.8 Protocol

Finally, by a protocol we mean a list of events, which can be defined as *protocol*([e_1, … ,e_n]) where e_i is an event in a system. Particularly, as an illustration, we take as an example of a protocol-specific constraint on the relationship between two neighboring events e_i, e_j ($j = i + 1$) such that one of the following holds:

- They have the same message senders.
- They have the same message recipients.
- The message recipient of e_i is the message sender of e_j.
- The message sender of e_i is the message recipient of e_j.

Another protocol representation is based on Strand spaces [Fabrega et al 98] and can be described as below. An event can be represented by *event*(*Operator*, *M*, *S*). When *Operator* is '+', it means an event sending a message *M* to strand *S*. Otherwise, *Operator* is '-' and it is an event receiving a message *M* from strand *S*. A strand is a sequence of events and represents either the execution of an action by a legitimate party in a security protocol or else a sequence of actions by a penetrator. It can be expressed by a structure *strand*(*Name*, *Events*), where *Name* is its name and *Events* is a sequence of events, represented by <e_1, … ,e_n> where e_i is an event as defined in section 2.7. Finally, a strand space is a collection of strands, equipped with a graph structure generated by causal interaction. It can be represented by the structure *bundle*(*Strands*) where *Strands* is a list of strands [s_1, … , s_m]. In this framework, protocol correctness claims may be expressed in terms of the connections between strands of different kinds.

From the protocol generation perspective, these two protocol representation methods are equivalent, which has already been demonstrated in our prototype. Therefore, in the following discussion, we will not distinguish them.

2.9 A Protocol Representation Example

Needham-Schroeder shared key authentication protocol (shown in Fig. 1) is described below using our representation syntax. 'A' and 'B' are two participating principals; 'S' is the trusted third party. For clarity purpose in the following representation, we introduce several simple notations first.

A = *identity*('A'),
B = *identity*('B'),
S = *identity*('S'),
K_{AB} = *key*(*symmetric*, [A, B], 'K_{AB}', 1, ['A', 'B']),
K_{AS} = *key*(*symmetric*, [A, S], 'K_{AS}', 1, ['A', 'S']),
K_{BS} = *key*(*symmetric*, [B, S], 'K_{BS}', 1, ['B', 'S']),
N_A = *nonce*('A', Na, 'N_A', 1, ['A']),
N_B = *nonce*('B', Nb, 'N_B', 1, ['B']),
$N_B - 1$ becomes *nonce*('B', Nb - 1, 'N_B', 1, ['B']).

E1 = *event*('A' [A, B, N_A], 'S'),
E2 = *event*('S', *encrypt*([N_A, K_{AB}, B, *encrypt*([K_{AB}, A], K_{BS})], K_{AS}), 'A'),
E3 = *event*('A', [S, *encrypt*([K_{AB}, A], K_{BS})], 'B'),
E4 = *event*('B', *encrypt*(N_B, K_{AB}), 'A'),
E5 = *event*('A', *encrypt*($N_B - 1$, K_{AB}), 'B'),

Finally, the Needham-Schroeder protocol is represented by *protocol*([E1, E2, E3, E4, E5]).

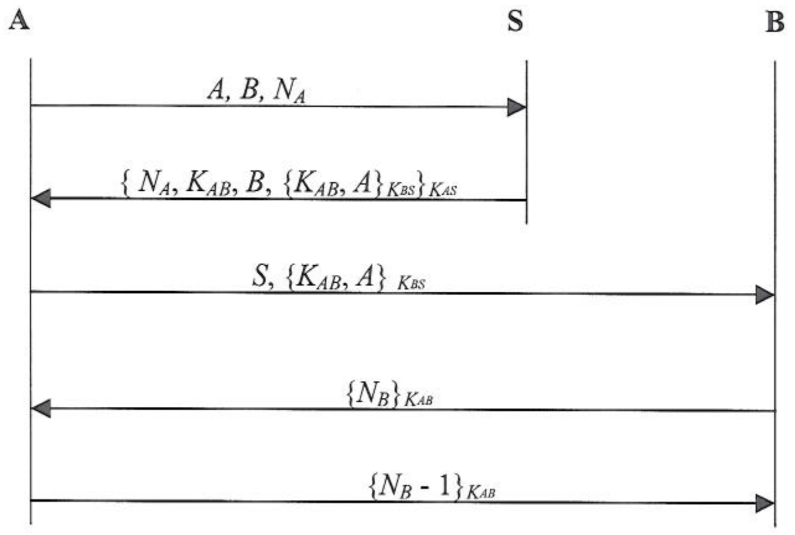
A
S
B
A, B, N_A
$\{N_A, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
$S, \{K_{AB}, A\}_{K_{BS}}$
$\{N_B\}_{K_{AB}}$
$\{N_B - 1\}_{K_{AB}}$

Figure 1. Needham-Schroeder Shared Key Protocol

3. Issues in the Generation of Messages Automatically

From our experiments on automated protocol generation, it is discovered that naïve message generation is inappropriate for an efficient generation of a security protocol. This is due to the fact that most generated messages using the syntax described in section 2.6 will not make sense when they appear in a protocol. Therefore, we adopt a constraints-based message generation.

The approach to systematically generating messages in a protocol is guided by the following constraints:

- The cost associated with a message *M*, which will be defined in section 3.1.
- The context of message generation provides some knowledge about what kind of messages will make sense. It includes all information available to the sending principal so far. Part of the context comes from the messages received from other principals earlier.
- The general knowledge about security protocol design that can be integrated in the protocol generator.
- A requirements description specific to the current security protocol generation.

3.1 The Cost of Generating a Message

The cost function $f(M)$ for message M is defined as follows:

$$f(M) = \begin{cases} C, \text{ when } M \text{ is a simple message. It is defined by } cost(T, C) \text{ as an input to the protocol generator, where } T \text{ is the message type.} \\ f(S) + C, \text{ if } M \text{ is } encrypt(S, K) \text{ and } C \text{ is the cost of encrypting a message using a key. } C \text{ is defined by } encrypting\_cost(T, C), \text{ where } T \text{ is the type of } K. \\ sum(f(M_i)), \text{ if } M \text{ is } [M_1, \dots, M_n], \text{ where } M_i \text{ is a message.} \end{cases}$$

3.2 The Depth and Width of a Message

Given a message M, its depth and width expressed by $d(M)$ and $w(M)$ respectively can be computed as follows.:

$$d(M) = \begin{cases} 1, \text{ if } M \text{ is a simple message} \\ d(S), \text{ if } M \text{ is } encrypt(S, K) \\ 1 + max\{d(M_1), \dots, d(M_n)\}, \text{ if } M \text{ is } [M_1, \dots, M_n]. \end{cases}$$

$$w(M) = \begin{cases} 1, \text{ if } M \text{ is a simple message} \\ w(S), \text{ if } M \text{ is } encrypt(S, K) \\ max\{n, w(M_1), \dots, w(M_n)\}, \text{ if } M \text{ is } [M_1, \dots, M_n] \end{cases}$$

3.3 The Requirements for a Specific Security Protocol

The requirements for a security protocol can be specified by a set of predicates. They will be used as an input to the generic security protocol generator. The specification describes the following details about the protocol to be generated:

- How many parties are involved and who are those participants
- How many nonces will be involved and what they are
- A description of keys available and which keys can be used to encrypt messages
- Whether a key can be exchanged between participants
- An indication of which cryptographic system is adopted and which keys can be used to encrypt messages
- Whether all messages in the generated protocol must be encrypted
- Whether embedded encryptions are needed
- With respect to nonces and keys, what each message in the generated protocol must include, both or either of them, or nothing.

3.4 How to Generate Messages

According to the message definition given in section 2.5, even with a very limited number of simple messages, a very large number of messages can still be generated. However, if the cost, depth, and width of a message to be generated and other protocol-specific constraints are enforced during the generation of a security protocol, the message space to explore can be dramatically reduced by trimming some branches as early as possible.

The generation of messages can be done by a systematic search method. Heuristic methods are normally much more efficient than brute-force methods by using domain-specific knowledge. However, if completeness is required in the protocol generator, they cannot be adopted; otherwise, some useful messages may not be generated.

With respect to the exploration of a message space, any of the following methods can be utilized:

- Depth-first
- Width-first
- Cost function-guided
- Hybrid method by mixing the cost-guided approach and limited depth-first

Because our generation of a message M is controlled by limiting the values of $f(M)$, $d(M)$, and $w(M)$ under the guidance of a protocol requirements specification, we adopt a hybrid approach to message space exploration.

It is realized that messages had better not to be generated at run-time. There are two reasons. One is that the run-time generation of messages is always inefficient, and the other is due to the fact that generating messages systematically will make the randomized protocol much difficult. On the contrary, if we can generate messages beforehand, it will be easier to randomly select a set of generated messages.

4. Generating Security Protocols

For the simplicity of discussing about protocol generation and analysis, we have the following assumptions:

- An attacker cannot guess a random number that is freshly chosen from a sufficiently large space;
- Without the correct secret or private key, an attacker can neither retrieve plaintext from given ciphertext, nor can they create valid ciphertext from given plaintext by using a perfect encryption algorithm;
- An attacker cannot find the private component, i.e., the private key corresponding to a given public key.

As discussed in section 2, a protocol can be simply defined as a list of events. From the perspective of a protocol's generation process, it is both helpful and understandable to decompose it into several steps. The first gives the protocol's initial state, describing who the participants are, which keys are available to each principal, and so on. It could also include its structure properties such as the number of participating principals, the number of message exchanges, the senders and recipients of each message exchange, and the order of message exchanges. The second one constructs a set of messages, which are generated under the control of a set of constraints and guided by the knowledge about a particular protocol design, both general and specific to it. Finally, the generated protocol is checked for correctness and attack-freeness.

The structure property of a security protocol can be either described explicitly in a knowledge base or constructed automatically, depending on how much automation we want to achieve during a protocol generation. Because many different sequences of messages can be constructed from a protocol's given initial state, for an efficient semi-automated protocol design, it is reasonable to specify the protocol's structure properties in a knowledge base and focus on the contents of messages exchanged in the protocol. The knowledge base will also contain all other information specific to a particular security protocol. However, if an automated protocol design is needed, a protocol's structure property can be produced as well using non-deterministic backtracking.

Alternatively, we can design a new security protocol by modifying an existing security protocol, which will be used as a seed protocol and modified in such a way that the resulted protocol is still rational and makes sense. To produce a protocol more randomly, there need several steps. First, we build a set of well-established protocols of different categories and functionalities in advance. Then, given the requirements for a specific security protocol, we decide which category of seed protocols is used. After randomly selecting one seed protocol in the chosen category, we perform a set of operations on it.

The permitted operations on a seed security protocol include but are not limited to the following:

- Inserting a list of randomized messages into the given protocol;
- Modifying one or more messages in a given protocol by either adding nonce, digital signatures or encrypting messages or replacing one of these sub-messages by another;
- Both adding randomized messages and modifying original messages.

However, there are several constraints over these protocol-manipulating operations when they are performed. Some of them are detailed below:

- The structure properties specific to a particular security protocol must meet the protocol's specification and still be valid and maintained. As stated in section 2.8, a protocol is assumed to consist of a continuous message exchanges. However, inserting randomized messages into a security protocol arbitrarily may destroy some of its structure properties. It turns out that planning techniques can be used to derive a sequence of messages intelligently.
- All information contained in any message must be accessible to or generated by the message sender, or have been received earlier from other participating

entities. Otherwise, it will result in an invalid security protocol. For example, we assume there is an element of encrypted sub-message in a compound message. It is not always possible for the message recipient to access the components of the encrypted sub-message due to the cryptographic restrictions. Actually, some sub-messages are designed to be used as a whole in its following messages for a particular purpose. Thus, the manipulation of a security protocol must get some support from reasoning technologies, which has been witnessed in our work.

- In order to avoid the message reply attack and unauthorized alternation of any protected protocol messages, messages may be required to provide a cryptographic binding between the intended users and the fresh session key established, which is associated with a newly created random number.
- It will result in some inconsistency with the specified protocol specification or violate some general protocol design rules. For example, any message must be able to contribute to the protocol by providing more information such as its liveliness during a protocol run.

Furthermore, message manipulations are also constrained by the following assumptions about a perfect encryption $\{\ldots\}_K$ [Mao 2001]:

- Without knowing about the encrypting key K, $\{\ldots\}_K$ does not provide any cryptanalytic means for finding the plaintext message "…";
- The ciphertext $\{\ldots\}_K$ and maybe some known information about the plaintext "…" do not provide any cryptanalytic means for finding the encrypting key K;
- Without the encrypting key K, even with the knowledge of the plaintext "…", it is still impossible to alter $\{\ldots\}_K$ without being detected during the time of decryption.

5. Some Thoughts on Automatic Security Protocol Generation

Confidentiality, authentication, and integrity are very important security protocol properties. Formal methods are the most commonly used approach to the verification of security protocols. From an artificial intelligence perspective [Massacci 97], the verification of security properties is also a deductive or model-based logical reasoning problem, and protocol design can be seen as a plan that exploits the structure of a security protocol and the message exchanges between its participating entities to achieve a set of given goals. The planning problem can be formalized as a variant of dynamic logic where actions are explicit computations and communication steps between agents. An application of planning techniques to automatically generating a sequence of actions to correct the faults in a system is demonstrated [Lin 98].

A schema consists of one or more primitive messages as discussed in section 2.5 and is accompanied by a set of explicit assertions made in the specification of its fine-grained security properties, which include the freshness of a nonce, the association with an identity, the provision of the integrity of a piece of data, and so on. Schemas are obtained by analyzing all sorts of well-known security protocols, designed by security experts. Schemas are protocol independent and for general-purposes. The proposed intelligent protocol generator is a knowledge-based planner, which makes extensive use of existing protocol design expertise. Its architecture is shown in Fig. 2.

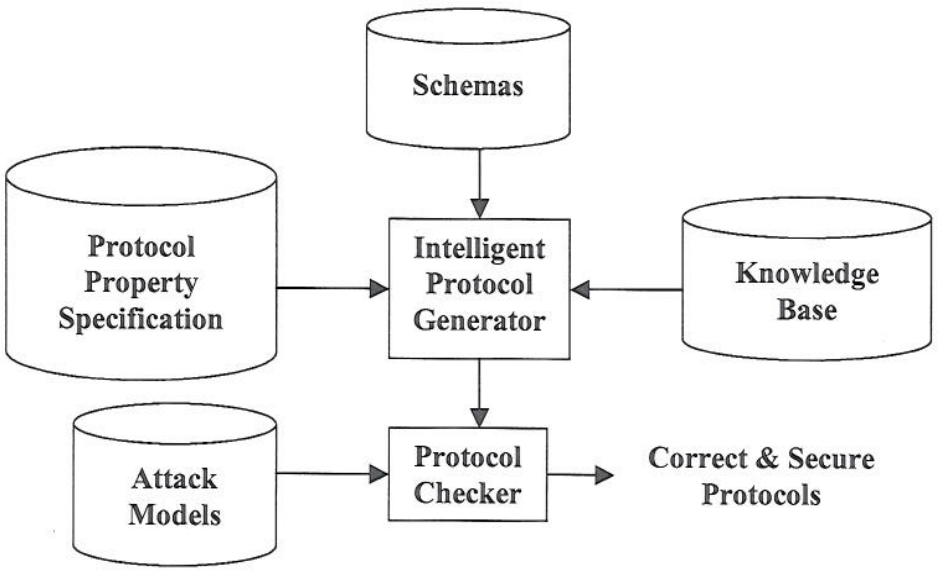
Schemas
Protocol Property Specification
Intelligent Protocol Generator
Knowledge Base
Attack Models
Protocol Checker
Correct & Secure Protocols

Figure 2. Architecture for an Intelligent Security Protocol Synthesizer

Using our protocol generation techniques, we can check whether a given security protocol's run can be attacked, based on a library of attacking models. If none of the attacking models can be applied to the generated security protocol, it is attack-free. Writing sound attacking models is the key to this knowledge-based approach. Even though protocol design experts have got a lot of experience on this, because we do not have a complete set of attacking models, it is still very hard to get a secure protocol attack-free. This is very similar to what anti-virus software faces.

6. Conclusions

In this paper, we first discuss the issues in generating security protocols, and then propose an intelligent approach to constructing security protocols automatically, based on a security property specification, an extensible general-purpose library of schemas, and a library of attacking models. Each schema is defined by specifying intended behaviors using patterns over a sequence of primitive messages and fine-grained security properties.

According to our experiments on generating security protocols automatically, we draw our conclusions as follows:

- It is useful to have several protocol representations for different purposes. It is realized that events-based protocol representation is suitable for quickly excluding inappropriate protocols, and the strands space-based representation may be more suitable for checking if a protocol run is secure and attack-free.
- It is necessary for generating the structure properties of a security protocol efficiently and sensibly based on its specification and previous knowledge about the protocol design of its kind.
- Modeling the behaviors of schemas is the key to automatically generating security protocols. Furthermore, it is expected that the correctness property of

a generated protocol based on schemas can be proved. Contrary to this, modifying a seed security protocol cannot guarantee the correctness of the produced protocol.

- A language for specifying primitive security protocol properties and schema behaviors should be well defined and needs further research.

Acknowledgement

We would like to thank the following people for their technical discussions and support during our protocol generation work: Brian Monahan, Keith Harrison, Pete Bramhall, Martin Sadler, Wenbo Mao, Liqun Chen, Yolanta Beres, and Mike Child.

References

[Dolev and Yao 81] D. Dolev and A. C. Yao, *On the security of public key protocols*, *Proc. of the IEEE 22nd Annual Symposium on Foundations of Computer Science*, pages 350-357, 1981.

[Fabrega et al 98] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman, *Strand spaces: proving security protocols correct*, *Proc. of the 11th IEEE Computer Security Foundations Workshop*, pages 191-230, June 1998.

[Fabrega et al 98] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman, *Strand spaces: why is a security protocol correct? Proc. of IEEE Symposium on Security and Privacy*, pages 160-171, May 1998.

[Jacquemard et al 2000] F. Jacquemard, M. Rusinowitch, and L. Vigneron, *Compiling and verifying security protocols*, *Proc. of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'2000)*, pages 131-160, Nov. 2000.

[Lin 98] Lin A. A Logic-Based Approach to Automated System Management, 6th Int. Conf. on the Practical Application of PROLOG (PAP'98), (1998) 417-425.

[Lowe 95] G. Lowe, *An attack on the Needham-Schroeder public-key authentication protocol*, *Information Processing Letters*, vol. 56, no. 3, pages 131-133, 1995.

[Massacci 97] F. Massacci, *Breaking security protocols as an AI planning problem*, *Proc. of 4th European Conference on Planning (ECP'97)*, pages 286-298, 1997.

[Needham and Schroeder 78] R. Needham and M. Schroeder, *Using encryption for authentication in large networks of computers*, *Communications of the ACM*, vol. 21, no. 12, pages 993-999, December 1978.

[Mao 2001] W. Mao, Cryptographic Protocols, unpublished manuscript, 2001.

[Perrig and Song 2000], A. Perrig and D. Song, *Looking for diamonds in the desert – extending automatic protocol generation to three-party authentication and key agreement protocols*, *Proc. of 13th IEEE Computer Security Foundations Workshop*, pages 64-76, July 2000.

[Song 99] D. X. Song, *Athena: a new efficient automatic checker for security protocol analysis*, *Proc. of the 12th IEEE Computer Security Foundations Workshop*, pages 192-202, June 1999.

[Song et al 2001] D. X. Song, S. Berezin, A. Perrig, *Athena: a novel approach to efficient automatic security protocol analysis*, *Journal of Computer Security*, vol. 9, no. 1-2, pages 47-74, 2001.