International Journal of Security and Its Applications
Vol.8, No.1 (2014), pp.423-432
http://dx.doi.org/10.14257/ijsia.2014.8.1.39

How to Formally Model Features of Network Security Protocols*

Gyesik Lee

Dept. of Computer & Web Information Engineering
Hankyong National University
Anseong-si, Gyonggi-do, 456-749 Korea
gslee@hknu.ac.kr

Abstract

We present a general idea of using formal methods in the verification of security protocols. In particular we show how to formally model intruders and security properties such as secrecy. We demonstrate that applying formal methods can help protocol designers and implementers to improve the quality of security protocols. We also give an example where a formal method is applied to verify of important features in the design of network protocols for vehicular security systems.

Keywords: *Formal modelling, security protocols, secrecy, formal methods*

1 Introduction

Network security protocols are usually based on cryptographic primitives, and analyzing them is one of the most challenging tasks. Indeed fields such as cryptosystems, signature schemes, secure hash functions, transfer mechanisms, and secure multiparty function evaluation methods are used whenever they are necessary to make the protocols more secure. They are also vulnerable to intruders in the network who may have control of one or more network principals. Therefore, network protocols are often subject to non-intuitive attacks. A security protocol must be able to achieve its goals in face of these hostile intruders.

In this sense, designing and implementing security protocols are error-prone and difficult. Moreover, they are supposed to work securely even over insecure networks, i.e., even in the presence of hostile agents that have access to the network.

Let's consider the following security protocol which is the core part of Needham-Schroeder's public key authentication protocol [15] published in 1978. The protocol in Figure 1 describes

$$(M1) \qquad A \to B : \{A, Na\}_{pk(B)}$$
$$(M2) \qquad B \to A : \{Na, Nb\}_{pk(A)}$$
$$(M3) \qquad A \to B : \{Nb\}_{pk(B)}$$

Figure 1. Needham-Schroeder's public key authentification protocol

*This paper is an extended and revised version of Lee [11].

ISSN: 1738-9976 IJSIA

a public key mutual authentication: Two agents Alice(A) and Bob(B) would like to check the authenticity of their partners before they start communicating. Alice creates a nonce and sends it together with her identity to Bob. Bob responds by creating a new nonce and sending it back to Alice with the nonce he received. Finally, Alice confirms the authenticity of Bob.

It seems that the authentication process is secure because the encrypted messages can only be read by a person possessing the corresponding private keys. Unfortunately, this protocol is vulnerable. If an impostor E(compromised) can persuade Alice to initiate a session with him, he can relay the messages to Bob and convince Bob that he is communicating with Alice. This attack was found in 1996, 18 years after the publication by Needham and Schroeder.

Lowe [13] discovered this flaw of the protocol, and it is called man-in-the-middle attack. See Figure 2 below which runs as follows:

(1) Alice sends N_a to E, who decrypts the message with his secret key.

(2) E relays the message to Bob, pretending that Alice is communicating.

(3) Bob sends N_b.

(4) E relays it to Alice.

(5) Alice decrypts N_b and confirms it to E, who learns it.

(6) E re-encrypts N_b, and convinces Bob that he has decrypted it. At the end, Bob falsely believes that Alice is communicating with him, and that N_a and N_b are known only to Alice and Bob.

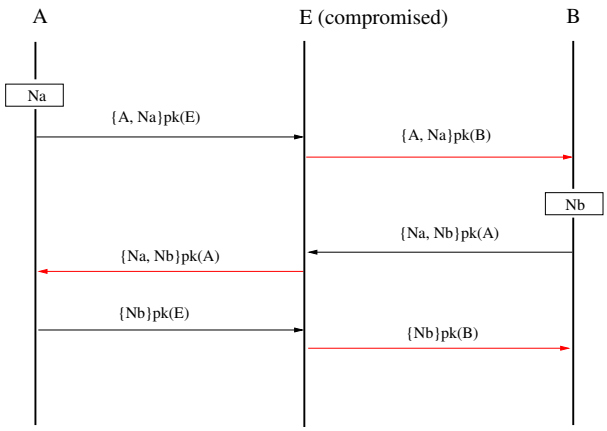

Figure 2. Lowe's man-in-the-middle attack

Interestingly, the attack is found not by a manual inspection, but by applying a formal method. Indeed, verifying security protocols manually is very hard. This is because security protocols are usually based on cryptographic primitives, and their analysis is one of the most challenging tasks. As demonstrated above, security protocols are often subject to non-intuitive attacks.

Fortunately, recent research progress has shown that applying formal methods can help in achieving security goals such as authentication and secrecy in data exchange [1, 2, 16, 3]. Moreover, the ISO/IEC 29128 [10] states a standard which provides definitions of different protocol assurance levels where the importance of the application of formal methods are

explicitly mentioned.

It seems nowadays inevitably required to verify that a security protocol satisfies its requirements based on a formal method. A formal method is based on a combination of a mathematical or logical model of a system and its requirements. Actually, the application of formal methods to cryptographic protocol analysis has been investigated since almost 30 years [14]. An important area is the development of tools for automatic verification of security protocols allowing unbounded number of sessions based on some intruder models such as Dolev-Yao model [8]. There are many tools for automated verification of security protocols such as ProVerif [4, 5] and Scyther [7], to name a few. See also Lee et al. [12] for an application of such tools.

In this paper, we present a general idea of using formal methods in the verification of security protocols. The following sections show how to formally model intruders and security properties, in particular secrecy.

The rest of the paper is organized as follows. Section 2 presents a general idea of modelling protocol specification. Section 3 describes how to model operating environment. Modelling of secrecy properties is introduced in Section 4. In Section 5, we introduce an example where a formal method is applied to verify of important features in the design of network protocols. We conclude in Section 6.

2 Modelling of protocol specification

2.1 Languages

A language $\mathcal{L} = (\mathcal{V}, \mathcal{C}, \mathcal{R}, \mathcal{F})$ for security protocols should be given. It consists of symbols for constructing terms like nonces, roles, functions, etc: a set $\mathcal{V}$ of variables to store received messages, a set $\mathcal{C}$ of local constant symbols for such as nonces and session keys, a set $\mathcal{R}$ of role name symbols, a set $\mathcal{F}$ of function symbols for such as global constants or hash functions. $\mathcal{F}$ includes some special binary function symbols for composition of pairs, for decompositions of pairs, for encryption of a message, and for decryption of a encrypted message. They are denoted by $(\cdot\,,\cdot)$, π_i, *enc*, and *dec*, respectively.

Given a language, $\mathcal{L} = (\mathcal{V}, \mathcal{C}, \mathcal{R}, \mathcal{F})$, terms are defined inductively. Every variable $v \in \mathcal{V}$ is a term. Every local constant $c \in \mathcal{C}$ is a term. Every role name $r \in \mathcal{R}$ is a term. For every function symbol $f \in \mathcal{F}$ of arity n, if $t_1, \ldots, t_n$ are terms, so is $f(t_1, \cdots, t_n)$. If t_1 and t_2 are terms, (t_1, t_2) stands for composition of pairs and $\{t_1\}_{t_2}$ for encryption of t_1 by using t_2. It is assumed that encryption is perfect. Fresh variables and fresh nonces can be generated whenever it is necessary. It is assumed that the cryptographic primitives such as *enc* and *dec* are perfect. Only their relevant properties are assumed to be known.

There is a set $\mathcal{E}$ of equations over terms which specify identities among terms such as equations respecting Diffie-Hellman key assignments, etc. The equations are formulated in general forms using variables and function symbols which can be instantiated by arbitrary terms. $t_1 = t_2 \in \mathcal{E}$ denotes the fact that the equation $t_1 = t_2$ is an instance of an equation in $\mathcal{E}$.

2.2 Protocol specifications

Given a language, $\mathcal{L} = (\mathcal{V}, \mathcal{C}, \mathcal{R}, \mathcal{F})$, a protocol specification, shortly a protocol, P describes the behavior of each of the roles such as initiator, responder, key server, etc. In the

specification, the behavior of each role is formalized as a transition system describing how to create messages, how to react to the received messages, and how to manipulate them.

Sending and receiving Two predicate symbols exist. *Send* for sending messages and *Receive* for receiving messages. Their arity depends on the decision about putting in the names of the communicating roles, communication channels, privacy or publicity of the channels, etc. as arguments.

Role specifications A role specification describes the behavior of a role, which is represented as a finite sequence of sending and receiving events. An initial knowledge also belongs to the role specification. The initial knowledge of a role contains the names of other roles, public keys of all roles, his own secret key, etc.

2.3 Computation

During any two events, each agent performs some computations. What exactly the computations do, depends on the protocol model. Below are some examples of computations.

Pattern matching It can be assumed that any agent in a role r could see the pattern of any message: nonces, agent names, session keys, pairs, encrypted messages, etc.

Knowledge inference Based on the pattern-matching, any Therefore, agent including the intruder can infer new knowledge from his initial knowledge together with the received messages. The knowledge inference can be inferred formally as follows: is reflected by the following inference rules in Figure 3.

$$\frac{t \in \mathcal{K}}{\mathcal{K} \vdash t} \qquad \frac{\mathcal{K} \vdash t_1 \quad t_1 = t_2 \in \mathcal{E}}{\mathcal{K} \vdash t_2}$$

$$\frac{\mathcal{K} \vdash t_1 \quad \mathcal{K} \vdash t_2}{\mathcal{K} \vdash (t_1, t_2)} \qquad \frac{\mathcal{K} \vdash (t_1, t_2)}{\mathcal{K} \vdash t_1} \qquad \frac{\mathcal{K} \vdash (t_1, t_2)}{\mathcal{K} \vdash t_2}$$

$$\frac{\mathcal{K} \vdash t \quad \mathcal{K} \vdash k}{\mathcal{K} \vdash \{t\}_k} \qquad \frac{\mathcal{K} \vdash \{t\}_k \quad \mathcal{K} \vdash k^{-1}}{\mathcal{K} \vdash t}$$

Figure 3. Knowledge inference rules: $\mathcal{K}$ denotes a knowledge set and k^{-1} is the only key for decryption of a message encrypted with k.

Readability test When a message is received, each agent would check its readability using his knowledge. The readability property of a term t based on his knowledge $\mathcal{K}_r$ can be implemented into the protocol specification as follows:

(1) Every variable $v \in \mathcal{V}$ is readable.

(2) Every term t such that $\mathcal{K}_r \vdash t$ is readable.

(3) A pair (t_1, t_2) is readable when t_1 is readable based on $\mathcal{K}_r \cup \{t_2\}$ and t_2 is readable based on $\mathcal{K}_r \cup \{t_1\}$.

(4) An encrypted message $\{m\}_k$ is readable when either $\mathcal{K}_r \vdash \{m\}_k$, or $\mathcal{K}_r \vdash k^{-1}$ and m is readable.

Then a role specification is well-formed if each term in a receiving event is readable based on the role knowledge which has grown after each receiving event. A formal definition of well-formed role specifications will be omitted, but the relationship between the role knowledge and an receiving event will be described formally below in the part of operational semantics. A protocol said to be well-formed when each role specification is well-formed, and only well-formed protocols are to be considered.

3 Modelling of operating environment

3.1 Intruder model

The network can be partially or completely under control of an intruder. Based on his knowledge, he can e.g. catch, eavesdrop, or fake messages. He can also interrupt or disturb the protocol running.

3.2 Intruder knowledge

The initial knowledge $\mathcal{K}_A^0$ of an agent A in a role consists of e.g. the names and public keys of all agents and his secret key of his role. The initial intruder knowledge $\mathcal{K}_I^0$ consists of the initial knowledge of all untrusted agents including their secret keys. The knowledge of an agent including the intruder will grow during the running of the protocol whenever he receives or catch messages.

3.3 Configuration state

The configuration state at some point during running a protocol P is composed of the local intruder knowledge and the local knowledge of every possible agent A_n, where n varies over natural numbers. The list of agents is made infinite such that it reflects the fact that the intruder could initiate new session at any step and perform unlimited sessions. In the initial state, every agent is in his initial state, i.e. his initial knowledge and initial control state. If an agent A_n is not active yet, then his initial knowledge is empty.

The operational semantics of the protocol P is the description how configuration states changes during the protocol running. Below are three rules which constitute the operational semantics of the protocol P.

(1) If an agent performs a new instance event, then he adds a new role instance to his state. If the agent is compromised, then he shares all the knowledge with the intruder.

(2) If an agent performs a sending event, the sent message m is added to the intruder knowledge. Then he moves to some state where he is waiting for another sending or receiving event, or stops.

(3) If an agent performs a receiving event, the agent performs some computations. Depending on the computation result, he moves to some state where he is waiting for another sending or receiving event, or stops.

New instances If an agent performs a new instance event, then he adds a new role instance to his state. This implies that unlimited number of role instances is possible. The agent who performs this role gets then the initial knowledge assigned to the role. If the agent is compromised, then he shares all the knowledge with the intruder.

Sending event If an agent performs a sending event, the sent message m is added to the intruder knowledge. Then he performs some computations, and depending on the computation result, he moves to some state where he is waiting for another sending or receiving event, or stops.

Receiving event If an agent performs a receiving event, the agent performs some computations. Depending on the computation result, he moves to some state where he is waiting for another sending or receiving event, or stops. The computations include e.g. the readability test. If the received message m passes the readability test, the message will be added to the knowledge of the agent.

3.4 Traces

A state transition is the conclusion of finitely many applications of the rules above, starting from the initial state. A trace of a protocol P is the description of any possible state transition starting from the initial state:

$$(\mathcal{K}_I^0, \langle \mathcal{K}_{A_n}^0 \rangle_n) \xrightarrow{m_1} \cdots \xrightarrow{m_\ell} (\mathcal{K}_I^\ell, \langle \mathcal{K}_{A_n}^\ell \rangle_n) \xrightarrow{m_{\ell+1}} (\mathcal{K}_I^{\ell+1}, \langle \mathcal{K}_{A_n}^{\ell+1} \rangle_n) \xrightarrow{m_{\ell+2}} \cdots$$

Here ℓ denotes the ℓ-th transition step and m_ℓ is the exchanged message at the ℓ-th transition. $\mathcal{K}_A^\ell$ and $\mathcal{K}_I^\ell$ stand for the local knowledge of the agent A and of the intruder I, respectively, at the ℓ-th step. $\langle \mathcal{K}_{A_n}^\ell \rangle_n$ is an abbreviation for the infinite list of $\mathcal{K}_{A_n}^\ell$, where n varies over natural numbers. The relationship between m_ℓ and m_i, $i < m$, is decided by the protocol specification.

4 Modelling of secrecy property

Secrecy expresses that certain information cannot be revealed to any other agent or the intruder except the honest agents who have run the protocol, even though the protocol is executed in an untrusted network. More formally, a protocol P satisfies secrecy of a message m among some honest agents $A_{n_1}, ..., A_{n_p}$ if and only if in an arbitrary trace, m cannot be inferred from the knowledge of anybody else, i.e.,

$$\mathcal{K}_I^\ell \nvdash m \quad \text{and} \quad \mathcal{K}_A^\ell \nvdash m$$

for any ℓ, where A is not one of $A_{n_1}, \ldots, A_{n_p}$.

Secrecy defined as it stands can be referred as weak secrecy, since it does not care about partial disclosure of the message content. There are also probabilistic secrecy, indistinguishability, etc. But they are out of scope here.

5 Formally verifiable features for secure communication

In an overview paper called *State of the Art: Embedding Security in Vehicles*, Wolf et al. [19] gave a general state-of-the-art overview of IT security in vehicles and describe core security technologies and relevant security mechanisms.

Lee et al. [12] used a tool called ProVerif [6] to show that a formal verification of important features required in common for secure communication is not just desirable, but can be realized using an existing tool for fully automatic verification of security protocols. The above mentioned important features are listed in [19]:

(P1) Only valid controllers can communicate.

(P2) All unauthorized messages are to be processed separately or immediately discarded.

(P3) Every communication is based on encryption and authentication in order to provide confidentiality and authenticity of exchanged data.

(P4) A single successful attack should not endanger the whole system.

(P5) It is desirable that a software security module can be verified formally.

Automatic or semi-automatic protocol verification for bounded or unbounded number of sessions has become the main object of formal analysis of security protocols. In case of unbounded number of sessions, it is typically based on language-based techniques such as typing or abstract interpretation. There are many (semi-) automatic tools for protocol verification.

ProVerif [6] is a leading automatic cryptographic protocol verifier in the so-called Dolev-Yao model [8]. Protocols are written by Horn clauses, and ProVerif checks full automatically whether some Horn clauses are derivable from the Horn clauses representing the protocol. It is sound in the sense that the security properties that it proves are really true. But there could be false attacks due to approximations. Approximations occur during the translation of protocols written in the applied pi-calculus into Horn clauses. The idea is based on a simple correspondence between traces of communications and exchanges of information: received messages as the antecedents and sent message as the conclusion of a Horn clause.

ProVerif can handle many cryptographic primitives like shared- and public-key cryptography (encryption and signatures), hash functions, and Diffie-Hellman key agreements. Thanks to some approximations, it can handle an unbounded number of sessions of the protocol and an unbounded message space and can prove secrecy, authentication, strong secrecy, equivalences between processes that differ only by terms.

We refer to Blanchet [6] for a good comprehensive introduction into ProVerif. We also refer to Fournet and Abadi [9] which gives the analysis of a protocol for private authentication in the applied pi-calculus. A brief overview of other verifiers can be found e.g. in Cremers [7].

6 Conclusion

In the last years lots of important progresses has been made in applying formal methods to verifying security protocols. And fully automated tools such as ProVerif and Scyther have been invented for the verification security protocols. This paper presented a general idea of using formal methods in the verification of security protocols. Moreover, we introduced an

concrete example where a formal method can be successfully applied. We also showed how to formally model intruders and security properties such as secrecy. This demonstrated that applying formal methods can help protocol designers and implementers to improve the quality of security protocols. Further results of applying formal approaches could be found e.g. in Singh et al.[17] and Wenjun and Bin [18].

Acknowledgments.

The author is grateful to Akira Otsuka for having intensive discussions at AIST. This work was supported by the National Research Foundation of Korea[NRF] grant funded by the Korea government[MEST] [No. 2012-030479 and No. 2012-044239].

References

[1] M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *ACM Conference on Computer and Communications Security*, pages 36–47, 1997.

[2] M. Abadi and P. Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). *J. Cryptology*, 15(2):103–127, 2002.

[3] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub. Implementing TLS with Verified Cryptographic Security. In *IEEE Symposium on Security and Privacy*, pages 445–459. IEEE Computer Society, 2013.

[4] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *CSFW*, pages 82–96. IEEE Computer Society, 2001.

[5] B. Blanchet. A Computationally Sound Mechanized Prover for Security Protocols. *IEEE Trans. Dependable Sec. Comput.*, 5(4):193–207, 2008.

[6] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[7] C. J. F. Cremers. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *CAV*, volume 5123 of *LNCS*, pages 414–418. Springer, 2008.

[8] D. Dolev and A. C.-C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[9] Cédric Fournet and Martín Abadi. Hiding Names: Private Authentication in the Applied Pi Calculus. In *ISSS 2002*, volume 2609 of *LNCS*, pages 317–338. Springer, 2003.

[10] ISO/IEC 29128. *Information technology – Security techniques – Verification of cryptographic protocols*, 2011. `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=45151`.

[11] G. Lee. Formal Modelling of Network Security Properties. In *Secutiry Technology, SecTech 2013*, Advanced Science and Technology Letters, Vol 29, pages 25–29, SERSC, 2013.

[12] G. Lee, H. Oguma, A. Yoshioka, R. Shigetomi, A. Otsuka, and H. Imai. Formally Verifiable Features in Embedded Vehicular Security Systems. In *First IEEE Vehicular Networking Conference, VNC 2009*, IEEE Xplore database, 2009.

[13] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. *Software - Concepts and Tools*, 17(3):93–102, 1996.

[14] C. Meadows. Open Issues in Formal Methods for Cryptographic Protocol Analysis. In V. I. Gorodetski, V. A. Skormin, and L. J. Popyack, editors, *MMM-ACNS*, volume 2052 of *LNCS*, page 21. Springer, 2001.

[15] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Commun. ACM*, 21(12):993–999, 1978.

[16] A. Pironti, D. Pozza, and R. Sisto. Automated Formal Methods for Security Protocol Engineering. In *Cyber Security Standards, Practices and Industrial Applications: Systems and Methodologies*, pages 138–166. IGI Global, 2011.

[17] M. Singh, M. Singh Patterh, and T. Kim. A Formal Policy Oriented Access Control Model for Secure Enterprise Network Environment . In *International Journal of Security and Its Applications*, 3(2):1–14, SERSC, 2009.

[18] T. Wenjun and H. Bin. A Stronger Formal Security Model of Three-party Authentication and Key Distribution Protocol for 802.11i. In *International Journal of Security and Its Applications*, 6(4):163–174, SERSC, 2012.

[19] Marko Wolf, André Weimerskirch, and Thomas Wollinger. State of the Art: Embedding Security in Vehicles. *EURASIP Journal on Embedded Systems*, 2007(Article ID 74706, 16 pages), 2007.

