Jan Długosz University in Częstochowa
Scientific Issues, Mathematics XV, Częstochowa 2010

ON SOME SPECIFICATION LANGUAGES OF CRYPTOGRAPHIC PROTOCOLS

Paweł Dudek, Mirosław Kurkowski

Institute of Computer and Information Sciences
Częstochowa University of Technology
ul. Dąbrowskiego 73, 42-200 Częstochowa, Poland
e-mail: pdudek@icis.pcz.pl, mkurkowski@icis.pcz.pl

Abstract. A key element of the security systems in computer networks are cryptographic protocols (CP). These protocols are concurrent algorithms used to provide relevant system security goals. Their main purpose is, for example, a mutual authentication (identification) of communicating parties (users, servers), distribution of new keys and session encryption. Literature indicates numerous errors in protocol constructions. Thus, there is a need to create methods for CP specification and verification.

In this paper, we investigate a problem of CP specification. The paper discusses the so-called Common Language – the simplest language of CP specification and HLPSL – a specification language used in the European verification project Avispa. Finally, we introduce PTL – the new language developed for CP specification which allows fully automatic verification.

1. Introduction

It is well known that today each IT system and computer network must meet certain security properties [6]. CP are now commonly used in various applications (banking, emails, encrypted Web pages, instant messaging networks, etc.) for achieving security goals. They are also widely used as essential components of larger systems such as communication protocols for wider application. Good examples are the systems of Kerberos, SSL and Zfone. A pioneering role in the area of CP has the paper published in 1978 by Needham and Schroeder [2]. In their work the authors presented main ideas of applying cryptographic techniques in order to solve problems related to the authentication of communicating parties in communication networks. Suggested layouts of authentication protocols can use symmetric and asymmetric cryptography.

CP are concurrent algorithms, designed to attain certain specific objectives during the transfer, including the transactions carried out electronically. In general, these protocols are algorithms whose implementations are performed in a concurrent way and may be used for cooperating computers, computer networks or simply across multiple CPUs. This is a significant difference between them and ordinary sequential algorithms. CP can also specify concurrent processes as communicating sequential processes with each other from time to time through the exchange of data (the parameters) or the use of common resources. Cryptographic protocols are those concurrent processes, which work using cryptographic algorithms.

A specification of any cryptographic protocol has to contain:

- the number of parties involved in the protocol,
- the nature of the participation of the parties,
- the goal of the protocol,
- actions comprised in the implementation of the protocol.

Basic security goals which CP need to ensure are the following:

- mutual authentication (confirmation of identity) of communicating parties,
- confidentiality of transmitted information,
- integrity of transmitted data,
- distribution of session key.

Actions performed during the execution of the protocol can be divided into internal and external ones. External actions are those which rely on the mutual exchange of transmitted information. Description of those actions will specify sources of any sent message (senders), recipients of sent messages and their contents. It must also indicate, respectively, which part of the sent letter has to be encrypted and how. Internal actions are all the other actions that each party must perform on its own during the execution of the protocol. As examples, one can give generating new, confidential information, encrypting and deciphering cryptograms, comparing data or performing mathematical operations on locally held data.

Applying cryptographic protocols in order to ensure adequate security purposes in computer systems requires special attention with regard to the correctness of their executions. Incorrect work of protocols can lead to different sorts of losses of users resources [4]. Cryptographic protocols are usually short and not too complicated in their structure, so often entirely informal arguments are used to justify that they operate properly and to all system users that the protocol actually does what it is expected to do [2].

In most cases, however, it is difficult to imagine all possible executions of these complex systems. This becomes especially difficult when dealing with programs that are executed concurrently on many computers, where the partial results of these performances may affect the implementation of the next instruction. For these reasons, the method of verifying the correctness of software systems is constantly an extensively developed area of computer science.

Basically, we can distinguish two main groups of verification methods:

1. Testing of real or virtual systems (simulations).
2. Formal modeling and verification.

In the first case, the verification process simply consists in testing the systems already implemented or simulating their performances by computers (eg. virtual machines). After carrying out several such tests or simulations, unfortunately, we can only say that so far the implementation of all programs works properly.

The second direction of research, namely formal modeling and verification, involves creating special mathematical structures which model processes taking place during protocol executions. It is therefore, in some sense, the creation of a new, specific types of simulation. However, as numerous examples show, this type of modeling can sometimes prove formally that certain undesirable behavior of the system will never occur.

Creating mathematical structures simulating the implementation of cryptographic protocols is not an easy process. This work, however, requires to use a specially constructed languages for protocols specification. In [2] a simple language for the specification of protocols has been applied, known simply as Common Language (CL) [1, 6]. As an example, we show below protocol specification using CL and some information about it.

2. Common Language – CL

Common Language has never been formalized. However, the grammar of the basic version is not too complicated. The protocol is described as a sequence of steps, specifying the sender of the message, recipient and content of the sent letter [1].

Each step is specified as follows:

$$A \rightarrow B \ : \ M,$$

where A is, of course, the sender of the message, B is the recipient and M is the message.

The grammar of messages is the following:

$$M \ : \ A \ | \ T \ | \ K \ | \ N \ | \ L \ | \ M, M \ | < M >_K,$$

where A belongs to a set of users, K to a set of cryptographic keys, T to the set of timestamps, L is a life time of T. The keys used in the specification, of course, may be symmetric or asymmetric. In the first case, we denote by K_{AB} the key, where A and B are their owners; in the second case, the symbol K_A denotes the public key of A and K_A^{-1} its private key. M, M is simply a concatenation of messages, and by writing $< M >_K$ we understand the ciphertext M encrypted with the key K.

Here, as an example, we show a specification for some version of Kerberos Protocol using the Common Language. The basic version of this protocol is as follows: we have two parties A and B, which share the server S with different secret keys. The main goal of this protocol is to generate by A a session key in order to conduct communication with B.

Protocol specification:

1. $A \ \rightarrow \ S \ : \ A, B,$
2. $S \ \rightarrow \ A \ : < T, L, K, B >_{K_{AS}}, < T, L, K, A >_{K_{BS}},$
3. $A \ \rightarrow \ B \ : < A, T >_K, < T, L, K, A >_{K_{BS}},$
4. $B \ \rightarrow \ A \ : < T >_K.$

In the first step of the protocol, the user A sends to the server S a message consisting of its identifier and the name of B. In this way, S possesses information with whom A wants to communicate. In the second step, the server generates two messages with a timestamp T, the ticket duration L and a newly generated, random session key K. S encrypts all of them using a secret key shared with B. Then it gets a timestamp, the duration L and the identifier B, and encrypts everything using secret key shared with A. Next, S sends two encrypted messages to A. In the third step, A generates a message containing its identifier and timestamp, encrypts them using the session key K newly obtained from S and sends it to B. A also sends to B a message encrypted by the server using a common key for B and S. Then B possesses the key K and creates a message consisting of the timestamp T, encrypts it using K and sends it to A.

Executing a protocol assumes the existence of an ideal clock allocating time in compliance with clocks of all users of the server. This is achieved by synchronizing every few minutes to a secure server clock time. The key server S needs to remember all the keys that it shares with users. However, the session key is created for the purpose of communication between A and B, then the server forgets about the result.

Obviously, as one can see from the above example, CL is very simple and it is probably difficult to imagine a simpler protocol specification language. However, it is important to note that it requires additional information about, for example, the description of internal actions during the protocol execution, including generating new elements such as keys, nonces (pseudo-randoms numbers generated for a single session) or timestamps. There is also no information about how users compose sent messages. That is the reason why CL cannot be used in fully automatic verification.

3. HLPSL Language

Currently, the world's most recognizable system of formal verification of cryptographic protocols is the AVISPA system (Automated Validation of Internet Security Protocols and Applications) [7, 10]. This system was created through cooperation of several institutions: Universities of Genova, Zurich, Nancy and subsidiaries of Siemens in Munich. For this project a special role-based, high-level language HLPSL for CP specification (High Level Protocol Specification Language) was created.

In HLPSL each participant has a defined primary role (basic role), which is described by various parameters related to the behavior of participants during the execution protocol. These roles define how users can transfer their information during the executions of the protocol. Data included in those roles determine the information, which a participant can use initially, and the initial state of the knowledge. Additionally, roles describe how the users knowledge might change during the execution of the protocol. The specification given in basic roles can be used later by one or more users who can play a particular role in the protocol execution. Then, to create the composed roles we describe how the individual members communicate among themselves by means of repeated basic roles. In this way, we obtain a specification schema for data exchange during the whole protocol execution. Roles are independent processes, which have a specific name, replaced by the value of initialization parameters, also contain local declarations. Actions of simple roles are specified in order to describe transitions in the form of a change in the role depending on the events occurred, while the complex roles determine the way in which pre-defined roles are combined.

The HLPSL specification also defines additional parameters of verification. Additionally, in a batch file there will be determined security properties which are to be examined and the size of the protocol performances in the needed searching space. The declaration and definition of the objectives which we want to achieve during the verification takes place in an another special section of specification.

HLPSL allows testing of the following security properties:

- maintaining the confidentiality of the information,
- strong user authentication on the basis of the message,
- weak user authentication based on a certain message.

Fixed data or variable used in specification must have assigned a unique type. The list of examples of variables is the following:

- *agent* – for users identifiers, for the intruder the letter *i* is reserved,
- *public_key* – for public keys of agents. Given a public key *pk* (resp. private), its reversed private (resp. public) key is obtained through the structures *inv(pk)*,
- *symmetric_key* – for keys used in symmetric encryption,
- *nat* – for the scope of variables of this type the natural numbers are used. *Nat* type is usually used to describe states,
- *protocol_id* – for identifiers used in the studied properties,
- *message* – for representing any message,
- *text* – for nonces.

Correct messages are defined as the submission of the concatenation operation '.' and/or encryption '_' (message_key) of basic data types. There is no difference between the descriptions of symmetric and asymmetric encryptions. Assuming that we have a type of agent, the agent A, the nonce N_a and the symmetric key K, the following messages are correct:

1. N_a – nonce N_a is a message,
2. $A.N_a$ – the message containing the identifier of agent A with a value N_a,
3. $\{A.N_a\}_K$ – the proper message encrypted with the key K.

A channel is a variable that connects communicating parties and exchanges messages between them. HLPSL's channels contain intruder acting in that channel. The model available in HLPSL is the well known Dolev-Yao model [3] (denoted by *dy*) in which the attacker is a network of canals.

The four predefined goal predicates listed above contain the following information:

- *secret(E,id,S)*: declares the information E as a secret shared by the agents from a set S; this secret will be identified by the constant *id* in the goal section;
- *witness(A,B,id,E)*: for a (weak) authentication property of A by B on E, declares that an agent A is witness for the information E; this goal will be identified by the constant *id* in the goal section;
- *request(B,A,id,E)*: for a strong authentication property of A by B on E, declares that an agent B requests checking the value E; this goal will be identified by the constant *id* in the goal section;
- *wrequest(B,A,id,E)*: similar to *request* property, but in this case for a weak authentication property.

Summing up, the language HLPSL is a very complex language which allows the full specification of cryptographic protocols. It is clear, however, that it has been specially designed deliberately to be used by a specific tool, namely the verification system Avispa. That is why one can reflect on its versatility. It is obvious that if we would like to apply the specification of the protocol in HLPSL in another tool in the study and application, we need appropriate special translators.

4. VerICS system and PTL language

VerICS [5, 11] is an original tool for automatic or semi-automatic verification of concurrent systems. The system allows verification of various properties of systems containing the temporal aspects. One module of VerICS is solely devoted for the CP verification. The results obtained by the VerICS team so far are competitive to the other results obtained in Europe and worldwide [7, 8, 9, 10, 12]. In the case of CP verification, a special mathematical model of CP executions has been developed. This model allows testing various executions of CP. The formalism has been designed so as to be able to identify accurately the correct sequences of steps protocol performances that make up executions performances.

For this project, a simple specification language called ProTocol Language (PTL) has been proposed. In this approach, the protocol is defined as a sequence of steps, and each of them is defined as an ordered pair of the form: (α_1, α_2). The component α_1 defines external actions of the protocol (messaging), while the component α_2 defines internal ones.

Both components contain basic and complete information about the specified protocol. More precisely:

$$\alpha_1 = (P, Q, M), \alpha_2 = (t, X, G, \tau),$$

where P is the step initiator, Q is the owner and M is the sent message. So far, there are no differences between this specification and the specification in the CL language. In this approach, we have yet more information: t is a variable indicating time when the step's execution starts, X is a set of information needed to compose a message, G is a set of sensitive information generated for a given step, and τ is a time constraint ensuring that each step can be performed. This specification allows precise determination of not only the external actions of the protocol but also internal ones.

The message grammar is the same as the corresponding grammar in CL:

$$M \ : \ A \ | \ T \ | \ K \ | \ N \ | \ L \ | \ M, M \ | < M >_K .$$

In addition, we specify time constraints according to the following grammar:

$$\tau \ : \ false \ | \ true \ | \ t - T \leq L \ | \ \tau_1 \wedge \tau_2.$$

As an example of a full protocol specification in the PTL language, we give now a formal description of the Kerberos Protocol mentioned above.

Protocol specification in PTL is as follows:

1. The first step $(\alpha_1, \ \alpha_2)$, where

$$\alpha_1 = (A; S; A, B), \ \ \alpha_2 = (t_1, \{A, B\}, \emptyset, true).$$

2. The second step $(\beta_1, \ \beta_2)$, where

$$\beta_1 = (S; A; < T, L, K, B >_{K_{AS}}, < T, L, K, A >_{K_{BS}}),$$

$$\beta_2 = (t_2, \{T, L, K, A, B, K_{AS}, K_{BS}\}, \{T, K\}, t_2 - T \leq L).$$

3. The third step $(\gamma_1, \ \gamma_2)$, where

$$\gamma_1 = (A; B; < A, T >_K, < T, L, K, A >_{K_{BS}}),$$

$$\gamma_2 = (t3, \{T, K, A, < T, L, K, A >_{K_{BS}}\}, \emptyset, t_3 - T \leq L).$$

4. The fourth step (δ_1, δ_2), where

$$\delta_1 = (B; A; < T >_K),\ \delta_2 = (t_4, T, K, \emptyset, t_4 - T \leq L).$$

Note that this specification gives precise information on both external and internal actions of the protocol. It also determines precisely the time conditions which users need to fulfill.

From the technical point of view, a PTL specification file contains only information needed for further steps of verification process. This file consists of two main parts. In the first one, we have basic information about numbers of considered users and protocol steps. The second one contains specification of all protocol steps in the way mentioned before. In next lines, we have the description of pairs specifying steps of the protocol.

5. Conclusion

Effective methods of specification and verification of cryptographic protocols are an important problem of applied cryptography. In this paper, we have discussed a few basic languages developed for CP specification. We have presented the Common Language – the simplest language for protocol specification, HLPSL – the language used in the European project Avispa, and the PTL language. Investigations in this area are still in progress. The next steps consists in expanding the expressive power of the PTL language to describe a larger class of protocols and delays in the network occurring during the transmission of information.

References

[1] *Security Protocols Open Repository* (SPORE): http://www.lsv.ens-cachan.fr/Software/spore/

[2] R. Needham, M. Schroeder. Using encryption for authentication in large networks of computers. *Comm. ACM*, **21** (12), 993–999, 1978.

[3] D. Dolev, A. Yao. On the security of public key protocols. *IEEE Trans. Information Theory*, **29** (2), 198–208, 1983.

[4] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: *Proc. TACAS*, pp. 147-166, Springer, 1996.

[5] P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, A. Zbrzezny. VerICS: A tool for verifying timed automata and estelle specifications. In: *Proc. 9th Int. Conf. TACAS'03*, vol. 2619 of LNCS, pp. 278–283, Springer, 2003.

[6] A. Menezes, P. van Oorschot, S. Vanstone. *Kryptografia stosowana*, WNT, 2005.

[7] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.C. Heam, O. Kouchnarenko, J. Mantovani, S. Modersheim, D. von Oheimband M. Rusinowitch, J. Santiago, M. Turuani, L. Vigano, L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In: *Proc. 17th Int. Conf. Computer Aided Verification (CAV'05)*, vol. 3576 of LNCS, pp. 281–285, Springer, 2005.

[8] M. Benerecetti, N. Cuomo, A. Peron. TPMC: A model checker for time-sensitive security protocols. In: *Proc. 2007 High Performance Computing and Simulation Conf. (HPCS 2007)*, pp. 742-749, Prague, 2007.

[9] M. Kurkowski, W. Penczek. Verifying security protocols modelled by networks of automata. *Fund. Informaticae*, **79** (3-4), 453–471, 2007.

[10] A. Armando, L. Compagna. Sat-based model-checking for security protocols analysis. *Int. J. Information Security*, **7** (1), 3–32, 2008.

[11] M. Kacprzak, W. Nabiałek, A. Niewiadomski, W. Penczek, A. Półrola, M. Szreter, A. Zbrzezny. Verics 2008 – a model checker for high-level languages. *Artificial Intelligence Studies* **5** (28), 131–140, 2008.

[12] M. Kurkowski, W. Penczek. Verifying timed security protocols via translation to timed automata. *Fund. Informaticae*, **93** (1-3), 245–259, 2009.