

Bindings for Security Protocol Message Composition

Genge Bela¹, Haller Piroska²

Abstract — We present a method for creating security protocols, based on message composition. The novelty of our approach is that it uses existent protocols to build new ones. Another benefit of the approach is that it maintains at the same time the security properties of all the involved protocols. The approach is based on an extension of the strand space model, which allows an atomic treatment applied on all messages. Using the proposed strand-based model and composition algorithms we illustrate the approach by creating a new protocol from two existing security protocols.

Keywords — Composition, Security Protocols, Strand Spaces.

I. INTRODUCTION

Security protocols are communication protocols in which participants use encryption to send each other encoded information. With the rapid growth of the Internet and a desperate need to secure communication, in the last few decades the attention of many researchers has been focused towards analyzing security protocols [1]-[2]-[3]-[4]-[5]-[6].

Recently, there have been several proposals developed to help the process of security protocol design using formal methods and tools [7]-[8]-[9]-[10]-[11]-[12]-[13]. Most of the proposed techniques use a modular approach in the design process, where the user is given a set of small protocols from which more complex protocols can be constructed, process also known as *composition* [9]-[10]-[11].

In the existing composition techniques, authors mainly deal with the sequential and parallel composability of security properties viewed as a set of information transmitted over messages. However, the composition of message components has not been addressed in a proper manner, meaning that users have to solve the problem of resulting message term duplicates on their own.

Solving this problem, apparently insignificant, can lead to protocols executed in half the time the original, composed protocols are executed in. In addition, the composition process can lead to multiple results, which

must be carefully analyzed on a message level to increase protocol performance.

In this paper, we address the composability problem at the message level. We introduce the concept of *binding* to capture the link between message terms. Based on these, we propose a formal specification of security protocols. The composition process is reduced to analyzing similar bindings, inserting new ones or simply extending existent bindings.

However, changing the position of bindings in a security protocol can be rather problematic if we consider that a protocol may also consist of messages that cannot be decrypted by the receiving party, because the key is not in his possession. Also, there are cases when the structure of a term must not be changed because, for example, the receiving party does a byte-to-byte verification of the received term, without actually decrypting it.

To solve these problems, the proposed specification contains not only the messages exchanged by parties, but also a set of knowledge attached to each specified role so that in the composition process we can check which messages can be constructed and sent.

The rest of the paper is structured as follows. Section II introduces the concept of *bindings*. Based on these, we provide a formal specification of security protocols in section III. In section IV, we present a simplified composition algorithm based on the proposed formal specification. In section V we provide an example composition of two protocols. We end with a conclusion and future work in section VI.

II. FORMALIZING BINDINGS

A. Existent bindings

A closer examination regarding the structure of security protocol messages reveals a tight connection between components. For example, let us consider the following message extracted from Lowe's modified "BAN concrete Andrew Secure RPC" [14] protocol:

$$A \rightarrow B: \{Na, K'ab, B\}_{Kab}$$

Here, role A is sending to role B a session key. B is ensured that the message is for him by including B's name in the message. Role B is also ensured about the fact that the message is fresh, because it contains the newly generated nonce (i.e. "number used once") Na. The newly generated key is encoded with the long-term key Kab, which also ensures B (together with Na) that role A is the one who is actually sending the message.

This work is part of research grant nr. 2445/10.07.2007 entitled "Contributions to security protocol composition using performance criteria" with the "Petru Maior" University of Targu Mures.

¹ Genge Bela is with the Faculty of Electrical Engineering, "Petru Maior" University of Targu Mures, Romania (e-mail: bgenge@upm.ro).

² dr. Haller Piroska is with the Faculty of Electrical Engineering, "Petru Maior" University of Targu Mures, Romania. (e-mail: phaller@upm.ro).

We can see that this message is atomic; neither of its components can be separated and sent alone encrypted with the long-term key without losing the properties intended in the design process. For example, the message loses its security property (i.e. freshness of the session key) if the components are separated, because the intruder, even if K_{ab} is not in his possession, can still separate and concatenate encrypted messages:

$$A \rightarrow B: \{Na, K'_{ab}\} Kab, \{B\} Kab$$

On the contrary to the removal or separation of terms, by adding terms to an encrypted message, the security properties are maintained and the message gains additional new properties. For example, by adding A's name to the encrypted message:

$$A \rightarrow B: \{A, Na, K'_{ab}, B\} Kab,$$

the new message is linked to role A not only through the key K_{ab} , but also through the newly included term A.

B. Formalizing terms and bindings

In this section we formalize the concept of *bindings*. Before getting into the formal specification, we need to define several basic sets and components used in the construction of *bindings*.

By analyzing security protocol messages, we realize that messages are mostly constructed from role names (i.e. participant names), nonces (i.e. numbers used once to ensure message freshness, including timestamps), keys (both session and long term keys, including public-private key pairs) and encryption functions (e.g. symmetric, asymmetric, hash).

Thus, we define the following basic sets, from which messages and our bindings are later constructed: R , denoting the set of role names (i.e. protocol participant names); N , denoting the set of nonces and K , denoting the set of cryptographic keys. The set of all subsets is denoted by: R^* for role names; N^* for nonces and K^* for cryptographic keys. To represent an empty sub-set, we use the \cdot symbol.

Every binding also contains a function and a key used to create the original encrypted message. When dealing with plain (i.e. unencrypted) messages we use the \cdot symbol for both the function name and key.

The encryption functions used to create cryptographic terms are formally defined as:

$$\begin{aligned} \text{FuncName} ::= & sk && (\text{secret key}) \\ & | pk && (\text{public key}) \\ & | pvk && (\text{private key}) \\ & | h && (\text{hash}) \\ & | \cdot && (\text{no function}) \end{aligned}$$

Using the mentioned sets, terms are defined as follows.

$$\begin{aligned} \mathcal{T} ::= & \cdot | R | N | K | (\mathcal{T}, \mathcal{T}) \\ & | \{\mathcal{T}\}_{\text{FuncName}(\mathcal{T})} \end{aligned}$$

To denote the set of all sub-sets of terms we use the \mathcal{T}^* symbol.

By simply creating bindings from terms we lose the position of each component. However, the position cannot be neglected because a simple modification may lead to a serious attack (e.g. type flaw attacks [4]-[5]-[6]). This is why a binding also has to contain the structure of the term. For this purpose we use a canonical representation based on *typed terms*, as already introduced by the authors in [17].

Typed terms denote the type of each message component and they are defined as follows:

$$\begin{aligned} \mathcal{T}_t ::= & r \quad (\text{role type}) \\ & | n \quad (\text{nonce type}) \\ & | k \quad (\text{key type}) \\ & | b \quad (\text{binding type}) \end{aligned}$$

To denote the set of all subsets of typed terms that can be constructed, we use the \mathcal{T}_t^* notation. In this case also, we use the \cdot symbol to denote an empty set.

Now, having defined all the components needed, we can provide the definition of *bindings*.

Definition 1. A binding is a tuple written as $\langle \rho, v, \kappa, \beta, f, k, \theta \rangle$, where $\rho \in R^*$, $v \in N^*$, $\kappa \in K^*$, $\beta \in B^*$, $f \in \text{FuncName}$, $k \in K$ and $\theta \in \mathcal{T}_t^*$. We use the B symbol to denote the set of all bindings and the symbol B^* to denote the set of all subsets of bindings. The \cdot symbol is used to denote an empty binding set.

1. To obtain the components of a binding b , we use the following projection functions:

$$\text{Roles}(b) = \rho, \text{Nonces}(b) = v, \text{Keys}(b) = \kappa,$$

$$\text{Bindings}(b) = \beta, \text{Func}(b) = f, \text{BindingKey}(b) = k,$$

$$\text{TypeSet}(b) = \theta$$

2. The binding composition operator $_+ _ : B \times B \rightarrow B$ composes all subsequent components using set operators;

3. The sub-binding operator \sqsubset is defined inductively as follows:

$$b \sqsubset b$$

$$b_1 \sqsubset b_2 \text{ if } \text{Roles}(b_1) \subseteq \text{Roles}(b_2) \wedge$$

$$\text{Nonces}(b_1) \subseteq \text{Nonces}(b_2) \wedge$$

$$\text{Keys}(b_1) \subseteq \text{Keys}(b_2) \wedge$$

$$\text{Bindings}(b_1) \subseteq \text{Bindings}(b_2) \wedge$$

$$\text{Func}(b_1) = \text{Func}(b_2) \wedge$$

$$\text{BindingKey}(b_1) = \text{BindingKey}(b_2)$$

An example specification of a binding is the following specification of a message from Lowe's modified "BAN concrete Andrew Secure RPC" [14] protocol:

$$\{Na, K'_{ab}, B\} Kab$$

$$\| \langle B, Na, Kab, \dots, sk, Kab, \{n, k, r\} \rangle$$

III. FORMALIZING THE PROTOCOL SPECIFICATION

In this section we provide a formal specification of security protocols combining the mentioned bindings with the strand space model proposed by Guttman et al in [15].

A. B-Strand spaces and B-Protocols

A *strand* is a sequence of send and receive events. A *strand space* is a collection of strands. Thus, protocol participants are modeled as strands and a protocol is represented as a strand space.

In the remaining of this section we provide a slightly modified (i.e. adapted) definition of the strand space model, that we call *b-strand space*. In the proposed model, the roles exchange bindings constructed from the original terms.

Strands are usually used to represent roles. However, they can also be used to represent internal operations specific to roles, such as encryption, decryption or memory storage. Unlike in the original model, we need to determine the type of every b-strand included in the specification. Because of this, before defining b-strands, we need to define the set of *classifiers* consisting of predefined symbols that can later be extended if needed:

$$C ::= C_{\mathcal{R}} \quad (\text{Role classifier}) \\ | C_{\mathcal{K}} \quad (\text{Knowledge classifier})$$

To denote the sending and receiving of terms, the original strand space model introduces *signed terms*. The appearance of a positive term denotes transmission and the appearance of a negative term denotes reception. Similarly, in the proposed model, a positive binding denotes sending and a negative binding denotes reception. A signed binding is also known as a *node*.

Next, we provide the definition for *signed bindings* and then proceed with the definition of b-strands and b-strand spaces.

Definition 1. A signed binding is a pair $\langle \sigma, b \rangle$ with $b \in \mathbf{B}$ and σ one of the symbols $+$, $-$. A signed binding is written as $-b$ or $+b$. $(\pm \mathbf{B})^*$ is the set of finite sequences of signed bindings. A typical element of $(\pm \mathbf{B})^*$ is denoted by $\langle \pm b_1, \pm b_2, \dots, \pm b_n \rangle$, with $b_i \in \mathbf{B}$.

Definition 2. A b-strand $s : (\pm \mathbf{B})^* \times C$ is a sequence of binding transmissions and receptions attached to a strand classifier. A set of b-strands is called a *b-strand space* and is represented as Σ . We use the Σ^* symbol to denote a set of b-strand space subsets.

1. A node is any transmission or reception of a binding, written as $n_i = \langle s, i \rangle$, with $s \in \Sigma$ and i an

integer satisfying the condition $1 \leq i \leq \text{length}(s)$, where $\text{length}(s)$ is a function returning the number of nodes from a b-strand. The set of all nodes is denoted by \mathcal{N} .

2. Let $n_1 = \langle s, i \rangle$ and $n_2 = \langle s, i+1 \rangle$ be two consecutive nodes from \mathcal{N} on the same b-strand s . Then, there exists an edge $n_1 \Rightarrow n_2$ in the same b-strand s .
3. Let $n_1, n_2 \in \mathcal{N}$. If n_1 is positive and n_2 is negative, and $\text{bstrand}(n_1) \neq \text{bstrand}(n_2)$, then there exists an edge $n_1 \rightarrow n_2$.
4. Let $n \in \mathcal{N}$. Then $\text{sign}(n)$ is a function returning the sign and $\text{binding}(n)$ is a function returning the binding corresponding to a given node.
5. Let $s \in \Sigma$ with $s = \langle \beta_s, c \rangle$. Then we define the following projection functions:

$$(s)_1 = \beta_s \quad (s)_2 = c$$

Definition 3. A role specification is a pair $\langle r, \xi \rangle$, such that $r \in \mathbf{R}$ and $\xi \in \Sigma^*$. Let r_s be a role specification. Then $\text{Role}(r_s)$ is a projection function returning the role name and $\text{BStrands}(r_s)$ is a projection function returning the set of b-strands corresponding to a role specification.

A set of role specifications is denoted by RoleSpec and RoleSpec^* denotes the set of all subsets of role specifications.

Simply specifying the sequence of messages for a protocol is not enough. A specification must also include initial role knowledge defined as follows.

Definition 4. Role knowledge is a pair $\langle r, b \rangle$, where $r \in \mathbf{R}$ and $b \in \mathbf{B}$, such that $\text{Func}(b) = \text{BindingKey}(b) = \dots$. We use RoleKnow to denote a set of role knowledge and RoleKnow^* to denote the set of all subsets of role knowledge.

Thus, a protocol based on bindings (also called a binding protocol, or simply b-protocol) is defined as a set of role knowledge attached to a set of role specifications: $\text{ProtSpec} = \text{RoleKnow}^* \times \text{RoleSpec}^*$.

To exemplify the specification of a protocol in the b-strand space model, we use Lowe's modified "BAN Concrete Secure RPC" protocol [14]. The regular protocol specification and the resulting b-strand model are presented in Fig. 1.

In the given example, there are two b-strands (Fig. 1(b)): one corresponding to role A (i.e. s_A) and one corresponding to role B (i.e. s_B). For the given b-strands:

$$(s_A)_2 = C_{\mathcal{R}}, (s_A)_3 = A, (s_B)_2 = C_{\mathcal{R}}, (s_B)_3 = B$$

$$\begin{aligned}
A \rightarrow B &: A, N_a \\
B \rightarrow A &: \{N_a, K, B\}_{K_{AB}} \\
A \rightarrow B &: \{N_a\}_K \\
B \rightarrow A &: N_b
\end{aligned}
\tag{a}$$

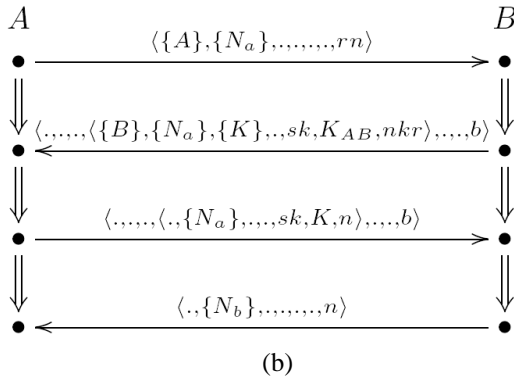


Figure 1. Lowe's protocol
(a) Regular specification (b) B-Strand space specification

B. Constructing knowledge b-strands

Although the specification model from the previous section includes initial role knowledge, this knowledge can change from one received binding to another. Because of this we also need to model the knowledge available for each node to construct the next binding.

Knowledge is modeled as a b-strand included in a role specification. Thus, a role specification consists of zero or more knowledge b-strands (i.e. b-strands having the classifier equal to $C_{\mathcal{K}}$) and one or more role b-strands (i.e. b-strands having the classifier equal to $C_{\mathcal{R}}$). For example, in Fig. 1(b) the specification includes zero knowledge b-strands and one role b-strand for each role.

The purpose of knowledge b-strands is to explicitly represent the relationship between positive nodes (i.e. sending nodes) and the knowledge extracted from received terms. We create knowledge b-strands from an existing b-protocol specification.

Knowledge is viewed as being private to each role and inaccessible by other roles. To model the concept of private knowledge, we consider that bindings exchanged between role b-strands and knowledge b-strands inside a role specification have a unique binding function "kenc" and a unique key corresponding to every role-knowledge b-strand pair. This key is included in the initial knowledge for each role.

The function name definition is thus extended with "kenc":

$$\begin{aligned}
FuncName ::= & FuncName \\
& / kenc \quad (knowledge\ encryption)
\end{aligned}$$

The construction algorithm receives as input an existing specification and generates the corresponding knowledge b-strands and role b-strands. Given an existing protocol specification, the knowledge b-strand construction

algorithm is the following:

Knowledge b-strand construction algorithm:
LET $P = \{RK, RS\}$ be a protocol specification
LET RS_{Constr} be an empty set of role specifications
FOR EACH $r_s \in RS$

$$\begin{aligned}
s_{Constr} &= \langle +getInitialB(r_s, RK) \rangle \\
s_{KConstr} &= \langle -getInitialB(r_s, RK) \rangle \\
\mathbf{FOR\ EACH\ } n \in & (getBStrand(r_s, C_{\mathcal{R}}))_1 \\
\mathbf{IF\ } sign(n) = & + \\
s_{Constr} &= \langle s_{Constr}, -getKnowB(s_{KConstr}, n) \rangle \\
s_{KConstr} &= \langle s_{KConstr}, +getKnowB(s_{KConstr}) \rangle \\
\mathbf{ELSE} \\
s_{Constr} &= \langle s_{Constr}, n, +binding(n) \rangle \\
s_{KConstr} &= \langle s_{KConstr}, -binding(n) \rangle \\
\mathbf{ENDIF} \\
\mathbf{ENDFOR} \\
RS_{Constr} &= \langle RS_{Constr}, \langle s_{Constr}, s_{KConstr} \rangle \rangle \\
\mathbf{ENDFOR}
\end{aligned}$$

In the construction process, a received binding generates an additional node transmitting a binding to the knowledge b-strand. A transmitted binding, however, generates a receiving node from the knowledge b-strand, denoting the causality between role knowledge and binding transmission.

The *getInitialB* function returns the initial knowledge binding for the given role specification. This is then used to initialize the role and knowledge b-strands, thus modeling in a natural way initial role knowledge.

The *getKnowB* function is used to return the accumulated knowledge binding from the knowledge b-strand. Because knowledge is accumulative, for every binding received by knowledge b-strands, a new knowledge binding is constructed and emitted. Also, for every received binding, the existing knowledge is checked for knowledge that can be extracted from internal bindings (e.g. the case of bindings corresponding to terms that can only be decrypted later, when the decryption key is received).

We used the *getBStrand* function to return a set of b-strands from a role specification, given a b-strand classifier.

IV. COMPOSITION ALGORITHM

In this section we provide a composition algorithm for generating new security protocols.

The composition algorithm starts with the creation of knowledge b-strands for each role specification, given the b-strand specification of the involved protocols. This is done using the proposed algorithm from the previous section.

Next, based on the created b-stands, a sequential search is started to find an “appropriate” place for each binding from one specification in the other protocol specification. Here, “appropriate” means the satisfaction of several requirements, such as existent knowledge (returned by the *accumKnow* function) and the existence of the same sequence of predecessor nodes (returned by the *predNodeSeq*).

However, not all nodes can be placed in the other protocol, because there may be cases when the requirements are not satisfied. In this case, the third step of the algorithm simply concatenates the remaining nodes unchanged to the end of the destination specification.

A more detailed description of the algorithm is the following:

Composition algorithm:

LET r_{s1}, r_{s2} be two role specifications from different b-protocols such that $Role(r_{s2}) = Role(r_{s1})$

LET $ProcNodes \in (\pm B)^*$ be a set containing the processed nodes, initially empty

1. Create knowledge b-stands

2. FOR EACH $n_2 \in (getBStrand(r_{s2}, C_{\mathcal{R}}))_1$,

$n_1 \in (getBStrand(r_{s1}, C_{\mathcal{R}}))_1$,

$sign(n_2) = +, sign(n_1) = +$

$b_1 = binding(n_1)$

$b_2 = binding(n_2)$

IF $n_2 \notin ProcNodes$ AND

$Func(b_2) \llcorner 'kenc'$ AND

$Func(binding(n_1)) \llcorner 'kenc'$ AND

$destRole(n_2) = destRole(n_1)$ AND

$accumKnow(n_2) \sqsubset accumKnow(n_1)$ AND

$predNodeSeq(n_2) \subseteq predNodeSeq(n_1)$

$b_1 = b_1 + b_2$

$accumKnow(destNode(n_1)) =$

$accumKnow(destNode(n_1)) +$

$accumKnow(n_2)$

$ProcNodes = ProcNodes \cup \{n_2\}$

ENDIF

ENDFOR

3. FOR EACH $n \in (getBStrand(r_{s2}, C_{\mathcal{R}}))_1$

IF $n \notin ProcNodes$ AND

$Func(binding(n)) \llcorner 'kenc'$ AND

$sign(n) = +$

$getBStrand(r_{s1}, C_{\mathcal{R}}) =$

$\langle getBStrand(r_{s1}, C_{\mathcal{R}}), n \rangle$

ENDIF

ENDFOR

By running the algorithm, there may result multiple possibilities for creating the new protocol. In fact, if we want to step through all possible forms of the constructed protocol, the presented algorithm should be included in a backtracking algorithm. However, the backtracking algorithm must have an evaluation function, which can include a combination of several criterias, such as binding size, number of sent/received bindings or total number of bindings for which $Func(binding(n)) \llcorner '!'.$

V. EXAMPLE COMPOSITION

In this section we provide an example composition of two different security protocols. Because of space considerations, we left out from the specifications the representation of knowledge b-stands.

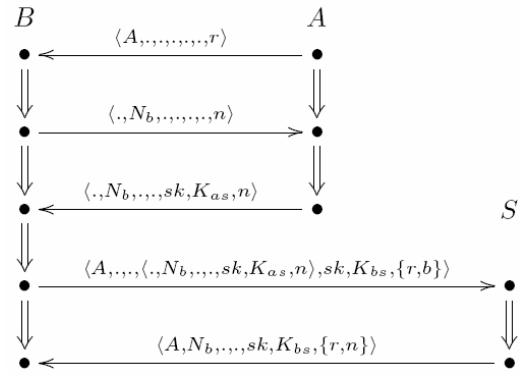


Figure 2. The “Woo and Lam Pi 3” authentication protocol representation in the b-strand space model

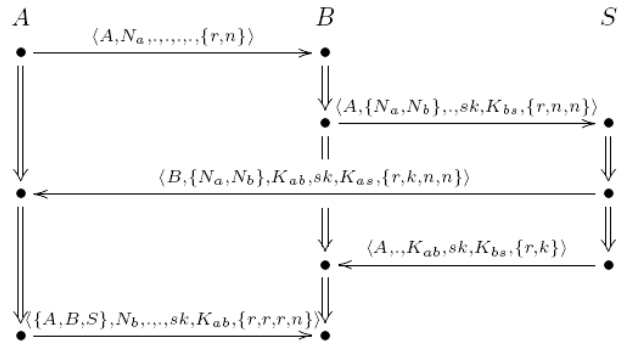


Figure 3. Lowe's modified version of the Yahalom protocol represented in the b-strand space model

The first protocol is the “Woo and Lam Pi 3” security protocol [16] (Fig. 2). This protocol was designed to provide a one-way authentication. Namely, role A proves his authenticity to role B through the trusted server S.

The second protocol is Lowe's modified version of the Yahalom protocol [18] (Fig. 3). This protocol, as mentioned by the designers, provides a mutual authentication between roles A and B and a session key distribution. The authentication of role B to role A is done using the third message from the specification sent from the server S to role A. The authentication of role B to role A is intended with messages four and five. However,

because message four is not linked to the current run (i.e. it does not contain the nonce N_b), the authentication sequence can be attacked, as proven by Paulson in [19].

The scope of the composition is to provide a mutual authentication and a key exchange protocol. By applying the composition algorithm from the previous section we can generate a new strengthened protocol, which provides mutual authentication of the involved roles and key exchange (also solving the problem reported by Paulson). The resulting protocol can be seen in Fig. 4.

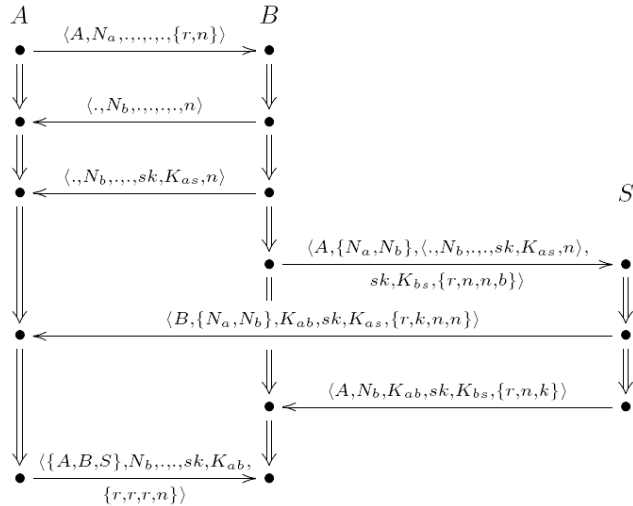


Figure 4. The composed protocol

VI. CONCLUSIONS

We have presented a method for security protocol message composition. The novelty of our method is the message-level approach, which allows the composition of existing security protocols without having to formulate additional security properties. The composition results in a strengthened protocol that embodies the sum of the security properties contained in the separate protocols.

We did not provide, however, a powerful composition algorithm to evaluate and chose the most adequate resulting security protocol. However, the model described in this paper can be used in conjunction with other, more complex algorithms, which the authors intend to include in their future research agenda.

In this paper we also proposed a performance evaluation criteria based on binding dimensions and enumeration. This can be replaced, however, with a more powerful practical model that also allows, for example, a comparison between the complexity of the encryption function and key.

Also, because the composition process can become rather complex even when dealing with two protocols, we intend to implement in the future an automated

composition tool.

REFERENCES

- [1] M. Abadi, A. D. Gordon, "A Calculus for Cryptographic Protocols: the spi-calculus", In Fourth ACM Conference on Computer and Communications Security, ACM Press, pp. 36-47, 1997.
- [2] Andrew D. Gordon, Alan Jeffrey, "Authenticity by Typing for Security Protocols", Journal of Computer Security, 11(4), pp. 451-520, 2003.
- [3] Cremers C., Scyther documentation, 2004, available at <http://www.win.tue.nl/~cremers/scyther>.
- [4] Catherine Meadows, "A Procedure for Verifying Security Against Type Confusion Attacks", 16th IEEE Computer Security Foundations Workshop (CSFW'03), p. 62, 2003.
- [5] Genge Bela, Iosif Ignat, "An Abstract Model for Security Protocol Analysis", WSEAS TRANSACTIONS on COMPUTERS, Issue 2, Volume 6, pp. 207-215, 2007.
- [6] Genge Bela, Iosif Ignat, "A typed specification for security protocols", Proceedings of the 5th WSEAS Int. Conf. on Data Networks, Communications and Computers, Bucharest, Romania, October 16-17, pp. 113-118, 2006.
- [7] Cas J. F. Cremers, "Compositionality of Security Protocols: A Research Agenda", Electr. Notes Theor. Comput. Sci., 142, pp. 99-110, 2006.
- [8] S. Andova, Cas J.F. Cremers, K. Gjøsteen, S. Mauw, S. Mjølnes, and S. Radomirovic, "A framework for compositional verification of security protocols", to appear, 2007.
- [9] Levente Buttyan, "Building blocks for secure services: Authenticated key transport and Rational exchange protocols", Thesis, 2001.
- [10] Joshua D. Guttman, "Security protocol design via authentication tests", In Proceedings of the 15th IEEE Computer Security Foundations Workshop, IEEE CS Press, June, 2002.
- [11] Hyun-Jin Choi, "Security protocol design by composition", Cambridge University, UK, Technical report Nr. 657, UCAM-CL-TR-657, ISSN 1476-2986, 2006.
- [12] Ran Canetti, Tal Rabin, "Universal Composition with Joint State", In Proceedings of CRYPTO 2003, Lecture Notes in Computer Science, vol. 2729. Springer Verlag, New York, pp. 265—281, 2003.
- [13] A. Datta, A. Derek, J. C. Mitchell, A. Roy, "Protocol Composition Logic (PCL)", Electronic Notes in Theoretical Computer Science (Gordon D. Plotkin Festschrift), to appear, 2007.
- [14] Gavin Lowe, Some new attacks upon security protocols, In Proceedings of the 9th Computer Security Foundations Workshop, IEEE Computer Society Press, pp. 162-169, 1996.
- [15] F. Javier Thayer Fabrega, Jonathan C. Herzog, Joshua D. Guttman, "Strand spaces: Proving security protocols correct", Journal of Computer Security 7, 191-230, 1999.
- [16] T.Y.C. Woo and S. S. Lam, "A lesson on authentication protocol design", Operating Systems Review, 1994.
- [17] Genge Bela, Iosif Ignat, "Verifying the Independence of Security Protocols", IEEE 3rd International Conference on Intelligent Computer Communication and Processing, Cluj-Napoca, Romania, pp.155-163, 2007.
- [18] Gavin Lowe, "Towards a completeness result for model checking of security protocols", Technical Report 1998/6, Dept. of Mathematics and Computer Science, University of Leicester, 1998.
- [19] Lawrence J. Paulson, "Relations between secrets: Two formal analyses of the Yahalom protocol", Journal of Computer Science, 2001.