STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIV**, Number 2, 2009

TOWARDS AUTOMATED EXECUTION OF SECURITY PROTOCOLS FOR WEB SERVICES

BÉLA GENGE

Abstract. Existing solutions for authentication and authorization in Web services make use of technologies such as SAML or WS-Security. These provide a static solution by using a set of predefined protocols. We propose a dynamic approach for the automated execution problem by developing a semantic security protocol model from which security protocol specifications are generated and automatically executed by participants. The proposed model consists of a sequential component, implemented as a WSDL-S specification, and an ontology component, implemented as an OWL specification. The correctness of the proposed model is ensured by using a set of rules and algorithms for generating it based on a protocol model given by the user. We validate our approach by generating and implementing several specifications for existing protocols such as the ISO9798 or Kerberos protocol.

1. Introduction

Security protocols are widely used today to provide secure communication in insecure environments. By examining the literature we come upon various security protocols designed to provide solutions to specific problems [1]. With this large amount of protocols to chose from, distributed heterogenous systems must be prepared to handle multiple security protocols.

Existing technologies, such as the Security Assertions Markup Language [2] (i.e. SAML) or WS-Security [3] provide a unifying solution for the authentication and authorization issues through the use of predefined protocols. By implementing these protocols, Web services authenticate users and provide authorized access to resources. However, despite the fact that existing solutions provide a way to implement security claims, these approaches are rather

Received by the editors: November 17, 2008.

2010 *Mathematics Subject Classification.* 68M14, 68Q55, 94A62.

1998 *CR Categories and Descriptors.* H.3.5 [**Information Systems**]: Information Storage and Retrieval – *Online Information Services* K.6.5 [**Computing Milieux**]: Management of Computing and Information Systems – *Security and Protection*.

Key words and phrases. Security protocols, Automated protocol execution, Web services.

static. This means that in case of new security protocols, services must be reprogrammed.

The solutions developed over the years such as the one described in [4], propose a formal description for protocol specifications. These specifications do not make use of Web service technologies, because of which inter-operability of systems executing the given specifications becomes a real issue. Another approach for automated security protocol implementation is provided in [5], where the specification is constructed as an XML document from which the code is automatically generated. In order to provide automated execution solutions, specifications must not generate code, but must provide an automated implementation solution without any user intervention.

In this paper we propose a semantic security protocol model (SSPM) for generating security protocol specifications that can be automatically executed by participants. The SSPM has two components: a sequential model and an ontology model. The first component is implemented as a WSDL-S [6] specification while the second component is implemented as an OWL [7] specification. The role of the WSDL-S implementation is to describe the message sequences and directions that must be executed by protocol participants. The role of the OWL implementation is to provide semantic information such as the construction, processing and implementation of cryptographic operations (e.g. encryption algorithm, encryption mode, key).

The proposed SSPM is constructed from a security protocol model (SPM) provided by the user. This model describes message sequences, protocol preconditions and effects. Protocol preconditions are used to identify the knowledge required for running the protocol while protocol effects identify the goal of the protocol (e.g. authentication, key exchange).

The construction of the SSPM from a given SPM must maintain the protocol's security properties. For this we propose several generating rules and algorithms that provide a mapping for each component from SPM to SSPM. The correctness of the proposed rules and algorithms results from the one-to-one mapping of each component and from the correctness of SPM.

In order to validate the proposed solution we have generated and implemented several security protocol specifications. From our results we can clearly state that the SSPM contains sufficient information to enable participants to execute the generated specifications.

2. Protocol Model

Protocol participants communicate by exchanging *terms* constructed from elements belonging to the following basic sets: P, denoting the set of role names; N, denoting the set of random numbers or *nonces* (i.e. "number once

used"); K, denoting the set of cryptographic keys; C, denoting the set of certificates and M, denoting the set of user-defined message components.

In order for the protocol model to capture the message component types found in security protocol implementations [2, 3] we specialize the basic sets with the following subsets:

- $\mathsf{P}_{DN} \subseteq \mathsf{P}$, denoting the set of distinguished names; $\mathsf{P}_{UD} \subseteq \mathsf{P}$, denoting the set of user-domain names; $\mathsf{P}_{IP} \subseteq \mathsf{P}$, denoting the set of user-ip names; $\mathsf{P}_U = \{\mathsf{P} \setminus \{\mathsf{P}_{DN} \cup \mathsf{P}_{UD} \cup \mathsf{P}_{IP}\}\}$, denoting the set of names that do not belong to the previous subsets;
- N_T, denoting the set of timestamps; N_{DH}, denoting the set of random numbers specific to the Diffie-Hellman key exchange; $\mathsf{N}_A = \{\mathsf{N} \setminus \{\mathsf{N}_{DH} \cup \mathsf{N}_T\}\}$, denoting the set of random numbers;
- $\mathsf{K}_S \subseteq \mathsf{K}$, denoting the set of symmetric keys; $\mathsf{K}_{DH} \subseteq \mathsf{K}$, denoting the set of keys generated from a Diffie-Hellman key exchange; $\mathsf{K}_{PUB} \subseteq \mathsf{K}$, denoting the set of public keys; $\mathsf{K}_{PRV} \subseteq \mathsf{K}$, denoting the set of private keys;

To denote the encryption type used to create cryptographic terms, we define the following *function names*:

$$\begin{aligned} FuncName ::= {} & sk && (symmetric\, function) \\ | {} & pk && (asymmetric\, function) \\ | {} & h && (hash\, function) \\ | {} & hmac && (keyed\, hash\, function) \end{aligned}$$

The above-defined basic sets and function names are used in the definition of *terms*, where we also introduce constructors for pairing and encryption:

$$\mathsf{T} ::= . \mid \mathsf{R} \mid \mathsf{N} \mid \mathsf{K} \mid \mathsf{C} \mid \mathsf{M} \mid (\mathsf{T}, \mathsf{T}) \mid \{\mathsf{T}\}_{FuncName(\mathsf{T})},$$

where the '.' symbol is used to denote an empty term.

Having defined the terms exchanged by participants, we can proceed with the definition of a *node* and a *participant chain*. To capture the sending and receiving of terms, the definition of nodes uses *signed terms*. The occurrence of a term with a positive sign denotes transmission, while the occurrence of a term with a negative sign denotes reception.

Definition 1. *A* node *is any transmission or reception of a term denoted as* $\langle \sigma, t \rangle$*, with* $t \in \mathsf{T}$ *and* σ *one of the symbols* $+, -$*. A node is written as* $-t$ *or* $+t$*. We use* $(\pm\mathsf{T})$ *to denote a set of nodes. Let* $n \in (\pm\mathsf{T})$*, then we define the function* $sign(n)$ *to map the sign and the function* $term(n)$ *to map the term corresponding to a given node.*

Definition 2. *A* participant chain *is a sequence of nodes. We use* $(\pm\mathsf{T})^*$ *to denote the set of finite sequences of nodes and* $\langle \pm t_1, \pm t_2, \ldots, \pm t_i \rangle$ *to denote an element of* $(\pm\mathsf{T})^*$.

In order to define a participant model we also need to define the preconditions that must be met such that a participant is able to execute a given protocol. In addition, we also need to define the effects resulting from a participant executing a protocol.

Preconditions and effects are defined using predicates applied on terms: $CON\_TERM : \mathsf{T}$, denoting a term that must be previously generated (preconditions) or it is generated (effects); $CON\_PARTAUTH : \mathsf{T}$, denoting a participant that must be previously authenticated (preconditions) or a participant that is authenticated (effects); $CON\_CONF : \mathsf{T}$, denoting that a given term must be confidential (preconditions) or it is kept confidential (effects); $CON\_INTEG : \mathsf{T}$, denoting that for a given term the integrity property must be provided (preconditions) or that the protocol ensures the integrity property for the given term (effects); $CON\_NONREP : \mathsf{T}$, denoting that for a given term the non-repudiation property must be provided (preconditions) or that the protocol ensures the non-repudiation property for the given term (effects); $CON\_KEYEX : \mathsf{T}$, denoting that a key exchange protocol must be executed before (preconditions) or that this protocol provides a key exchange resulting the given term (effects).

The set of precondition-effect predicates is denoted by $\mathsf{PR\_CC}$ and the set of precondition-effect predicate subsets is denoted by $\mathsf{PR\_CC}^*$. Next, we define predicates for each type of term exchanged by protocol participants. These predicates are based on the basic and specialized sets provided at the beginning of this section. We use the $TYPE\_DN : \mathsf{T}$ predicate to denote distinguished name terms, $TYPE\_UD : \mathsf{T}$ to denote user-domain name terms, $TYPE\_IP : \mathsf{T}$ to denote user-ip name terms, $TYPE\_U : \mathsf{T}$ user name terms, $TYPE\_NT : \mathsf{T}$ to denote timestamp terms, $TYPE\_NDH : \mathsf{T}$ to denote Diffie-Hellman random number terms, $TYPE\_NA : \mathsf{T}$ to denote other random number terms, $TYPE\_NDH : \mathsf{T} \times \mathsf{T} \times \mathsf{T} \times \mathsf{P} \times \mathsf{P}$ to denote Diffie-Hellman symmetric key terms $(term, number_1, number_2, participant_1, participant_2)$, $TYPE\_KSYM : \mathsf{T} \times \mathsf{P} \times \mathsf{P}$ to denote symmetric key terms $(term, participant_1, participant_2)$, $TYPE\_KPUB : \mathsf{T} \times \mathsf{P}$ to denote public key terms $(term, participant)$, $TYPE\_KPRV : \mathsf{T} \times \mathsf{P}$ to denote private key terms $(term, participant)$, $TYPE\_CERT : \mathsf{T} \times \mathsf{P}$ do denote certificate terms $(term, participant)$ and $TYPE\_MSG : \mathsf{T}$ to denote user-defined terms.

The set of type predicates is denoted by $\mathsf{PR\_TYPE}$ and the set of type predicate subsets is denoted by $\mathsf{PR\_TYPE}^*$. Based on the defined sets and predicates we are now ready to define the participant and protocol models.

Definition 3. *A* participant model *is a tuple* $\langle prec, eff, type, gen, part, chain\rangle$, *where* $prec \in \mathsf{PR\_CC}^*$ *is a set of precondition predicates,* $eff \in \mathsf{PR\_CC}^*$ *is a set of effect predicates,* $type \in \mathsf{PR\_TYPE}$ *is a set of type predicates,* $gen \in \mathsf{T}^*$ *is a set of generated terms,* $part \in \mathsf{P}$ *is a participant name and* $chain \in (\pm\mathsf{T})^*$ *is a participant chain. We use the* MPART *symbol to denote the set of all participant models.*

Definition 4. *A* security protocol model *is a collection of participant models such that for each positive node* n_1 *there is exactly one negative node* n_2 *with* $term(n_1) = term(n_2)$. *We use the* MPROT *symbol to denote the set of all security protocol models.*

3. Semantic Security Protocol Model

In this section we described the proposed semantic security protocol model (SSPM). The proposed model must maintain the security properties of the protocol and must provide sufficient information for participants to be able to execute the protocol.

Protocols are given using their SPM model described in the previous section. Based on this model we must generate the corresponding SSPM from which the specifications can be constructed. The SSPM has two components: the sequential model (SEQM) and the ontology model (ONTM). The first component is implemented as a WSDL-S specification while the second component is implemented as an OWL specification. In the remaining of this section we provide a description of each component and we provide a set of rules to generate SSPM from a given SPM.

3.1. Sequential and Ontology Models.

We use the symbol URI to denote the set of *Uniform Resource Identifiers*, CONC to denote the set of all concepts and CONC^* to denote the set of subsets with elements from CONC.

Definition 5. *An* annotation *is a pair* $\langle uri, c\rangle$, *where* $uri \in \mathsf{URI}$ *and* $c \in \mathsf{CONC}$. *The set corresponding to a* SSPM *is denoted by* ANNOT *and the set of subsets with elements from* ANNOT *is denoted by* ANNOT^*. *A* message *is a pair* $\langle d, a\rangle$, *where* $d \in \{in, out\}$ *and* $a \in \mathsf{ANNOT}$. *We define* MSG *to denote a set of messages and* MSG^* *to denote the set of subsets with elements from* MSG.

Next, we define the sequential model as a collection of preconditions, effects and messages, based on the previous definitions.

Definition 6. *A* sequential model *is a triplet* $\langle s\_prec, s\_eff, s\_msg\rangle$, *where* $s\_prec \in \mathsf{ANNOT}^*$ *is a set of preconditions,* $s\_eff \in \mathsf{ANNOT}^*$ *is a set of effects and* $s\_msg \in \mathsf{MSG}^*$ *is a set of messages.*

The ontology model follows the description of OWL.

Definition 7. *An* ontology model *is a triplet* $\langle conc, propr, inst \rangle$*, where* $conc \in \mathsf{CONC}$ *is a set of concepts,* $propr \in \mathsf{PROPR}$ *is a set of properties and* $inst \in \mathsf{INST}$ *is a set of instances. An element from propr is a pair* $\langle \alpha, \beta \rangle$*, where* α *is a unique id and* β *is a syntactic construction denoting the property name.*

Let $pr_1 = \langle \alpha_1, \beta_1 \rangle$ *and* $pr_2 = \langle \alpha_2, \beta_2 \rangle$*. Then* $pr_1 = pr_2$ *iff* $\alpha_1 = \alpha_2$ *and* $\beta_1 = \beta_2$*. We define the function* $(\_)_{\mathrm{id}}$ *to map the* α *component and the function* $(\_)_{\mathrm{nm}}$ *to map the* β *component of a given property.*

We use PROPR *to denote the set of all properties and* INST *to denote the set of all instances. We use* PROPR^* *to denote the set of all subsets with elements from* PROPR *and* INST^* *to denote the set of all subsets with elements from* INST*.*

In order to handle the previously defined ontology model we define the function $(\_)_{\mathrm{d}} : \mathsf{PROPR} \rightarrow \mathsf{CONC}$ to map the domain concept of a given property, $(\_)_{\mathrm{c}} : \mathsf{PROPR} \rightarrow \mathsf{CONC}$ to map the category concept of a given property, $(\_,\_)_{\mathrm{ci}} : \mathsf{CONC} \times \mathsf{PROPR} \rightarrow \mathsf{INST}$ to map the instance corresponding to a domain concept and property, $(\_)^{\mathrm{s}}_{\mathrm{e}} : \mathsf{CONC} \rightarrow \mathsf{CONC}^*$ to map the set of concepts for which the given concept is parent, $(\_)_{\mathrm{p}} : \mathsf{CONC} \rightarrow \mathsf{PROPR}^*$ to map the set of properties for which the given concept is domain.

3.2. **Generating the Semantic Security Protocol Model.** In order to generate the SSPM of a given SPM, we start with a core ontology model (OM) (figure 1) that contains concepts found in classical security protocols. The core OM was constructed by consulting security protocols found in open libraries such as SPORE [1] or the library published by John Clark [8].

The core ontology is constructed from 7 sub-ontologies. The sub-ontologies that must be extended with new concepts for each SSPM are denoted in figure 1 by interrupted lines, while the permanent sub-ontologies are denoted by continuous lines.

The *SecurityProperty* sub-ontology contains concepts such as *Authentication*, *Integrity*, *Confidentiality*, *Session_key_exchange*. The *TermType* sub-ontology includes concepts related to term types used in security protocol messages such as *SymmetricKey*, *PublicKey* or *ParticipantName*. Concepts related to cryptographic specifications such as encryption algorithms or encryption modes are found in the sub-ontology *CryptoSpec*. In order to model modules needed to extract keys, names or certificates we use the *LoadingModule* sub-ontology. The *ParticipantRole* sub-ontology defines concepts modeling roles handled by protocol participants such as *Initiator*, *Respondent* and *Third_Party*.

The *Knowledge* sub-ontology contains 5 concepts: *PreviousTerm*, *AccessedModule*, *InitialTerm*, *GeneratedTerm* and *DiscoveredTerm*. Each concept defines a class of terms specific to security protocols: terms from previous executions, modules, initial terms, generated terms and discovered terms.

The last sub-ontology is *CommunicationTerm*, which defines two concepts: *SentTerm* and *ReceivedTerm*. The ontology is extended for each SEM-S with concepts that are sent or received. For each concept, functional properties are defined denoting the operations performed on the terms corresponding to concepts. The concepts used to extend the core ontology are specific to each

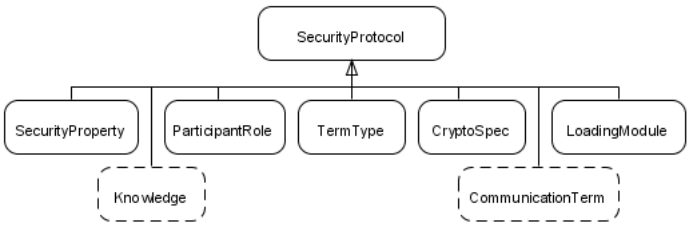

FIGURE 1. Core ontology of SSPM

protocol, however, the defined properties are applied on all constructions. From these properties we mention: *isOfType*, *isEncrypted*, *isStored*, *isVerified*, *isExtracted*, *hasSymmetricAlgorithm*, *hasKey*, *hasLength*.

In order to generate the SSPM from a given SPM we define a set of rules and generating algorithms. The developed rules use the $\_ \leftarrow_{\mathrm{r}} \_$ operator to denote set reunion and the $\_ \leftarrow_{\mathrm{a}} \_$ operator to denote a value transfer.

The first two rules generate the predicate concepts corresponding to preconditions *prec* from a SPM, where the function $gc : \mathsf{T} \rightarrow \mathsf{CONC}$ is used to generate the concept corresponding to a given term and the function $gcc : \mathsf{PR\_CC} \rightarrow \mathsf{CONC}$ is used to generate the concept corresponding to a given precondition predicate:

$$\frac{pr \in prec \;\; pr = CON\_TERM(t)}{c \leftarrow_{\mathrm{a}} gc(t) \;\; s\_prec \leftarrow_{\mathrm{r}} \{\langle uri, c\rangle\} \;\; (InitialTerm)^{\mathrm{s}}_{\mathrm{e}} \leftarrow_{\mathrm{r}} \{c\}}\text{pr\_term},$$

$$\frac{pr \in prec \;\; pr \neq CON\_TERM(t)}{s\_prec \leftarrow_{\mathrm{r}} \{\langle uri, gcc(pr)\rangle, \langle uri, gc(t)\rangle\}}\text{pr\_propr}.$$

The rules generating the effects have a similar structure because of the *eff* set. Concatenated terms corresponding to each transmitted or received term are modeled using similar rules. For each sent term the SSPM must provide the construction operations and for each received term the SSPM must provide processing operations.

Processing the received terms is done according to the type of each term and to the knowledge available to the user. The modeled operations introduce constraints on the type and location of knowledge through the following rules.

In the *Knowledge* sub-ontology, each concept has an *isOfType* property attached based on which participants can decide on the operations to execute. For each type, additional properties are defined such as the *hasSymmAlg* or *hasKey* properties for symmetric encrypted terms. The rules based on which these properties are generated are specific to each type.

The remaining generating rules are similar to the presented ones and a complete presentation is out of the scope of this paper. We now provide a brief description of the algorithms that apply the rules we have defined.

The first algorithm generates the preconditions, effects and message sequences of SSPM.

Algorithm 1 Generate preconditions, effects and message sequences

Require: $\langle prec, eff, type, gen, part, chain \rangle \in \mathsf{MPART}$
  for all $pr \in prec$ **do**
    @pr_term(pr), @pr_propr(pr)
  end for
  for all $ef \in eff$ **do**
    @eff_term(pr), @eff_propr(pr)
  end for
  for all $n \in chain$ **do**
    if $sign(n) = +$ **then**
      @msg_tx(n)
    else
      @msg_rx(n)
    end if
  end for

Generating concepts corresponding to the *Knowledge* sub-ontology is done through the use of algorithm 2 and 3. Here, the set of knowledge *KNOW*, corresponding to each executing participant, grows with the construction and reception of each new term. We used the function $mpart : \mathsf{T} \rightarrow \mathsf{T}^*$ to map the set of concatenated terms and the keyword "Exec" to denote the execution of sub-algorithms.

3.3. **Correctness of SSPM.** In the generation process of SSPM from a given SPM, we consider a correct SPM constructed by the user. With the large number of attacks reported in the literature [9], [10], it is vital for new protocol

Algorithm 2 Model positive nodes

Require: $n \in (\pm\mathsf{T}), sign(n) = +$
  for all $t \in mpart(term(n))$ **do**
    Let $c = gc(t)$
    Let $p \Leftarrow$ @con_extr(c)
    if $t \in KNOW$ **then**
      $(p)_{\mathrm{c}} \leftarrow_{\mathrm{a}} c$
    else if $t = \{t'\}_{f(k)}$ **then**
      $(GeneratedTerm)^{\mathrm{s}}_{\mathrm{e}} \leftarrow_{\mathrm{r}} \{c\}$
      Exec $ModelEncryptedGenerated(t)$
    else if $t \in gen$ **then**
      $(GeneratedTerm)^{\mathrm{s}}_{\mathrm{e}} \leftarrow_{\mathrm{r}} \{c\}$
      Exec $ModelPlainGenerated(t)$
    else
      $(DiscoveredTerm)^{\mathrm{s}}_{\mathrm{e}} \leftarrow_{\mathrm{r}} \{c\}$
      Exec $ModelDiscoveredLoaded(t)$
    end if
    $KNOW \leftarrow_{\mathrm{r}} t$
  end for

Algorithm 3 Model negative nodes

Require: $n \in (\pm\mathsf{T}), sign(n) = -$
  for all $t \in mpart(term(n))$ **do**
    if $t \in KNOW$ **then**
      @con_verif
    else if $t = \{t'\}_{f(k)}$ **then**
      if $f = sk \vee (f = pk \wedge TYPE\_KPUB(k, r) \in type, r \in \mathsf{P})$ **then**
        @con_decr
        Exec $ModelEncryptedDiscovered(t)$
      else
        @con_verif
        Exec $ModelEncryptedGenerated(t)$
      end if
    else
      @con_stored
      Exec $ModelDiscovered(t)$
    end if
    $KNOW \leftarrow_{\mathrm{r}} t$
  end for

models to maintain the security properties of protocols for which security properties have been proved to hold.

In order to prove the correctness of the generated SSPM, we consider Γ representing the set of all information included in an SSPM. The information generated by the proposed rules can be divided into three components: mapped information, user-provided information and participant knowledge-based information.

The set of mapped information is denoted by γ_{map} and represents information originating directly from SPM. The set of user-provided information is denoted by γ_{up} and represents information originating from the user (e.g. cryptographic algorithms). The set of knowledge-based information originates from the knowledge available when running the protocol and is denoted by γ_{know}.

By using the above sets $\Gamma = \gamma_{map} \cup \gamma_{up} \cup \gamma_{know}$. The correctness of the information contained in γ_{map} results from the original protocol model, while the correctness of the information contained in γ_{up} results from the assumption that the user provides correct input data.

The information contained in γ_{know} is generated based on the design principles of *fail-stop* [14] protocols. These principles state that the correctness of each received term must be verified and the protocol execution must be stopped immediately in case of invalid terms. By using these principles, the rules we proposed generate verification properties for each received term found in the participant's knowledge set. Protocols that do not follow these rules can not be modeled with our method.

The the correctness of the generated SSPM follows from the correctness of the information generated in the Γ set, constructed from the three sets $\gamma_{map}, \gamma_{up}, \gamma_{know}$ for which the correctness has been discussed above.

4. Experimental Results

In this section we exemplify the construction of a SSPM from a given SPM and we provide a few experimental results from the collection of semantic security protocol models we have implemented.

4.1. **Constructing the SSPM for the "BAN" protocol.**

In order to provide an example for constructing an SSPM for a given SPM, we use the well-known "BAN Concrete Secure Andrew RPC" protocol [1]. This is a two-party protocol providing a session key exchange using symmetric cryptography. The protocol assumes that participants are already in the possession of a long-term key K_{ab}.

In the remaining of this sub-section we provide only the construction of the SSPM for participant A. The SSPM corresponding to participant B can be

similarly constructed because it defines the inverse operations of participant A.

The precondition set $prec_A$ for participant A is $prec_A = \{CON\_TERM(A), CON\_TERM(B), CON\_TERM(K_{ab})\}$ and the effect set eff_A for the same participant is $eff_A = \{CON\_KEYEX(K_{ab})\}$. The set $type_A = \{TYPE\_UD\ (A), TYPE\_UD(B), TYPE\_KSYM(A, B, K_{ab}), TYPE\_KSYM(A, B, K), TYPE\_NA\ (N_a), TYPE\_NA(N_b)\}$ defines the type corresponding to each term and the set $gen_A = \{N_a\}$ defines the terms generated by participant A. The participant name is $part_A = A$ and the participant chain is $chain_A = \langle +(A, N_a), -\{N_a, K, B\}_{sk(K_{ab})}, +\{N_a\}_{sk(K)}, -N_b \rangle$.

By applying the rules and algorithms described in the previous sections we generate the SSPM model. The sequential model is implemented as a WSDL-S specification, while the ontology model is implemented as a OWL specification.

Part of the resulted WSDL-S specification is given in figure 2 and part of the graphical representation of the OWL specification is given in figure 3.

```
...
<xsd:element name="Msg1Request">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="Term1" type="xsd:base64Binary"
         wssem:modelReference=".../SecProt.owl#SentTerm1">
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Msg2Response">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="EncTerm1" type="xsd:base64Binary"
         wssem:modelReference=".../SecProt.owl#RecvdEncTerm1">
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
...
<wsdl:operation name="Msg1">
 <wsdl:output message="tns:Msg1Request"/>
</wsdl:operation>
<wsdl:operation name="Msg2">
 <wsdl:input message="tns:Msg2Response"/>
</wsdl:operation>
<wssem:effect name="SessionKeyExchange"
      wssem:modelReference=".../SecProt.owl#SessionKey"/>
...
```

FIGURE 2. Sequential model partial implementation

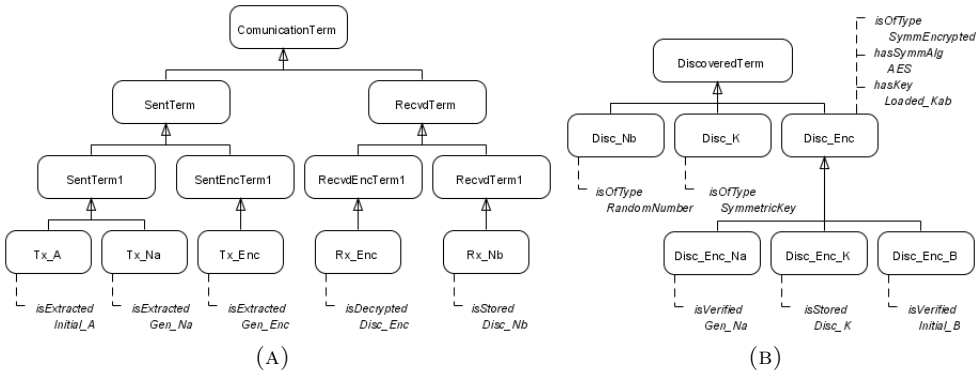

FIGURE 3. Ontology model partial implementation: (a) Communication terms sub-ontology (b) Discovered terms sub-ontology

4.2. **Experimental Results.** In order to prove that the SSPM model contains sufficient information for participants to execute the generated implementations, we generated over 38 WSDL-S and 38 OWL specifications corresponding to initiator and respondent protocol roles.

In order to execute the specifications, messages were encoded and transmitted according to the constructions provided by the WS-Security standard [3]. In the experiments we conducted, participants downloaded the specification files from a public server and they were able to execute the protocols based only on the received descriptions. The participants hardware and software configurations: Intel Dual Core CPU at 1.8GHz, 1GByte of RAM, MS Windows XP.

Part of the experimental results are given in table 1, where the values correspond to milliseconds. The "Spec. proc" column denotes the specification processing time, the "Msg. constr." column denotes the message construction time (for output messages) and the "Msg. proc." column denotes the message processing time (for input messages). The table contains two two-party protocols ("BAN Concrete Andrew Secure RPC", or more simply BAN, and ISO9798) and one three-party protocol (Kerberos). The performance differences between the BAN and ISO9798 protocols are due to the fact that ISO9798 makes use of public key cryptography, while BAN uses only symmetric cryptography.

TABLE 1. Protocol execution timings

Protocol participant	**Spec. proc. (ms)**	**Msg. constr. (ms)**	**Msg. proc. (ms)**	**Total (ms)**
BAN Init.	14.58	11.81	3.68	30.08
BAN Resp.	14.03	2.86	1.62	18.52
ISO9798 Init.	13.07	35.784	23.30	72.16
ISO9798 Resp.	13.51	6.876	12.24	32.63
Kerb. Init. 1	22.63	0.83	0	23.47
Kerb. Init. 2	12.61	0.55	1.58	14.76
Kerb. Init. 3	2.23	3.34	0.94	6.52
Kerb. Resp. 1	19.28	0	0.41	19.69
Kerb. Resp. 2	10.81	3.379	1.67	15.87
Kerb. Resp. 3	5.25	11.41	3.59	20.26

5. Conclusion and Future Work

We developed a novel method for the automated execution of security protocols. Our approach is based on a semantic security protocol model from which security protocol specifications are generated. The sequential component of the proposed model is implemented as a WSDL-S specification while the ontology component is implemented as an OWL specification.

Constructing the SSPM model is not a trivial task and can induce new flaws in correct protocols that can lead to attacks. In order to ensure a correct construction process, we developed several generating rules and algorithms. The proposed rules and algorithms map each component from the input protocol model to a component in the SSPM model. The components from SPM are extended with implementation-specific elements, that do not affect the security properties of the original protocol, as long as correct methods are used to execute the resulted specifications.

In order to prove that the proposed model contains sufficient information for automated execution, we generated and implemented several security protocol specifications. The generated specifications were constructed for well-known security protocols such as the ISO9798 protocol, CCITTX509 or the Kerberos protocol.

As future work we intend to develop a service-based middleware to support secure distribution of these specifications. The middleware will also be able to create new protocols based on already existing protocols and distribute the new specifications to Web services.

References

[1] *Security Protocol Open Repository.* Laboratoire Specification et Verification, http://www.lsv.ens-cachan.fr/spore/, 2008.
[2] *SAML V2.0 OASIS Standard Specification.* Organization for the Advancement of Structured Information Standards, http://saml.xml.org/, 2007.
[3] *OASIS Web Services Security (WSS).* Organization for the Advancement of Structured Information Standards, http://saml.xml.org/, 2006.
[4] L. Mengual, N. Barcia, E. Jimenez, E. Menasalvas, J. Setien, and J. Yaguez. Automatic implementation system of security protocols based on formal description techniques. *Proceedings of the Seventh International Symposium on Computers and Communications*, pages 355–401, 2002.
[5] I. Abdullah and D. Menasc. Protocol specification and automatic implementation using XML and CBSE. *IASTED conference on Communications, Internet and Information Technology*, November 2003.
[6] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma. *Web Service Semantics - WSDL-S.* A joint UGA-IBM Technical Note, 2005.
[7] W. W. W. Consortium. *OWL Web Ontology Language Reference.* W3C Recommendation, 2004.
[8] J. Clark, J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0. York University, 17 November 1997.
[9] Gavin Lowe. Some new attacks upon security protocols. In Proceedings of the 9th Computer Security Foundations Workshop, IEEE Computer Society Press, 1996, pp. 162–169.
[10] C. J. F. Cremers. Compositionality of Security Protocols: A Research Agenda. Electr. Notes Theor. Comput. Sci., 142, pp. 99–110, 2006.
[11] C.J.F. Cremers, S. Mauw, E.P. de Vink. Injective Synchronization: an extension of the authentication hierarchy. TCS 6186, Special issue on ARSPA'05, Editors: P. Degano and L. Vigano, 2006, Elsevier.
[12] P. Gutmann. Cryptlib Encryption Toolkit. http://www.cs.auckland.ac.nz/-pgut001/cryptlib/index.html, 2008.
[13] OpenSSL Project. version 0.9.8h, http://www.openssl.org/, 2008.
[14] Gong, L.: Fail-Stop Protocols: An Approach to Designing Secure Protocols. In Proceedings of the 5th IFIP Conference on Dependable Computing and Fault-Tolerant Systems, pp. 44–55 (1995).

"Petru Maior" University of Târgu Mureş, Electrical Engineering Department, Nicolae Iorga str., No. 1, 540088, Târgu Mureş, jud. Mureş, ROMANIA
E-mail address: bgenge@engineering.upm.ro