Middleware for Automated Implementation of Security Protocols

Béla Genge and Piroska Haller

"Petru Maior" University of Târgu Mureş, Department of Electrical Engineering,
N. Iorga Str., No. 1, (540088) Târgu Mureş, Romania
{bgenge,phaller}@engineering.upm.ro
http://www.upm.ro

Abstract. We propose a middleware for automated implementation of security protocols for Web services. The proposed middleware consists of two main layers: the communication layer and the service layer. The communication layer is built on the SOAP layer and ensures the implementation of security and service protocols. The service layer provides the discovery of services and the authorization of client applications. In order to provide automated access to the platform services we propose a novel specification of security protocols, consisting of a sequential component, implemented as a WSDL-S specification, and an ontology component, implemented as an OWL specification. Specifications are generated using a set of rules, where information related to the implementation of properties such as cryptographic algorithms or key sizes, are provided by the user. The applicability of the proposed middleware is validated by implementing a video surveillance system.

Keywords: Middleware, Web services, security protocols, automated execution, ontologies.

1 Introduction

In order to ensure security properties such as confidentiality, integrity or availability, Web services use technologies such as the Security Assertions Markup Language [19] (i.e. SAML) or WS-Security [20], providing a unifying solution for the authentication and authorization issues. The security tokens defined by WS-Security have been extended with additional ones and a set of new primitives in WS-Trust [21] allowing inter-domain authentication and authorization. The primitives defined by WS-Trust correspond to security protocols, denoting "communication protocols dedicated to achieving security goals" (C.J.F. Cremers and S. Mauw) [1]. The security protocols defined by WS-Trust consist of request-response messages with flexible message components.

Despite it's flexibility, WS-Trust does not define the operations that must be executed for each message that is constructed or processed. By defining these operations, services can execute new protocols without relying on predefined protocols.

L. Aroyo et al. (Eds.): ESWC 2009, LNCS 5554, pp. 476–490, 2009.

Based on these observations we propose a middleware for automated implementation of security protocols in Web services. The proposed middleware consists of two layers: the service layer and the communication layer. The *service layer* consists of several services providing the discovery of services and the authorization of client applications. We define four types of services: name services, specification services, authorization services and resource services. The *communication layer* is built on the SOAP layer and ensures the implementation of security protocols and the implementation of service protocols. By using these protocols we provide a secure communication channel for the service protocols implemented above.

Each system implementing the proposed platform defines a single name service, a single specification service, a single authorization service and multiple resource services. Services are accessed dynamically by using service protocol and security protocol specifications. For the automated execution of security protocol specifications we propose a semantic security protocol model (SSPM). The SSPM has two components: a sequential model and an ontology model. The first component is implemented as a WSDL-S [4] specification while the second component is implemented as an OWL [15] 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 SSPM is constructed from a given security protocol model (SPM) and it must maintain the protocol's security properties. For this we propose several generating rules and algorithms that generate the SSPM.

The paper is structured as follows. In section 2 we present the architecture of the proposed middleware. We continue with the construction of the specification model in section 3. Based on the proposed middleware, in section 4 we exemplify the construction of specifications and we present a Web service-based video surveillance system, where video capturing resources can be accessed by automatically executing security protocols. We relate our work to others in section 5. We end with a conclusion and future work in section 6.

2 Middleware Architecture

2.1 Service Oriented Architecture

Client applications are able to access resources by first locating them, followed by the download of the service and security protocol specifications and by the execution of an authentication sequence. The services provided by the platform must provide a way to publish, locate and automatically access resource services. In addition, in order to face the challenges of rapidly changing protocol specifications, the protocol implementations must provide flexible and extensible components.

Based on these requirements, we define four types of services: name services, specification services, authorization services and resource services. Name services

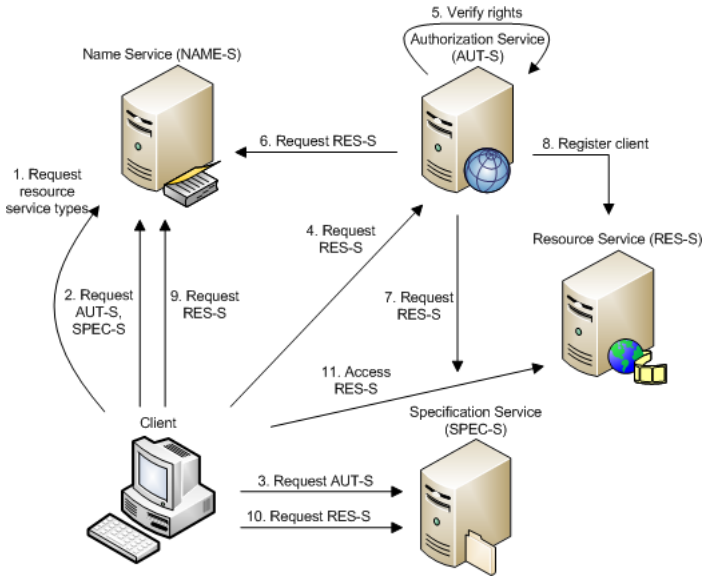


Fig. 1. Accessing services by client applications

(NAME-S) are implemented through UDDI [22] registries and are used to register, identify or locate existing services. Specifications are stored and managed by specification services (SPEC-S). Specifications are implemented using Web service technologies such as SAWSDL [12], WSDL-S [4] and OWL [15]. Authorization services (AUT-S) implement the verification mechanisms of client credentials and provide accessing mechanisms to the requested resources. Finally, the resource services (RES-S) implement a set of capabilities provided for client applications.

Accessing resources by a client application is done in several steps, as shown in figure 1. First, the client must establish the set of services implemented by the system (step 1). In order to access a resource, the client must be authenticated and its rights must be verified. This is done by accessing the AUT-S service, for which the specifications must be first downloaded. The client requests the location of AUT-S and SPEC-S (step 2) from NAME-S, followed by the request of the specifications for AUT-S (step 3). The request for accessing RES-S is sent to AUT-S (step 4), containing the user credentials. These are verified (step 5) and a security token is generated by AUT-S that is sent to RES-S (steps 6, 7, 8). The client receives the generated token and sends it to RES-S (steps 9, 10, 11), after which it is able to access the capabilities provided by the resource.

2.2 Software Architecture

The architecture of the software stack is given in figure 2. We identified two main layers: the communication layer and the service layer, given in figure 2.

The *communication layer* provides the implementation of the service and security protocols needed to access service capabilities. It is built on existing

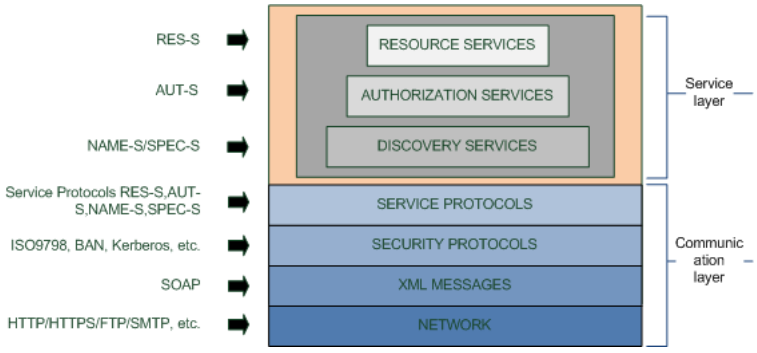


Fig. 2. Software stack

network and XML message-based protocols. Security protocols are implemented using extensions of the WS-Security standard, provided in our previous work [14]. The extensions consist of XML constructions for user names and binary keys required by key exchange and authentication protocols. The automated execution of security protocols is based on specifications developed using existing Web service technologies such as WSDL-S [4] and OWL [15], presented in the following sections. Service protocols are described by SAWSDL [12] specifications and are specific to each service. Name services define a set of messages for interrogating service registry, while specification services define messages for downloading specifications. Authorization services define messages for requesting access to services and to send security tokens. Resource services provide two types of specifications, for each external entity accessing them. The AUT-S specifications provide messages for setting user security tokens, while the client specifications provide client access messages.

The *service layer* provides the implementation of service capabilities. The capabilities of NAME-S, SPEC-S and AUT-S have already been discussed before. The capabilities of RES-S are specific to each implementation, ranging from video image capabilities to data storage capabilities. In order to implement new resource services, the only components that require change are the capabilities and the service protocol specifications. The security properties of communications with new resources are ensured by the underlying communication layer that remains unchanged.

3 Constructing Security Protocol Specifications

In order to provide an automated implementation of security protocols we need to construct a detailed specification model that includes sufficient information for protocol participants to generate and process messages. Specification models are generated from protocol models that contain a limited number of information. Based on the protocol model, we provide several rules and algorithms to generate specification models. However, this process is not entirely automated. The user

must select parameters of cryptographic algorithms such as generated key and random number sizes.

3.1 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 [19], [20] 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 encryption and decryption process makes use of cryptographic keys. Decrypting an encrypted term is only possible if participants are in the possession of the decryption key pair. In case of symmetric cryptography, the decryption key is the same as the encryption key. In case of asymmetric cryptography, there is a public-private key pair. Determining the corresponding key pair is done using the function $\_^{-1} : \mathsf{K} \rightarrow \mathsf{K}$.

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

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

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* : T, denoting a generated term; *CON_PARTAUTH* : T, denoting participant authentication; *CON_CONF* : T, denoting the confidentiality of a given term); *CON_INTEG* : T, denoting the integrity of a given term; *CON_NONREP* : T, denoting the non-repudiation property for a given term; *CON_KEYEX* : T, denoting a key exchange protocol.

The set of precondition-effect predicates is denoted by $\mathsf{PR\_CC}$ and the set of precondition-effect predicate subsets is denoted by $\mathsf{PR\_CC}^*$. The types attached to each protocol term are modeled using the following predicates: *TYPE_DN* : T to denote distinguished names, *TYPE_UD* : T to denote user-domain names, *TYPE_NT* : T to denote timestamps, *TYPE_NDH* : T to denote Diffie-Hellman random numbers, *TYPE_NA* : T to denote other random numbers, *TYPE_NDH* : $\mathsf{T} \times \mathsf{T} \times \mathsf{T} \times \mathsf{P} \times \mathsf{P}$ to denote Diffie-Hellman symmetric keys, *TYPE_KSYM* : $\mathsf{T} \times \mathsf{P} \times \mathsf{P}$ to denote symmetric keys, *TYPE_KPUB* : $\mathsf{T} \times \mathsf{P}$ to denote public keys, *TYPE_KPRV* : $\mathsf{T} \times \mathsf{P}$ to denote private keys, and *TYPE_CERT* : $\mathsf{T} \times \mathsf{P}$ do denote certificate 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,* $part \in P$ *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.*

3.2 Semantic Security Protocol Model

In this section we propose a new semantic security protocol model (SSPM) based on which we construct security protocol specifications that can be automatically executed by protocol participants. Protocols are given using their SPM model described in the previous section. Based on this model we generate the corresponding SSPM that 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.

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^**.*

By consulting the WSDL-S specification we define a message as a pair consisting of the message direction and an annotation.

Definition 6. *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 7. *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 8. *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, $(\_)^{s}_{e} : \mathsf{CONC} \rightarrow \mathsf{CONC}^{*}$ to map the set of concepts for which the given concept is parent, $(\_)_{p} : \mathsf{CONC} \rightarrow \mathsf{PROPR}^{*}$ to map the set of properties for which the given concept is domain.

3.3 Generating the Semantic Security Protocol Model

In order to generate the SSPM for a given SPM, we start with a core ontology model (OM) (figure 3) that contains concepts found in classical security protocols. The core OM was constructed by consulting security protocols found in open libraries such as SPORE [18] or the library published by John Clark [5].

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 3 by interrupted lines, while the permanent sub-ontologies are denoted by continuous lines.

The *SecurityProperty* sub-ontology contains concepts such as *Authentication*, *Confidentiality* or *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*. This sub-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 protocol, however, the defined properties are applied on all constructions. From these properties we mention: *hasKey*, *isStored*, *isVerified*.

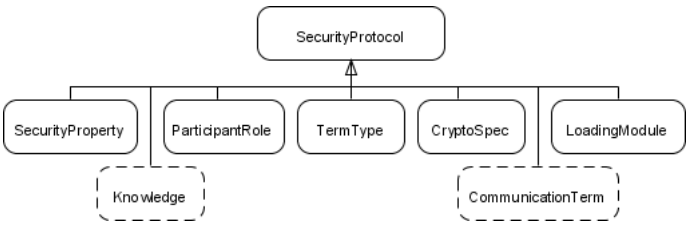


Fig. 3. Core ontology of SSPM

Next, we define a set of rules and algorithms to generate the SSPM for a given SPM. 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. For each positive or negative node there is a corresponding concept in the *SentTerm* and *ReceivedTerm* sub-ontologies, generated by the following rules:

$$\frac{n \in chain \;\; sign(n) = +}{c \leftarrow_{\mathrm{a}} gtx(term(n)) \;\; s\_msg \leftarrow_{\mathrm{r}} \{\langle out, c\rangle\} \;\; (SentTerm)^{\mathrm{s}}_{\mathrm{e}} \leftarrow_{\mathrm{r}} \{c\}}\text{msg\_tx},$$

$$\frac{n \in chain \;\; sign(n) = -}{c \leftarrow_{\mathrm{a}} grx(term(n)) \;\; s\_msg \leftarrow_{\mathrm{r}} \{\langle in, c\rangle\} \;\; (ReceivedTerm)^{\mathrm{s}}_{\mathrm{e}} \leftarrow_{\mathrm{r}} \{c\}}\text{msg\_rx}.$$

The 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. Sub-concepts of *SentTerm* are connected to sub-concepts of *Knowledge* through the *isExtracted* property, generated according to the following rule, where we used the function $\mathsf{PR\_CC}^* \rightarrow \mathsf{ID}$ to generate a new property id:

$$\frac{c \in (SentTerm)^{\mathrm{s}}_{\mathrm{e}}}{p \leftarrow_{\mathrm{a}} \langle gid(propr), isExtracted\rangle \;\; (c)_{\mathrm{p}} \leftarrow_{\mathrm{r}} \{p\} \;\; (p)_{\mathrm{c}} \in (Knowledge)^{\mathrm{s}}_{\mathrm{e}}}\text{con\_extr}.$$

Processing of received terms is done according to the type of the given 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, where we used the $E\_SYM : \mathsf{CONC}$ predicate to denote symmetric encryption:

$$\frac{c \in (ReceivedTerm)^{\mathrm{s}}_{\mathrm{e}} \;\; p \in (c)_{\mathrm{p}} \;\; (p)_{\mathrm{nm}} = isDecrypted}{c' \leftarrow_{\mathrm{a}} (p)_{\mathrm{c}} \;\; E\_SYM(c') \vee E\_SYM(c') \;\; (c') \in (DiscoveredTerm)^{\mathrm{s}}_{\mathrm{e}}}\text{con\_decr},$$

$$\frac{c \in (ReceivedTerm)^{\mathrm{s}}_{\mathrm{e}} \;\; p \in (c)_{\mathrm{p}} \;\; (p)_{\mathrm{nm}} = isStored}{c' \leftarrow_{\mathrm{a}} (p)_{\mathrm{c}} \;\; (c') \in (DiscoveredTerm)^{\mathrm{s}}_{\mathrm{e}}}\text{con\_stored},$$

$$\frac{c \in (ReceivedTerm)^{\mathrm{s}}_{\mathrm{e}} \;\; p \in (c)_{\mathrm{p}} \;\; (p)_{\mathrm{nm}} = isVerified}{c' \leftarrow_{\mathrm{a}} (p)_{\mathrm{c}} \;\; (c') \in \{(DiscoveredTerm)^{\mathrm{s}}_{\mathrm{e}} \setminus (AccessedModule)^{\mathrm{s}}_{\mathrm{e}}\}}\text{con\_verif}.$$

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. For example, the following rules define the algorithm type and key for an encrypted term that must be processed or constructed:

$$\frac{c \in (Knowledge)^{\mathrm{s}}_{\mathrm{e}} \;\; E\_SYM(c)}{p \leftarrow_{\mathrm{a}} \langle gid(propr), hasSymmAlg \rangle \;\; (c)_{\mathrm{p}} \leftarrow_{\mathrm{r}} \{p\} \;\; (c,p)_{\mathrm{ci}} \in (Symmetric)^{\mathrm{s}}_{\mathrm{e}}} \mathrm{sim\_alg},$$

$$\frac{c \in (Knowledge)^{\mathrm{s}}_{\mathrm{e}} \;\; E\_SYM(c)}{p \leftarrow_{\mathrm{a}} \langle gid(propr), hasKey \rangle \;\; (c)_{\mathrm{p}} \leftarrow_{\mathrm{r}} \{p\} \;\; (p)_{\mathrm{c}} \in (Knowledge)^{\mathrm{s}}_{\mathrm{e}}} \mathrm{sim\_key}.$$

The rules presented above are executed by algorithms. For example, modeling positive nodes in SSPM is done through the use of algorithm 1. 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.

Algorithm 1. Model positive and negative 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

4 Experimental Results

In this section we exemplify the construction of a SSPM from a given SPM and provide a few experimental result from implementing several generated SSPM.

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 [18]. 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}.

Because of space considerations, we only provide the construction of the SSPM for the A participant. Based on this, the construction of the SSPM for the second participant is straight-forward.

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. Due to space considerations, instead of describing the actual SSPM we describe the implementation of the 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 4 and part of the graphical representation of the OWL specification is given in figure 5.

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

Fig. 4. Part of the sequential model's implementation

4.2 Case Study

We have generated several security protocol specifications corresponding to protocols such as ISO9798 and Kerberos V5. These specifications were stored by the SPEC-S and were downloaded and automatically executed by client test applications and the services provided by the platform.

Based on these specifications and the proposed middleware, we implemented a video surveillance system. The basic requirements of these systems include real time image transfer, low bandwidth consumption and access control procedures [13]. To these, we add another requirement: automated security protocol execution in heterogeneous environments.

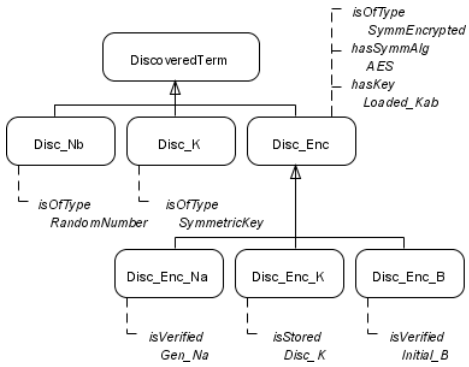
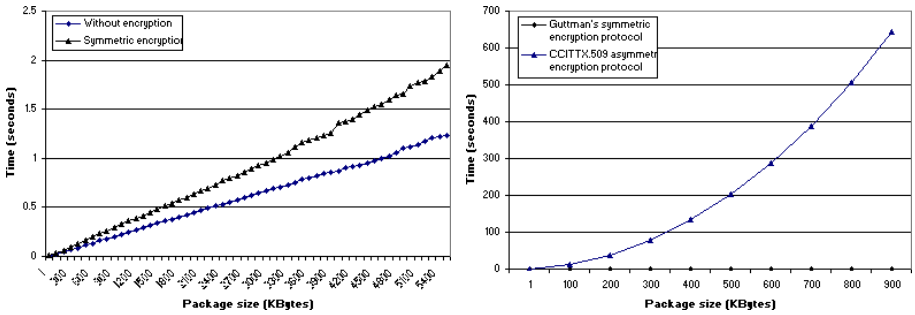


Fig. 5. Part of the ontology model's implementation



(a) (b)

Fig. 6. Performance of the communication layer: (a) Symmetrical encryption (b) Asymmetrical encryption

Our middleware satisfies all of the above formulated requirements. In figure 6 we have illustrated the performance of the communication layer. In the first case, we illustrated the performance overhead of using symmetric key-based protocols against the case when no cryptography is used. In the second case we illustrated the performance overhead of using security protocols based on asymmetric cryptography.

In order to provide automated access to resources we used asymmetric algorithms for key exchange protocols and symmetric algorithms for data transfer protocols. The keys exchanged using the first protocol types were used by the second protocols to encode data. All security protocol implementations were done using SOAP protocol header, according to the WS-Security standard.

Based on our experimental results, the time needed for a client application to access a service resource is around 520ms, as shown in figure 7, if the specifications are not cached. In case caching mechanisms are used, the time reduces to around 375ms. In both cases, if the specifications for the requested resources are

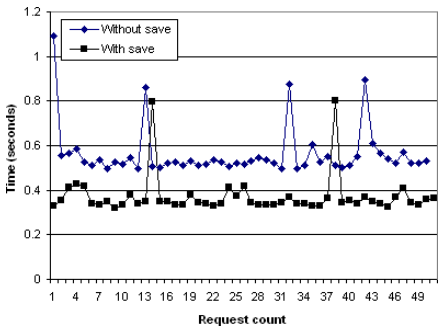


Fig. 7. Accessing system resources

not cached by AUT-S, the accessing time is much higher, as illustrated by the spikes from figure 7.

5 Related Work

An approach that aims at the automatic implementation of security protocols is given in [2]. This approach uses a formal description as a specification which is executed by participants. The proposed specification does not make use of Web service technologies, because of which inter-operability and extendability of systems executing the given specifications becomes a real issue.

Abdullah and Menasc propose in [3] a specification that is constructed as an XML document from which code is generated. The resulted code is then compiled and executed by participants. Because of this aspect, our proposal is more dynamic in the sense that applications can download and execute new protocols based on the developed specifications automatically, without having to stop program execution.

The authors from [8] propose a security ontology for resource annotation. The proposed ontology defines concepts for security and authorization, for cryptographic algorithms and for credentials. This proposal was designed to be used in the process of security protocol description and selection based on several criteria. In contrast, our ontologies, have a more detailed construction. For example, the ontology from [8] defines a collection of cryptographic algorithms, however, it does not define the algorithm mode, which is an implementation-specific information.

There have been several other security ontologies proposed [9], [10] which can be used to complete our core ontology with additional concepts and properties, for generating more complex protocol models.

6 Conclusion and Future Work

We developed a middleware for the automated execution of security protocols. The proposed middleware makes use of specifications generated from a semantic

security protocol model. 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 that map each component from the input protocol model to a component in the SSPM model. The resulted specifications were automatically executed by client applications and services implemented in a proposed video surveillance system. The experimental results have shown that the proposed middleware can be used in a wide variety of applications ranging from multimedia to eCommerce.

The proposed specification model encapsulates cryptographic properties that must be met by services and clients executing them. As future work we intend to develop a negotiation mechanism of such properties between RES-S and AUT-S as well as between client applications and AUT-S. These mechanisms can be implemented using the proposed communication layer enhanced with negotiation protocols.

References

1. Cremers, C.J.F., Mauw, S.: Checking secrecy by means of partial order reduction. In: Leue, S., Systä, T.J. (eds.) Scenarios: Models, Transformations and Tools. LNCS, vol. 3466. Springer, Heidelberg (2005)
2. Mengual, L., Barcia, N., Jimenez, E., Menasalvas, E., Setien, J., Yaguez, J.: Automatic implementation system of security protocols based on formal description techniques. In: Proceedings of the Seventh International Symposium on Computers and Communications, pp. 355–401 (2002)
3. Abdullah, I., Menascé, D.: Protocol specification and automatic implementation using XML and CBSE. In: IASTED conference on Communications, Internet and Information Technology (2003)
4. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Verma, K.: Web Service Semantics - WSDL-S. A joint UGA-IBM Technical Note (2005)
5. Clark, J., Jacob, J.: A Survey of Authentication Protocol Literature: Version 1.0. York University (1997)
6. Gavin, L.: Some new attacks upon security protocols. In: Proceedings of the 9th CSFW, pp. 162–169. IEEE Computer Society Press, Los Alamitos (1996)
7. Cremers, C.J.F.: Compositionality of Security Protocols: A Research Agenda. Electr. Notes Theor. Comput. Sci. 142, 99–110 (2006)
8. Kim, A., Luo, J., Kang, M.: Security ontology for annotating resources. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3761, pp. 1483–1499. Springer, Heidelberg (2005)
9. Blanco, C., Lasheras, J., Valencia-Garcia, R., Fernandez-Medina, E., Toval, A., Piattini, M.: A systematic review and comparison of security ontologies. In: Proc. of the Third International Conference on Availability, Reliability and Security, pp. 813–820 (2008)
10. Denkera, G., Kagal, L., Finin, T.: Security in the semantic web using owl. Information Security Technical Report 1(10), 51–58 (2005)

11. 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)
12. Martin, D., Paolucci, M., Wagner, M.: Toward Semantic Annotations of Web Services: OWL-S from the SAWSDL Perspective. In: OWL-S: Experiences and Directions - workshop at 4th European Semantic Web Conf. (2007)
13. Ostheimer, D., Lemay, S., Ghazal, M., Mayisela, D., Amer, A., Dagba, P.: A Modular Distributed Video Surveillance System Over IP. In: Electrical and Computer Engineering Canadian Conference, pp. 518–521 (2006)
14. Genge, B., Haller, P.: Extending WS-Security to Implement Security Protocols for Web Services. In: International Conference on Recent Achievements in Mechatronics, Automation, Computer Science and Robotics, Targu Mures, Romania (to appear, 2009)
15. World Wide Web Consortium, OWL Web Ontology Language Reference, W3C Recommendation (2004)
16. Gutmann, P.: Cryptlib Encryption Toolkit, http://www.cs.auckland.ac.nz/~pgut001/cryptlib/index.html
17. OpenSSL Project, version 0.9.8h, http://www.openssl.org/
18. Laboratoire Specification et Verification, Security Protocol Open Repository, http://www.lsv.ens-cachan.fr/spore
19. Organization for the Advancement of Structured Information Standards, SAML V2.0 OASIS Standard Specification (2007), http://saml.xml.org/
20. Organization for the Advancement of Structured Information Standards, OASIS Web Services Security (WSS) (2006), http://saml.xml.org/
21. Organization for the Advancement of Structured Information Standards, WS-Trust (2007), http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html
22. Organization for the Advancement of Structured Information Standards, UDDI (2004), http://www.uddi.org/pubs/uddi_v3.htm