

# Towards Automated Secure Web Service Execution

**Genge Béla and Haller Piroska**

“Petru Maior” University of Târgu Mureș, Romania  
{bgenge, phaller}@engineering.upm.ro

May 14, 2009



# Presentation overview

- Introduction
- Motivation
- Proposed specification
- Developed platform
- Experimental results
- Conclusion and future work

# Introduction

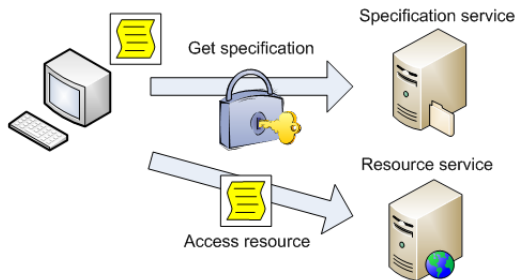
- Security protocols are “communication protocols dedicated to achieving security goals” (Cremers and Mauw) such as confidentiality, integrity or availability
- In the field of Web services, security protocols have been successfully used to provide inter-domain authentication through the use of technologies such as WS-Security, SAML or WS-Trust
- These technologies provide descriptions of security tokens, messages and security protocols to be used by participants

# Motivation

- One of the most recent approaches for ensuring secure Web service communications is WS-Trust (OASIS Standard, 2007)
- It provides descriptions of request-response authentication, key exchange, signature, negotiation, binary exchange, and many more protocols
- In order to execute a new protocol or a modified version of one of these protocols, all parties must be aware of this
- There have also been several security ontologies proposed for the classification of security protocols based on their properties

# A new security protocol specification

- We propose a specification that allows participants to automatically execute security protocols without previous knowledge on protocol messages, construction or processing operations
- Specifications are downloaded from a file service using a predefined security protocol
- Are used to create a new secure channel, to authenticate users, to exchange keys, binary data or other security tokens

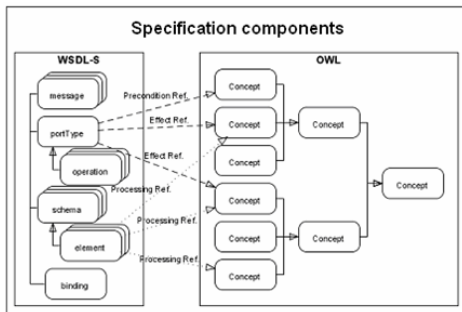


# Requirements

- Must provide a description of the cryptographic properties for constructing and processing messages
- Must provide the specification of explicit verification operations
- Must maintain the security properties of the initial protocol description
- Must use existing technologies in order to be easily integrated with existing systems

# Specification structure

- A specification is constructed for each protocol participant from two components: a sequential and an ontology component
- The *sequential* component denotes the messages that must be executed by protocol participants, preconditions and effects
- The *ontology* component denotes the structure of protocol messages, cryptographic properties, modules



# Sequential component structure

- Sequential component is given as a WSDL-S specification
- Contains:
  - Protocol preconditions: keys, user names, random numbers, security properties
  - Protocol effects: keys, user names, random numbers, security properties
  - Annotated XML schemas denoting protocol messages

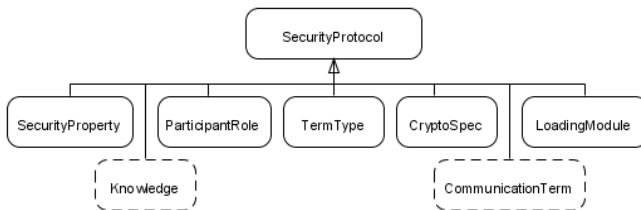
```

<xsd:element name="Msg">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Term1" type="xsd:string"
        wssem:modelReference="http://../sec.owl#Tx_1">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<wsdl:portType name="EncComm">
  <wssem:precondition name="Session key"
    wssem:modelReference="http://../sec.owl#Kab"/>
  <wssem:effect name="Key exchange"
    wssem:modelReference="http://../sec.owl#KeyExchange"/>
</wsdl:portType>

```

# Ontology component structure

- The ontology component is given as an OWL specification
- Contains:
  - Hierarchical message component representation, according to the encryption level
  - Cryptographic encryption and decryption properties: keys, algorithms, encryption modes
  - Construction and processing operations, on sent and received message components
- Specifications are constructed from a core ontology



# Enriched description

- Specifications are constructed from a slightly enriched form of the informal protocol description
- We additionally added protocol preconditions and effects for each description

```

A, B :           principal
Kab, K'ab :      symkey
Na, Nb, N'b :    nonce
succ :           nonce -> nonce

1.   A  -> B  :   A, {Na}Kab
2.   B  -> A  :   {succNa, Nb}Kab
3.   A  -> B  :   {succNb}Kab
4.   B  -> A  :   {K'ab, N'b}Kab

```

# Enriched protocol model

- Terms (i.e. message components) are constructed from the following basic sets:
  - P, denoting the set of participant names
  - N, denoting the set of random numbers
  - K, denoting the set of keys
  - C, denoting the set of certificates
  - M, denoting the set of user-defined messages
- We also defined *function names* to denote cryptographic operations, resulting the following *term* definition:

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

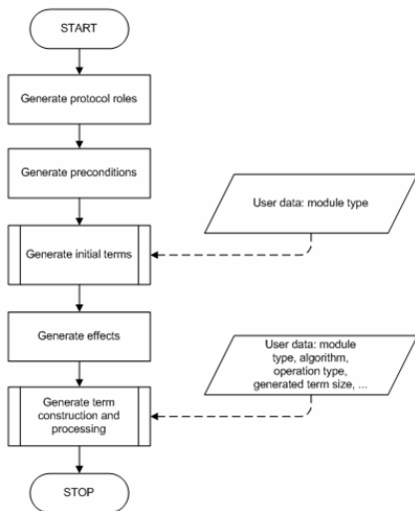
- Based on these, we defined participant models as the tuple  $\langle prec, eff, type, gen, part, chain \rangle$  and protocol models as a set of participant models

# Specification model

- The *sequential model* is defined as the triplet  $\langle s\_prec, s\_eff, s\_msg \rangle$ , where  $s\_prec \in ANNOT^*$  is a set of preconditions,  $s\_eff \in ANNOT^*$  is a set of effects and  $s\_msg \in MSG^*$  is a set of messages.
- An *ontology model* is a triplet  $\langle conc, propr, inst \rangle$ , where  $conc \in CONC$  is a set of concepts,  $propr \in PROPR$  is a set of properties and  $inst \in INST$  is a set of instances

# Constructing security protocol specifications

- Specifications are generated starting with the enriched protocol descriptions
- In order to provide a correct specification generation process, we provide a set of rules and algorithms
- Missing information is provided by the user
- Correctness of the generated specifications:
  - Initial protocol is provided by the user and considered to be correct
  - Missing information is provided by the user
  - Terms are handled by modules that must be correctly implemented



# Rules and algorithms

## Precondition description

$$\frac{pr \in prec \quad pr = CON\_TERM(t)}{c \leftarrow_a gc(t) \quad s\_prec \leftarrow_r \{\langle uri, c \rangle\} \quad (InitialTerm)_e^s \leftarrow_r \{c\}}$$

$$\frac{pr \in prec \quad pr \neq CON\_TERM(t)}{s\_prec \leftarrow_r \{\langle uri, gcc(pr) \rangle, \langle uri, gc(t) \rangle\}}$$

## Construction operation description

$$\frac{c \in (SentTerm)_e^s}{p \leftarrow_a \langle gid(propr), isExtracted \rangle \quad (c)_p \leftarrow_r \{p\} \quad (p)_c \in (Knowledge)_e^s}$$

# Rules and algorithms

## Processing operation description

$$\frac{c \in (ReceivedTerm)_e^s \quad p \in (c)_p \quad (p)_{nm} = isDecrypted}{c' \leftarrow_a (p)_c \quad E\_SYM(c') \vee E\_ASYM(c') \quad (c') \in (DiscoveredTerm)_e^s}$$

$$\frac{c \in (ReceivedTerm)_e^s \quad p \in (c)_p \quad (p)_{nm} = isStored}{c' \leftarrow_a (p)_c \quad (c') \in (DiscoveredTerm)_e^s}$$

$$\frac{c \in (ReceivedTerm)_e^s \quad p \in (c)_p \quad (p)_{nm} = isVerified}{c' \leftarrow_a (p)_c \quad (c') \in \{(DiscoveredTerm)_e^s \setminus (AccessedModule)_e^s\}}$$

# Rules and algorithms

## Part of the main algorithm

Let  $c = gc(t)$

Let  $p \leftarrow @con\_extr(c)$

**if**  $t \in KNOW$  **then**

$(p)_c \leftarrow_a c$

**else if**  $t = \{t'\}_{f(k)}$  **then**

$(GeneratedTerm)_e^s \leftarrow_r \{c\}$

Exec *ModelEncryptedGenerated*( $t$ )

**else if**  $t \in gen$  **then**

$(GeneratedTerm)_e^s \leftarrow_r \{c\}$

Exec *ModelPlainGenerated*( $t$ )

**else**

$(DiscoveredTerm)_e^s \leftarrow_r \{c\}$

Exec *ModelDiscoveredLoaded*( $t$ )

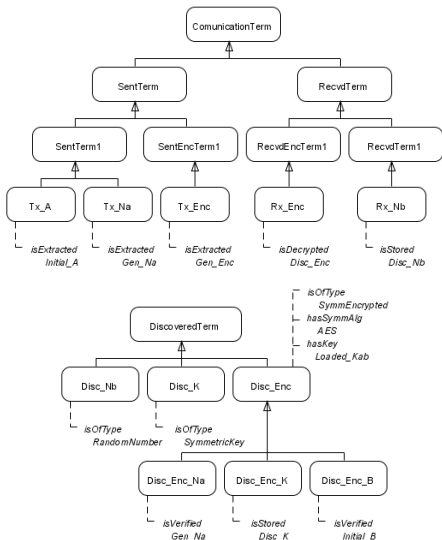
**end if**

# Part of Lowe's BAN security protocol specification

```

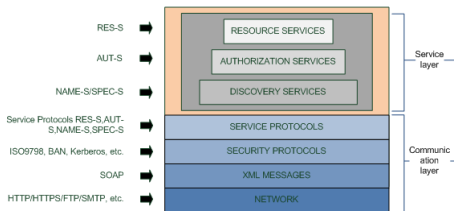
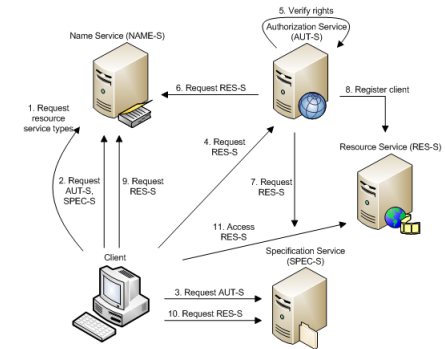
...
<xsd:element name="Msg1Request">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Term1" type="xsd:base64Binary"
        wssem:modelReference="../../../SecProt.owl#SentTerm1">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
...
<wsdl:operation name="Msg1">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

```



# Automated secure execution of Web services

- We developed a middleware for automated execution of Web services that use security protocols
- Services are located using a name service
- Specifications are downloaded from a specification service
- Requests are sent to an authorization service
- All the above mentioned are executed through secure connections provided by automated execution of specifications



## Performance characteristics

- We constructed the specification for 10 security protocols that use symmetric or asymmetric cryptography

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

## Conclusion and future work

- We developed a new specification for the automated secure execution of Web services
- The specifications are constructed from an initial enriched protocol description
- The proposed rules and algorithms provide a correct specification construction
- The specifications have been tested in a middleware, where participants were able to automatically execute authentication, key exchange and user data transfer protocols
- Construction of a tool for the automated generation of specifications
- Using the specifications in other applications that require automated execution of security protocols, such as eCommerce, Single Sign-On, Multimedia

**Thanks for your attention!**

**Questions?**