Automatic Composition of Services with Security Policies *

Yannick Chevalier† Mohamed Anis Mekki
Michaël Rusinowitch
INRIA Nancy Grand Est, 615 allée du jardin botanique Nancy, France
{chevalie,mekkimoh,rusi}@loria.fr

Abstract

Automatic composition of web services is a challenging task. Many works have considered simplified automata models that abstract away from the structure of messages exchanged by the services. For the domain of security services (such as digital signing or timestamping) we propose a novel approach to automated composition of services based on their security policies.

The approach amounts to collecting the constraints on messages, parameters and control flow from the components services and the goal service requirements. A constraint solver checks the feasability of the composition — possibly adapting the message structure while preserving the semantics— and displays the service composition as a message sequence chart. The resulting composed service can be verified automatically for ensuring that it cannot be subject to active attacks from intruders.

The services that are input to our system are provided in a declarative way using a high level specification language. The approach is fully automatic and we show on a case-study how it succeeds in deriving a composed service that is currently proposed as a product by a company.

1 Introduction

To meet frequently changing requirements and business needs, for instance in a federation of enterprises, components are replaced by *services* that are distributed over the network (e.g. the Internet) and *composed* in a demand-driven and flexible way.

Soap-based Web services. Service-oriented architectures (SOAs) have gained much attention as a unifying technical architecture that can address the challenges of this ever-evolving environment. The Web service incarnation of SOA denotes those services that communicate with one another over the HTTP protocol. Web services usually come in two flavors. We in particular are interested in this paper by the those that rely on an abstract communication layer provided by the Soap protocol, and employ a richer stack of standards issued by W3C, Oasis and others to facilitate their re-use, or *composition*, in order to achieve complex activities.

Policies. Since it is not acceptable in many cases to give access to a service to any person present on the Internet one has to regulate the use of services by *policies*. These policies express the context—including the requester's identity, her credentials, the link between the service and the requester, and higher-level business rules—to which a service is subject. They also dictate how the information transmitted between services has to be protected on the wire.

Composition of Services with Policies. Each service may rely on the existence and availability of other—possibly dynamically retrieved—services to perform its computation; this includes dynamic adaptation and explicit combination of applicable policies which determine the actions to be executed and the messages to be exchanged. For example, a service granting the access to a resource of a business partner may use a local authentication service trusted by both partners to assess the identity of a client and rely on authorisation services on both ends that combine their policies to decide whether to grant the access or not.

Related work. Many works have been dedicated to Web service composition. Several approaches are based on automata representations of Web services [3, 4, 25, 15] and reduce the composition problem to the synthesis of an automata exhibiting the same behavior as a goal service and built from available services, themselves modelled by automatas. Thus, these approaches where automata can be Petri nets or I/O automata are focusing on control flow. Another direction [20] is to apply planning techniques to address the scalability problem, retaining only those features from the services that are relevant to compose them. We follow in this paper a related goal-oriented approach but in a symbolic constraint solving framework.

Our approach. According to [25] most solutions involve too much human encoding or do not address the problem of

*Work supported by ARA SSIA Cops and AVANTSSAR FP7 216471
†On sabbatical leave from Université de Toulouse, Toulouse, France

978-0-7695-3286-8/08 $25.00 © 2008 IEEE
DOI 10.1109/SERVICES-1.2008.13

data heterogeneities, hence are still far from automatic generation of executable processes. It is thus crucial to take into account semantic and syntactic heterogeneities in messages. The resolution of the former is the goal of a large amount of works relying *e.g.* on ontological annotations. In this paper we address the latter problem: assuming that semantically equivalent constructions are represented by identical symbols, we adapt the structures of the messages exchanged between Web services. This approach is based on *Deduction systems* (expressing the possible computations on messages) and symbolic derivations (sequences of deductions and communications states) defined in [7].

An advantage of our approach is that it is possible to employ tools that were originally designed for the security analysis of cryptographic protocols such as [1] to *compose* Web services once the messages have been translated into the Dolev-Yao abstract model. These tools output a sequence of service calls that has to be carried out the orchestrator—the *composition*. An additional advantage is that it is possible to analyse the security of the composed service employing the very same tools. While all these results are valid in the Dolev-Yao abstract model of messages with cryptography, the tools can be extended to XML messages to take into account rewriting attacks [7].

Paper organization. In Sect. 2 we explain the problem that we address, show how it complements trust negotiation, and give an example of a composed service to be automatically derived from given component services. In Sect. 3 we introduce our framework. We give a quick account of the HLPSL language in which we model services as HLPSL Roles. Then we show how to encode the composition problem, the security policies of the involved services and the assumptions on the network. In Sect. 4 we introduce the associated formal model for secured Web services based on so-called *symbolic derivations*. In Sect. 5 we describe the experiments we have performed on a case-study provided by *OpenTrust* company. We show how one can derive automatically a Digital Contract Signing service from components services. We have also modified the messages to demonstrate that our approach can cope with message structure heterogeneities. Finally we prove automatically that the resulting service cannot be subject to active attacks. We conclude and present future works in Sect. 6.

2 Motivations and Approach Outline

2.1 Composition at Message Level

2.1.1 Adaptation in SOA

A key advantage of Service-Oriented Architectures (SOA) is their supposed *adaptability*. Indeed, SOA promise to simplify the deployment and administration of large software infrastructure through:

- Enabling business processes already implemented as orchestration of services to be changed rapidly;
- Enhancing adaptability by reducing maintenance.

Adaptability aims at reducing the time from a change in the business to a change in the software infrastructure. The former often are changes to the rules that drive the process. Granting a loan may for example depend on internal and external conditions and parameters such as the total sum a bank is willing to lend at a given time, interest rates, etc.

One of the advantage of the Soap web services stack is to separate a service (the actual code) from its policy (the rules governing the conditions in which this service can be employed.) In this paper we are interested in the part of a policy that translates into rules on the messages received or sent by a service, *i.e.* the policy described in a WS-SecurityPolicy [18] file attached to a service description.

2.1.2 Security Wrappers

In order to keep a clean separation between a service and its security, web services frameworks such as Axis, Glassfish, and so on distinguish the specification of a service (using WSDL [24] or BPEL [17]) from its security policy, and provide dedicated tools for specifying the latter. *Security wrappers* (or handler) implement the checks and cryptographic operations needed to:

- Verify that an inbound or outbound message satisfies the declared security policy, and is of a correct type;
- Expurge an inbound message from its cryptographic assertions and protection to transform it into a message that can be processed by the service.
- Add cryptographic assertions and protection to an outbound message.

Though the user interfaces such as Netbeans, Eclipse or Visual Web Studio provide means to quickly adapt the security policy of a service to changes in the business, we have not been able to find, either in available tools or in academic papers, a description of a tool that could, from a security policy specification, automatically generate the security wrapper around this service. For example, in Axis2, the *Rampart* module permits the service deployer to specify the sequence of cryptographic operations to apply on the in- and outbound messages, but this sequence is not automatically generated from the security policy of a service.

We believe that this lack significantly hampers the applicability of automated web service composition techniques such as those described in [3, 4, 25, 15, 20]. Indeed, these results assume that the format of the messages related to a given service is fixed, and that the security policy of services is always compatible. We note that in [25], a finer model is employed to take into account the actual payload of these messages, and thus to have a more realistic

model. Consider a service receiving a message R and replying with a message S, and let R' (resp. S') be the message R (resp. S') with the cryptographic protection required by WS-SecurityPolicy. The problem of synthesizing security wrappers can be reduced to the problem of composition with adaptation of the service that sends R' and receives S' with the service that receives R and sends S. Thus the automated security wrapper generation is a by-product of the composition result presented in this paper, that we will investigate in future works.

2.1.3 Composition and Adaptation of Web Services

There are numerous scenarios in which one would need to consider the security policy of services:

- When the security policy of a service employed in a composition changes, to check that the new policy still permits to employ the service in this composition;
- After some participants have concluded a trust negociation round acceptable by everyone, to check that each participant can play its role in the composition;
- When abstract BPEL Web service is advertised, and the services accessible to compose this abstract Web service change, to check statically (after every change) or dynamically (upon invocation) that it can be implemented given the known services and their policies;
- When changing the security policy of a Web service, to synthetize automatically its new security wrapper.

2.2 Simple Example

Let us present informally a simple composition problem involving only security services and abstracting from implementation details. Assume that a Client process needs to get a hashed (with hash function H) and timestamped (with timestamp t) value $H(H(M).t)$ of some message M. The Client has to invoke some service that does not exist yet but may be obtained as a composition of the Hash and TS services described below:

Processes	Actions	Messages
Client	sends	M
	receives	$H(H(M).t)$ signed by TS
Hash	receives	Y
	sends	$H(Y)$
TimeStamp (TS)	receives	X
	sends	$H(X, \mathit{Time})$ signed by TS

The specification of the required composed service CS is in our approach the list of incoming and outcoming messages to ensure a conversation with the Client:

CS	receives	Z
	sends	$H(H(Z).t)$ signed by TS

The available services can now be invoked in a suitable way to build the requested response to the Client, namely $H(H(M).t)$ signed by TS. The service composer has to invoke TS with $H(M)$ to get it. Hence the service composer has first to invoke the Hash service with M to get $H(M)$. As a consequence our system will derive the resulting service as the sequential composition of Hash and TS services.

The service composition problem is encoded as a deduction problem: the available services are considered as deduction rules, the goal is the final message to be received by the client (if we have several intermediate messages then they are considered as subgoals). Moreover, since we are working with security services we need to add specific deduction rules such as $\langle x, y\rangle \vdash x$ expressing that if given a pair of messages $\langle x, y\rangle$, one can deduce the message x.

2.3 Relation with Abstract BPEL

Abstract processes describe process behaviour without covering every detail of execution. Abstract processes may be used for testing *protocol matching*, i.e. for checking whether two processes provided by two business partners can interact with each other, in a multiple steps conversations. However in our case one of the two processes to be matched is obtained by some unknown composition of available services. Hence the problem is to find a suitable combination that allows for the matching.

3 From Cryptographic Protocols to Web Services

We model composed web services in the HLPSL language [1] originally designed to specify security protocols. In Subsect. 3.1 we give a brief overview of the concepts of roles and transitions underlying HLPSL. In Subsect. 3.2 we present how web services specified in BPEL or WSDL and protected by a security policy can be encoded into this language. This encoding will be employed in Sect. 4 to solve composition and adaptation problems for web services.

3.1 Introducing HLPSL

We refer to [1][1] for a presentation of the HLPSL language originally designed to specify abstract participants (roles) in cryptographic protocols. These HLPSL roles:

- are instantiated by participants, who also instantiate the parameters of the role;
- have a memory defined by an association between variables and values. This memory can be local to the roles, for ordinary variables, or shared between multiple role instances for sets;

[1] see `http://www.avispa-project.org/`

- define a set of receive/send actions. In each action:
  - tests are performed w.r.t. the received message and the memory of the agent,
  - local variables can be employed as flags to signal which action can be fired in the current state,
  - the value of variables in the memory is updated according to the received message and/or other variables in memory.

3.2 Process and Policies Encoding

3.2.1 WSDL and WS-SecurityPolicy

A WSDL [24] specification specifies operations published by the service. Operations are defined by a name, the type of their in- and output messages, and by bindings describing how they can be accessed. A WS-SecurityPolicy [18] specification specifies security assertions on the messages and the bindings declared in a WSDL specification to which it is attached. These assertions range over the specification of credentials that have to be present in a message, the parts of a message that have to be signed and/or encrypted, the transport protocol, etc. In its full generality, the WSDL+WS-SecurityPolicy association permits one to define messages in which encryption and signature are applied to all nodes in the result of some XPath query on the message.

In this paper we consider a much simpler setting. We assume that the schema defining the messages can be translated as first order terms, and that it is possible to precompute the positions in terms where encryption and signature algorithms are applied—and with which key. We also assume that the credentials are defined by first-order terms. Finally we abstract by cryptographic operations the transport protocol. If the latter:

- Provides only confidentiality, the message is encrypted using the intended receiver's public key;
- Provides authentication, the message is signed by its issuer's signature key;
- Provides both, the message is encrypted by a symmetric key known only to its issuer and intended recipient.

These simplification assumptions permit us to encode a WSDL operation with its attached security policy by an HLPSL action. This translation permits to encode stateless operations, in which case the variables appearing in a transition will be local to the transition.

3.2.2 BPEL

Since an HLPSL role is a set of actions with additional control and memory, we have chosen to encode a BPEL specification of a service by an HLPSL role. While this choice is natural, there are some subtleties that do not permit yet an exact encoding. Firstly, in a BPEL process, it is possible to define several scopes in order to declare variables that are local to a part of the process. Secondly, in BPEL, a `receive` activity comes into two flavors: either a new instance of the business process is created, or an already existing instance has to handle the incoming message. This distinction does not exist in HLPSL, as the set of active processes has to be declared before the analysis begins. This lack of precision in the translation has however the advantage of bounding the total number of processes and thereby allows one to obtain decidability for several reachability problems.

4 Formal Description of Service Composition and Adaptation

4.1 A formal model for secured Web services

Since we present in this paper a first approach to the problem of Web service composition taking into account the security policies, we have made several assumptions to reduce the problem to ones that already existing tools can solve. In particular, we rely on the Dolev-Yao model [9].

Messages. In their full generality, messages exchanged between Web services are XML documents adhering to a specified schema. These schemas permit one to define open-ended structures such as lists which are beyond the scope of existing tools, though decision procedures could in principle be implemented to take into account schemas or the XML format [7]. In this paper, we consider messages defined by terms over the signature:

$$\mathcal{F}_{\text{DY}} = \left\{ \langle \_, \_ \rangle, \{\_\}^s_\_, \{\_\}^p_\_, \_^{-1} \right\}$$

where $\langle m_1, m_2 \rangle$ denotes the concatenation of the two messages m_1 and m_2, $\{m\}^s_k$ denotes the encryption of the message m with the symmetric key k (which is itself a message), $\{m\}^p_k$ denotes the encryption of m with the public key k, and k^{-1} denotes the private key in asymmetric encryption. Digital signature of a message m is represented formally by an encryption with the private key, *i.e.* $\{m\}^p_{k^{-1}}$.

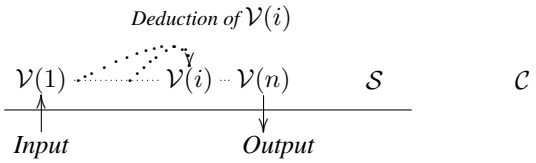
Composed services. While the HLPSL description of a service permits one to express loops, the analysis tools restrict the number of time a given transition can be fired. We thus have to assume that the BPEL specification of a composed service does not contain loops (`while`, `repeatUntil` and `forEach` activities).

Security policies. We assume, unless this is otherwise specified, that security policies are already applied on the messages. This means that, instead of specifying that the payload of a message contains two nodes a and b, and that in this message all nodes a have to be signed, and all nodes b have to be encrypted with a symmetric key, we assume we are given the term $\left\langle \{a\}^p_{k_1^{-1}}, \{b\}^s_{k_2} \right\rangle$.

In the following definition, we employ these simplifying hypotheses to introduce *symbolic derivations*, a recent model for secured Web services that we have proposed in [7]. We give here a definition parameterized by a deduction system $\mathcal{D}$ on a signature $\mathcal{F}$. This deduction system defines computation rules that express how a new term can be constructed from an existing set of terms. On $\mathcal{F}_{DY}$ this deduction system $\mathcal{D}_{\mathrm{DY}}$ expresses that one can encrypt a message or decrypt a cipher if one has the appropriate key, construct pairs, etc. We refer to [7] for a more detailed description of deduction systems.

Definition 1 *A* symbolic derivation *for a deduction system $\mathcal{D}$ is a tuple $(\mathcal{V}, \mathcal{S}, \mathcal{K}, \mathrm{In}, \mathrm{Out})$ where $\mathcal{V}$ is a mapping from a linearly ordered set $(Ind, <)$ to a set of variables $\mathrm{Var}(\mathcal{V})$, $\mathcal{K}$ is a set of ground terms (the initial knowledge) In, Out are disjoint subsets of Ind and $\mathcal{S}$ is a set of equality constraints $s \stackrel{?}{=} t$ such that for all $i \in Ind$ one of the following holds:*
input: $i \in \mathrm{In}$*;*
initialization: $\mathcal{V}(i) \stackrel{?}{=} t \in \mathcal{S}$ *with some* $t \in \mathcal{K}$*;*
computation: *There exist indices $\alpha_1, \ldots, \alpha_i < i$ and a computation rule r in $\mathcal{D}$ such that for all substitutions σ of the variables satisfying $\mathcal{S}$, the term $\mathcal{V}(i)\sigma$ is computed by r from $\mathcal{V}(\alpha_1)\sigma, \ldots, \mathcal{V}(\alpha_i)\sigma$ with $\alpha_1, \ldots, \alpha_i < i$.*
A symbolic derivation is closed *if* $\mathrm{In} = \emptyset$*. A substitution σ* satisfies *a closed symbolic derivation if $\sigma \models \mathcal{S}$. In that case we say that the closed symbolic derivation is satisfiable.*

Example 1 (Graphical representation) *To represent a secured composed service we write the list of variables in $\mathcal{V}$, and use in- or outbound arrows to represent, respectively, the input and output variables. Variables that are not indexed by In are either deduced (as shown here for $\mathcal{V}(i)$) or instantiated by a term in the set $\mathcal{K}$. Deductions and instantiations are represented by a set $\mathcal{S}$ of equality constraints $s \stackrel{?}{=} t$ between terms (including variables $\mathcal{V}(i)$).*

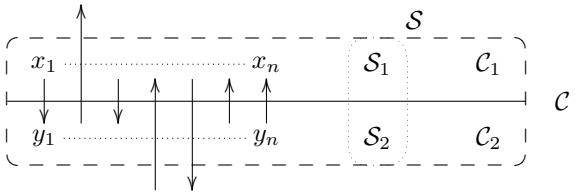


Resolving the equality constraints stemming from checks and deductions is possible in the case of the Dolev-Yao deduction system. It permits one to abstract the internals of a symbolic derivation by a sequence of patterns for input and output messages. The HLPSL is a language that in turn permits one to express these patterns together with a control on the flow of in- and output messages.

4.2 Solutions of symbolic derivations

As is described in [7], the search for an attack on a protocol represented by a symbolic derivation $\mathcal{C}$ can be reduced to the satisfiability of a non-closed symbolic derivation. A crucial remark is that, since a symbolic derivation represents internal computations and receive/send actions, *the actions of an attacker can also be represented by a symbolic derivation*. We refer to [7] for the description of how symbolic derivations can be composed (*i.e.* a message sent by one is received by the other), but that remark permits one to reduce the search for an attack on a protocol to the problem of existence of a symbolic derivation $\mathcal{C}'$ such that the composition of $\mathcal{C}$' and $\mathcal{C}$ is closed and the resulting equality constraints are satisfiable.

Example 2 *The composition of two symbolic derivations $\mathcal{C}_1$ and $\mathcal{C}_2$ identifies variables in the input of one with variables in the output of the other. Variables that have been identified are removed from the input/output set of the resulting symbolic derivation $\mathcal{C}$. The equality constraints of $\mathcal{C}$ is the union of equality constraints in $\mathcal{C}_1$ and $\mathcal{C}_2$, plus equalities stemming from the identification of input and output.*



Solution of a symbolic derivation. A symbolic derivation $\mathcal{C}'$ *solves* $\mathcal{C}$ iff there exists a composition of $\mathcal{C}'$ and $\mathcal{C}$ which is satisfiable. We note that a witness solution to the problem of deciding—given a symbolic derivation $\mathcal{C}$ and a finite set of ground terms $\mathcal{K}$—whether there exists $\mathcal{C}$' with knowledge set $\mathcal{K}$ that solves $\mathcal{C}$ contains in particular a list of cryptographic operations to apply on messages. This sequence can be compiled into a security wrapper. If the services are built from standard cryptographic operations and the number of service instances that can be composed is bounded, finding a witness solution is feasible in NP time.

5 Experimental Results

This section describes the digital contract signing case study, provided by OpenTrust and the experiments we have performed on. First we present informally the case study. We show how the decision problem presented in the previous section, is encoded and discuss the solution generated by our algorithm: a trace of the execution of the composed web service. Then we discuss some experiments with adaptation issues. Finally we prove that the generated composed service satisfies the security properties required for proper use of the composed service. This is done automatically in less than one second with Avispa Tool [1].

5.1 OpenTrust Case Study

A Business Portal (BP) is provided to two parties that plan to digitally sign a contract. First the BP application generates an electronic document corresponding to the terms of the agreement between the two parties. Then the first signer accesses the BP using a browser, checks the contract and signs it using a digital certificate. The BP verifies the signature and stores it. The second signer, in turn, connects to the Web site, checks the status of the existing signature then co-signs the contract after viewing it. Once the signature have been verified by the BP, the signers are notified and the contract is archived.

The BP system is Web service enabled. It delegates the processing of proof elements (signatures, signed documents, timestamps ...) to a Signature and Proof management Infrastructure, the Security Server, using Soap messages. The Security Server relies on the following services: PKI, Public Key Infrastructure issuing signing certificates and certificate revocation; Timestamper, timestamping server synchronized with a reliable time source; Archiver, proprietary archiving provider that guarantees long-term safekeeping for proof elements.

5.2 Automatic Derivation of the Digital Contract Signing Service

To state the problem, we have:

The composition problem we are tackling is formally defined by a set of available Web services—Timestamper, PKI, Archiver, Security Server— and an abstract process—Business Portal (BP)— specifying the interface between the clients and the expected composed service. The available services and the abstract process are modelled by HLPSL roles. For example the PKI Web service is defined by:

```
role pki (SS,PKI: agent,
    PKPKI: public_key,
    Hash : hash_func,
    CRL: (agent.public_key) set,
    SND, RCV: channel (dy)) played_by PKI ≜
local State : nat, A : agent, K : public_key
init State := 1
transition
  pki1. State = 1 ∧RCV (A'.K') ∧not (in (A'.K',CRL)) ⇒
    State'= 1 ∧SND ({Hash(A'.K')}^p_{PKPKI^-1})
end role
```

The Business Portal role specifies the interface between a client and the desired composed service. It can be viewed as the complementary sequence of messages needed to be sent and received by the composed service to fulfill the contract signing task. In the case study some assumptions are made about the communication channels between the abstract process and the available web services. These assumptions are encoded in HLPSL through the use of cryptographic keys. For example the Security Server is supposed to communicate with the business portal through HTTPS with strong authentication. This is modeled by the sharing of a symmetric key KBPSS between portal and security server. Below is the HLPSL encoding of the Business Portal:

```
role bp (S1,S2,BP,SS,ARC: agent,
    KS1BP, KS2BP, KBPSS: symmetric_key,
    PKS1, PKS2 : public_key,
    SND, RCV: channel (dy),
    Hash : hash_func,
    Contract: message) played_by BP ≜
local State : nat, SignaturePolicy : message
const EoC : text
init State := 1
transition
  bp1. State = 1 ∧RCV (start) ⇒
    State'= 2 ∧SND ({Contract.S1.PKS1.S2.PKS2}^s_{KBPSS})
  bp2. State = 2 ∧RCV ({Hash(Contract).S1}^s_{KS1BP}) ⇒
    State'= 3 ∧SND ({Hash(Contract).S1}^s_{KBPSS})
  bp3. RCV ({Hash(Contract.S1.SignaturePolicy')}^s_{KBPSS})
    ∧State = 3 ⇒State'= 4 ∧
    SND ({Hash(Contract.S1.SignaturePolicy')}^s_{KS1BP})
  bp4. RCV ({{Hash(Contract.S1.SignaturePolicy)}^p_{PKS1^-1}}^s_{KS1BP})
    ∧State = 4 ⇒State'= 5 ∧
    SND ({Hash(Contract.S1.SignaturePolicy)_inv(PKS1)}^s_{KBPSS})
        ··· bp 5–7 omitted as similar to bp 2–4 ···
  bp8. State = 8 ∧RCV ({EoC}^s_{KBPSS}) ⇒
    State'= 9 ∧SND (EoC)
end role
```

Transition bp1, states that the Business Portal first sends information about the contract to be signed (Contract), the identities (S1,S2) and the public keys of the intended signers (PKS1,PKS2), to the composed service. In transition bp2, the BP forwards to the composed service, the signature query of the first signer, then in transition bp3, it forwards the SignaturePolicy (specifying which attributes have to be added to the contract before signing it) from the composed service to the first signer. Transition bp4, is a forward of the signed contract from the first signer to the composed service. In transitions bp5, bp6 and bp7, the same operations are repeated for the second signer.

The last transition bp8 is added here to specify that if the composed service is run successfully (*i.e.* returned the

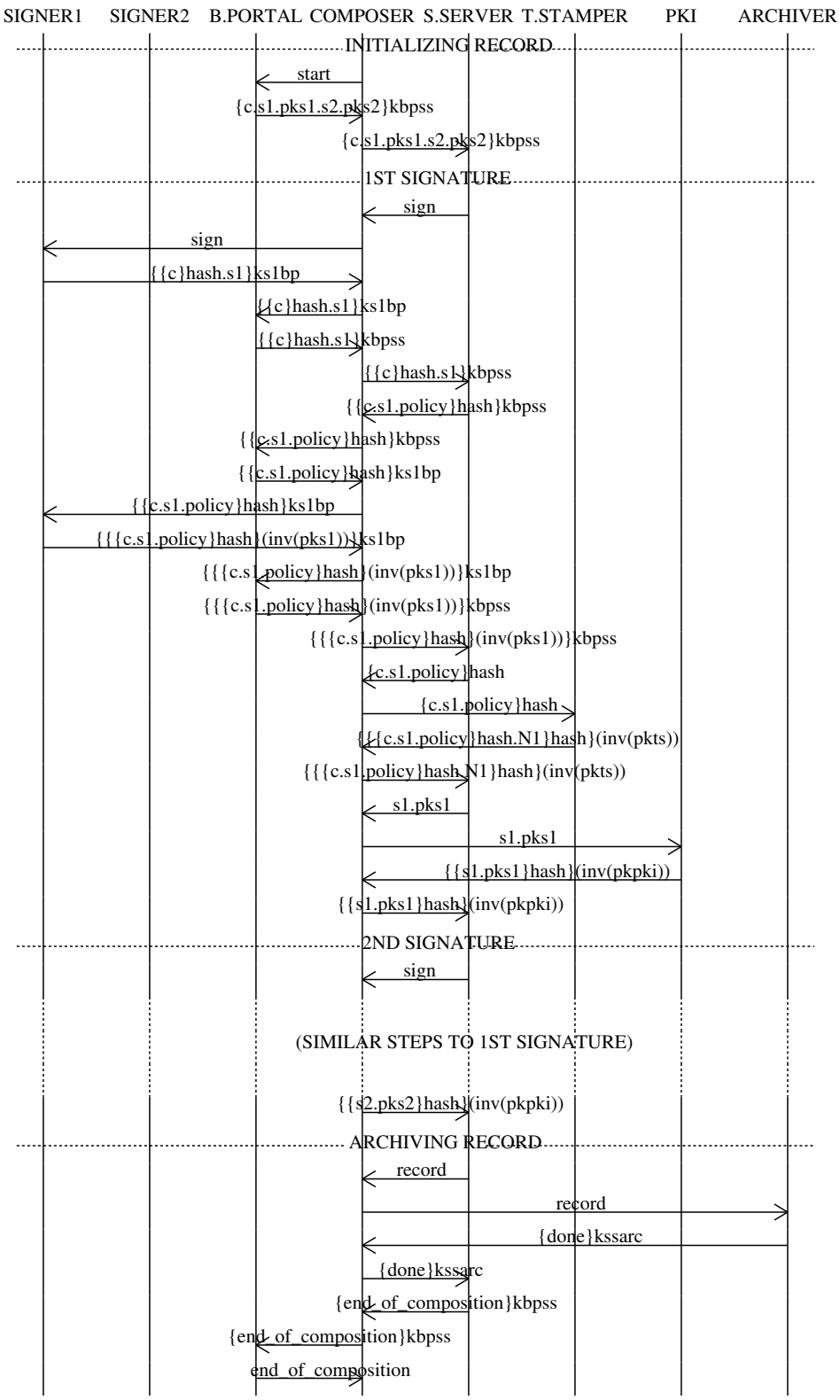


Figure 1. Sequence Diagram for Digital Contract Signing

EoC constant), the Business Portal forwards the EoC constant to the intruder. Initially the EoC, constant, is unknown to all the roles. From the way we have encoded the problem, the intruder succeeds to construct correctly the message exchange corresponding to the abstract process role, by composing the roles corresponding to the available web services, iff he can get the EoC constant in some scenario. Hence, solving the problem of composition is equivalent to solving the problem of finding an attack on the secrecy of the EoC constant, for the protocol encoding the abstract process role and the available web services roles.

The composer (the intruder): The two signers are involved in the composition, through the messages they have to exchange with the Business Portal abstract process. We can choose either to abstract them away, considering only the business part of the composition, or alternatively consider them as web services and encode their corresponding roles in HLPSL. In the first case, we must however add to the composer (the intruder) sufficient knowledge to compose the signers messsages. In this first experiment, we decided to explicitly model the two signers as HLPSL roles.

We run CL-Atse [23], one of the back-end of Avispa Tool suite, with the HLPSL specification described above. It returns an attack on the secrecy of EoC, which corresponds to the trace of the messages exchanged between the component services and the Business Portal, see Fig. 1. This trace can be directly translated to a composed service for Digital Contract Signing. In fact this automatically synthetized service is exactly the same as the one provided by OpenTrust.

5.3 Experimenting Adaptation

First we can show by experiments that the intruder ability to manipulate the messages exchanged between web services components, can be exploited in order to adapt the messages automatically to the ones that are expected by the Business Portal abstract process.

Let us consider the HLPSL role specification of the PKI web service above. The PKI disposes of a set of pairs (agent,public key) corresponding to the certificate revocation list (CRL). When it receives a pair (A,K), it checks if it is in the CRL and otherwise sends it back signed with its private key. We can invert the order of the expected pair and rely on the intruder ability to adapt the message, sent by the Security Server, asking if some pair (agent,key) is in the CRL. We can do the same experience with the timestamper role, and generalize the type of messages it is waiting for, in its unique transition. We can specify that the timestamper role is waiting for a general message M', (and not necessary some hash value of some message M), and expect the composer (the intruder) to find out that a particular hashed message still matches this new and more general pattern.

As mentioned in the previous section, we can also reconsider the scope of the composition problem. Instead of explicitly specifying the two signers as HLPSL roles, we can also give to the composer (the intruder) sufficient knowledge to construct the messages they were supposed to construct. This separates more clearly the business and the client sides in the composition.

We can generalize this approach, by reducing a large composition problem to several smaller ones: for example considering two complementary composition problems: one between the abstract process Business Portal and the business side web services, and the second between the same abstract process and the client side.

We have done some experiments taking these considerations into account: abstracting away the signers roles in the specification and generalizing the messages patterns in the business web services roles. The composition is still possible and the CL-Atse back-end of the Avispa tool, returns sensibly the same trace as in the previous section.

5.4 Security of the Composed Service

The trace computed for the composition and adaptation of services has then been translated to an HLPSL role modelling the orchestrator, and this role has been added in parallel with the roles modelling the component services. We have applied CL-Atse to verify some security properties of the resulting composition. The security requirements collected from the OpenTrust specification led us to consider the following secrecy and authentication properties:

Secrecy of the proof record: The proof record is a digital case used to gather the contract, the signatures and some additional proof elements, used for their validation (e.g. CRL). It is created in the security server context and shared with the archiver web service at the end of the execution The secrecy property considered here asserts that this knowledge is only shared between the security server and the archiver.

Authentication of the BP by the security server: The content of the contract is unknown to the security server, before the execution (Only the BP and the signers are aware of it). It is important to verify, that anytime the security server receives a contract to be signed from the business portal, the former must ensure that it was sent to him by the latter, within the same session (this guarantees the impossibility of replay attacks).

Authentication of the BP by the signers: Before signing the contract, each signer must receive the signature policy from the security server through the business portal. The policy tells the signer which attributes must be attached within and signed with the contract. The signer must be sure to receive the policy from the right agent, in order to avoid signing sensitive data.

6 Conclusion

We have proposed a new approach to generate automatically services compositions with security constraints. The key idea was to interpret available services as protocol roles, an orchestration activity as an intruder (in security protocol analysis) and executable services compositions as attacks scenarios. Then we have exploited the whole machinery that we have developped for security protocol analysis to show the feasability of the proposed approach on a real composition problem. An advantage is that we can also generate automatically the security handlers that are required to monitor/enforce the composed service policy. We plan to pursue the development of a system integrating all these features in European Project Avantssar [2] with partners SAP, IBM and Siemens. In particular we should relax some limitations we have concerning the number of service instances to appear in a composition. We aim to avoid bounding the number of instances. Doing that amounts to consider, in protocol words, an intruder extended with new deduction rules (each one associated to a service). Also we may need to build two composed services simultaneously, and for each of these compositions different services and data are available. For coping with such a situation, our constraint solving procedure needs to be revisited.

References

[1] Armando, A., et al. The Avispa Tool for the automated validation of internet security protocols and applications, CAV'05, pp. 281–285.

[2] Barbon, F., Traverso P., Pistore, M., Trainotti,M. Run-Time Monitoring of Instances and Classes of Web Service Compositions, ICWS'06, pp. 63–71.

[3] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. VLDB'05, pp. 613–624.

[4] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. WWW'03, pp. 403–410.

[5] Chevalier, Y., Rusinowitch, M. Combining Intruder Theories. ICALP'05, pp. 639–651.

[6] Chevalier, Y., Vigneron. L. Strategy for Verifying Security Protocols with Unbounded Message Size. J. of Aut. Soft. Engineering, 11(2):141–166, April 2004.

[7] Chevalier, Y., Lugiez, D., Rusinowitch, M. Towards an automatic analysis of web services security. FROCOS'07, pp. 133–147.

[8] Colomb, M., Di Nitto, E., and Mauri, M. CENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules ICSOC'06, pp. 191–202.

[9] Dolev, D., Yao, A. On the Security of Public-Key Protocols. IEEE Trans. on Inf. Theory **2**(29) (1983)

[10] Dong, H., Wang, Z., Morris, R.A. Sellers, D. Schema-Driven Security Filter Generation For Distributed Data Integration, 1st IEEE Workshop on Hot Topics in Web Systems and Technologies(HOTWEB '06), 13-14 Nov. 2006.

[11] Gagne, D., Sabbouh, M., Bennett, S., Powers, S. Using Data Semantics to Enable Automatic Composition of Web Services SCC'06, pp. 438–444.

[12] He, D., Yang, J. Security Policy Specification and Integration in Business Collaboration SCC'07, pp. 20–27.

[13] Hung, P., Fung, C. Web Services Security and Privacy. SCC'07.

[14] Kazhamiakin, R., Pistore, M., Santuari, L. Analysis of communication models in web service compositions. WWW 2006, pp. 267–276.

[15] Mitra, S., Basu, S., Kumar, R. Local and On-the-fly Choreography-based Web Service Composition, Web Intelligence, pp. 521–527.

[16] Narayanan S., McIlraith, S. Simulation, verification and automated composition of web services. WWW'02, pp. 77–88.

[17] Oasis Open. Web Services Business Process Execution Language Version 2.0. `http://docs.-oasis-open.org/wsbpel/2.0/Primer/-wsbpel-v2.0-Primer.html`

[18] Oasis Open. WS-SecurityPolicy 1.2 specification. `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/ws-securitypolicy-1.2-spec-cd-01.pdf`

[19] Pathak, J., Basu, S., Lutz, R., Honavar, V. Parallel Web Service Composition in MoSCoE: A Choreography-Based Approach. ECOWS'06, pp. 3–12.

[20] Pistore, M., Marconi, A., Bertoli, P., Traverso, P. Automated Composition of Web Services by Planning at the Knowledge Level, IJCAI'05.

[21] Singaravelu, L., Pu, C. Fine-Grain, End-to-End Security for Web Service Compositions, SCC 2007, pp. 212–219.

[22] Tziviskou, C. Di Nitto, E. Logic-based Management of Security in Web Services, SCC'07, pp. 228–235.

[23] Turuani, M. The CL-Atse Protocol Analyser. RTA'06, pp. 277–286.

[24] W3C. WSDL specification. `http://www.w3.org/TR/WSDL/`.

[25] Wu, Z., Gomadam, K., Ranabahu, A., Sheth, A., Miller, J. Automatic Composition of Semantic Web Services Using Process Mediation. ICEIS (4) 2007: 453-462

[2] see `http://www.avantssar.eu/`