Orchestration under Security Constraints
Turning Intruders into Mediators

Yannick Chevalier, Mohamed Anis Mekki, Michaël Rusinowitch

LORIA & INRIA Nancy Grand Est, France
E-mail: FirstName.LastName@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 secured services (using e.g. digital signing or timestamping) we propose a novel approach to automated composition of services based on their security policies. Given a community of services and a goal service, we reduce the problem of composing the goal from services in the community to a security problem where an intruder should intercept and redirect messages from the service community and the goal in such a way that the goal service reaches its final state, considered as insecure. 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. Moreover 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. Web services usually come in two flavors. The *Rest* Web services dialog style is richer as it fully exploits the capabilities of the HTTP protocol, while the *Soap*-based Web services rely on an abstract communication layer provided by the Soap protocol, and on a rich stack of standards issued by W3C, Oasis and others to facilitate their re-use, or *composition*, in order to achieve complex activities.

Secured Services. Since it is not acceptable in many cases to grant 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. In the following we call *secured service* a service that is protected by a security policy.

Composition of Secured Services. Each service may rely on the existence and availability of other (possibly dynamically retrieved) partner services to perform its computation. For this one needs 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.

Contribution of this work. For the domain of secured services we propose a novel approach to automated composition of services based on their security policies. Given a community of services and a goal service, we reduce the problem of composing the goal from services in the community to a security problem where an intruder should intercept and redirect messages from both the community services and the goal in such a way that the goal service reaches its final state, defined as an insecure one.

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.

Related work. Many works have been dedicated to Web service composition. Several approaches are based on automata representations of web services [6,7,33,18], trying to synthetize, from available components, an automata whose behaviour simulates the specification of the goal service. In general automata-based approach are focusing on control flow. Automata models can be Petri nets, or I/O automata for instance. The Roman model [5] focuses on deterministic atomic actions, and has been extended by data and communication capabilities in Colombo model [6].

Synthesis of composite services in [6,5] is based on Propositional Dynamic Logic. Another approach to composition relies on advanced AI planning techniques. For instance [25] applies these techniques at the knowledge level to address the scalability problem, retaining only the features of services that are relevant to compose them.

According to [33] most solutions in the literature involve too much human encoding or do not address the problem of data heterogeneities, hence are still far from automatic generation of executable processes. Given the variations in information representations such as message-level heterogeneities *data mediation* is crucial for handling service composition. Our work is in line of [20] w.r.t. its goal but with a particular emphasis on structural and semantic heterogeneity as defined by [26]. Furthermore we take into account the effects of the security policy of the services on the format of the messages.

An advantage of our approach is that it can handle automatically message structure adaptation since the orchestrator is simulated by an intruder that has capabilities (presented by a formal deduction system) to apply operations on messages and build new messages. This provides for free automatic adaptation of messages for proper service communications.

We also address the problem of checking that the composed service satisfies some security properties. For the validation of the synthetised service we can employ directly our cryptographic protocol validation tools [1].

Paper organization. In Section 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 Section 3 we introduce the associated formal model for secured Web services. In Section 4 we describe the precise encoding of orchestration problems.In particular in Subsection 4.1 we introduce our framework. We give a quick account of HLPSL language since services are expressed as Roles in this language. Then we show how to encode the composition problem, the security policies of the involved services and the assumptions on the network. In Section 5 we describe the experiments we have performed on a case-study provided by *OpenTrust* company. We show how we can derive automatically a Digital Contract Signing service from their components services. Finally we also prove automatically that the resulting service cannot be subject to active attacks. We conclude in Section 6 and give several perspectives.

2 Motivations and Approach Outline

One of the advantage of Web services is to separate a service provider, *i.e.* the actual code, from its policy, *i.e.* 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 constraints on the messages received or sent by a service, and we focus on the policy described in a WS-SecurityPolicy file attached to a service description.

Composition and Adaptation of Web Services Even though considering only the security policy of services restricts the domain in which we consider the composition problem, there are numerous scenarios in which one has to consider the adaptation of a service to changes in other services' policy, or to compose a new service that resolves the security constraints imposed by clients and some existing services. For example:

- When the security policy of a service employed in a composition changes, check that the new policy still permits to employ the service in this composition;
- After different participants have concluded a trust negotiation round acceptable by everyone, each participant has to check that it can play its role in the composition;
- To check, either statically and after every change, or dynamically when the publicized service is employed, that it can be implemented given the known Web services and their policies.

2.1 Orchestration under security constraints

Orchestration under security constraints is a particular kind of orchestration in which besides requirements on the internal actions to be performed by the goal service, security constraints are put on the messages sent and received by this service. They can be either on the format of this message, for example mandating that a timestamp has to be inserted in the header of the message, or on the payload of the payload, for example mandating that some signature keys have not been revoked.

Our approach is to transform these two kinds of constraints into constraints on messages, and to annotate existing services with assertions that are explicitly represented by message components (see Section 4). Then we apply appropriate constraint solving techniques to check the possibility of deriving the goal service.

2.2 Simple Example

To introduce our approach we will 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 from available Hash and Timestamping services.

Processes	Actions	Messages	Security constraints
Client	sends	M	
	receives	$H(H(M).t)$	$H(H(M).t)$ signed by TS
Community of services			
Hash	receives	Y	
	sends	$H(Y)$	
TimeStamp (TS)	receives	X	
	sends	$H(X, \mathit{Time})$	$H(X, \mathit{Time})$ 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. Hence the service composer has first to invoke the Hash service with M to get $H(M)$. Then the service composer has to invoke TS with $H(M)$ to get the requested message. The goal service can therefore be obtained as the sequential composition of Hash and TS services. We describe in the remaining of the paper a procedure that can be employed for generating automatically a composition of services that solves such goals.

3 Formal Description of Service Composition and Adaptation

We introduce a simple formal model for secured services that is sufficient for our applications. We first present some syntactical constructs for building messages and services. Then we specify service execution by transition semantics. Finally we define the service composition problem and reduce it to the existence of an attack on a protocol in Dolev Yao model.

3.1 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 simplifying assumptions to reduce the problem to ones that already existing tools can solve. In particular, we rely on the classical Dolev-Yao model [11] for the message security aspects.

Messages and Security policies. 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.* $\{m\}^p_{k^{-1}}$.

Assertion language. We consider a finite set of relation symbols, each symbol having an arity $n \geq 0$. An *atom* is a formula rooted by a relation symbol of arity n and with n messages as arguments. An *assertion* is a quantifier free conjunction of atoms.

Available services. We consider services that does not contain any iteration or replication. Here we shall also abstract from internal actions and we shall focus on communications. Therefore a service S will be considered as a sequence of in- and out-bound messages denoted respectively ***RCV(m)*** and ***SND(m)***, as well as conditions $c(\phi)$ or guarantees $g(\phi)$, which are both specified as assertions

For instance a ***RCV*** operation may be guarded by a *condition* $c(\phi)$ to be evaluated before accepting message m. Similarly a ***SND*** may be followed by a guarantee $g(\phi)$ in order to signal a valid property of m. For instance ϕ may be used to stipulate that the public key of the sender is validated by a trusted authority. ***Services*** are defined by the grammar:

P, Q	:=	services
	$\mathbf{0}$	null service
	$\mathit{RCV}(r) \cdot P$	message input
	$\mathit{SND}(s) \cdot P$	message output
	$c(\phi) \cdot P$	condition
	$g(\phi) \cdot P$	guarantee

Parallel composition of services Π_1 and Π_2 is denoted by $\Pi_1 | \Pi_2$. It is associative and commutative, and has a unit element 0, the null process. We consider a *community* to be a parallel composition of all its available services.

Transition semantics. We introduce transition semantics to define how services are executed in interaction with their environment and in particular with clients. The state of a service Π can be viewed as the list of remaining operations it has to perform to end properly. For instance the service in state $RCV(r) \cdot \Pi'$ should wait a message matching r with substitution σ and proceed with $\Pi'\sigma$. The global configuration is a triple $(\mathcal{S},\ \mathcal{E},\ \mathcal{G})$ with first component the set of service states, second component the set of messages that have been sent so far, and third component the set of assertions verified when sending messages. The evolution of the global configuration is given by the transition rules:

$$\overbrace{(RCV(r) \cdot \Pi'_1}^{\Pi_1} \mid \cdots,\ \mathcal{E} \cup \{m\},\ \mathcal{G}) \xrightarrow{?m} (\Pi'_1\sigma \mid \cdots,\ \mathcal{E} \cup \{m\},\ \mathcal{G})$$
where σ is a substitution such that $r\sigma = m$.

$$\overbrace{(SND(s) \cdot \Pi'_2}^{\Pi_2} \mid \cdots,\ \mathcal{E}, \mathcal{G}) \xrightarrow{!s} (\Pi'_2 \mid \cdots,\ \mathcal{E} \cup \{s\}, \mathcal{G})$$

$$\overbrace{(c(\phi) \cdot \Pi'_1}^{\Pi_1} \mid \cdots,\ \mathcal{E}, \mathcal{G}) \xrightarrow{c(\phi)} (\Pi'_1 \mid \cdots,\ \mathcal{E},\ \mathcal{G}) \qquad \text{if } \mathcal{G} \models c(\phi).$$

$$\overbrace{(g(\phi) \cdot \Pi'_1}^{\Pi_1} \mid \cdots,\ \mathcal{E},\ \mathcal{G}) \xrightarrow{g(\phi)} (\Pi'_1 \mid \cdots,\ \mathcal{E},\ \mathcal{G} \cup \{g(\phi)\})$$

A matching substitution is applied to the service receiving a new message, since in general this message bring values to instanciate some of the service variables.

A *derivation* is a sequence of transitions. We say that a service T has *ended* in a derivation if it is reduced to a null process.

Composition Goal. To answer a client $\mathcal{C}$ request we often need a new service $\mathcal{T}$ to be obtained as a composition of some of the ones that are available in the community. We define the composition goal as the ordered list of messages that $\mathcal{C}$ should receive from $\mathcal{T}$ and that $\mathcal{T}$ should receive from $\mathcal{C}$. Hence the composition goal is also a service ("the client") that can be specified with our service grammar above.

Orchestrator. Given a derivation as above, we can exploit it to generate an orchestrator: the messages sent by the services are dispatched by the mediator and they can possibly be adapted before assigning them to the proper recipient. In order to express this adaptation capability of the mediator, we simply add another transition rule denoted by $\xrightarrow{adapt}$. The $\xrightarrow{adapt}$ relation is defined *w.r.t.* a deduction relation $\triangleright$ on messages that expresses which manipulations can be performed:

$$(\mathcal{P},\ \mathcal{E},\ \mathcal{A}) \xrightarrow{adapt} (\mathcal{P},\ \mathcal{E} \cup \{m\}, \mathcal{A}) \quad \text{where } \mathcal{E} \triangleright m.$$

In this paper we take for $\triangleright$ the standard Dolev-Yao deduction model which is employed when analyzing cryptographic protocols. This means $\triangleright$ is the reflexive, transitive closure of the following rules:

pairing rules	public encryption rules	symmetric encryption rules
$\langle a, b\rangle \vdash a, b$	$\{a\}^p_K, K^{-1} \vdash a$	$\{a\}^s_b, b \vdash a$
$a, b \vdash \langle a, b\rangle$	$a, K \vdash \{a\}^p_K$	$a, b \vdash \{a\}^s_b$

3.2 Decidability Result

The problem we are interested in to check whether a client C can be satisfied by a composition of services from the community. More formally we can state it as:

Service Composition Problem

Input:	A community of service $\mathcal{S} = \{S_1, \ldots, S_n\}$ A composition goal C (specified by the client requests)
Output:	True iff there exists a sequence of transitions from initial state $(\mathcal{S} \cup \{C\},\ \emptyset,\ \emptyset)$ to a state where C has ended, and each service in $\mathcal{S}$ has either ended or is in its initial state.

In other word we have to check for the existence of a derivation (applying the transition rules) from an initial state ($\mathcal{S} = (\Pi_1 \mid \cdots \mid \Pi_2, \emptyset, \emptyset)$, to a state where all requests from the client have been satisfied (C has ended) and the services from the community that have been initiated have properly terminated. Under our hypothesis we have:

Theorem 1. *The Service Composition Problem is NP-complete.*

Sketch of proof: We reduce the Service Composition Problem to showing the existence of an attack on a protocol built from the services and the client (in Dolev Yao model).

To ensure proper termination of services that are involved in an interaction with the client, we guess at the beginning whether a service S_i will be employed or not. Let $\{S'_1, \ldots, S'_m\}$ be the subset of services to be really employed. After this guessing step the composition problem is reduced to the reachability of a configuration $(0, \mathcal{E}, \mathcal{G})$ from a configuration $(C \mid S'_1 \mid \cdots \mid S'_m, \emptyset, \emptyset)$ with $\{S'_1, \ldots, S'_m\} \subseteq \{S_1, \ldots, S_n\}$.

For each service S in $\{C, S'_1, \ldots, S'_m\}$ we introduce a new constant c_S and transform the service S into a service $\bar{S} = S.\mathit{SND}(c_S)$. It is clear that a service S reduces to the null process if, and only if, $\bar{S}$ sends c_S. Finally we add a monitor service M to the community that checks that all constants are sent. We let

$$M = \mathit{RCV}(c_C) \cdot \mathit{RCV}(c_{S'_1}) \cdot \ldots \cdot \mathit{RCV}(c_{S'_m}) \cdot \mathit{SND}(secret)$$

It is also clear that M sends $secret$ if and only if all the service $C, S'_1, \ldots, S'_m$ reduce to the null process.

Thus we have transformed the problem of the reachability of a configuration $(0, \mathcal{E}, \mathcal{G})$ from $(C \mid S'_1 \mid \cdots \mid S'_m, \emptyset, \emptyset)$ into the problem of the reachability of a configuration $(P, \mathcal{E}', \mathcal{G}')$ with $secret \in \mathcal{E}'$ from the initial configuration $(M \mid C \mid S'_1 \mid \cdots \mid S'_m, \emptyset, \emptyset)$. This latter problem is a classic problem for cryptographic protocols and is called the *Protocol insecurity problem*. Since the existence of an attack on a protocol is a problem known to be in NP [28] we can conclude. We can prove NP-hardness as for protocols too [28].

Encoding assertions as messages. To facilitate the use of existing protocol analysis tools we can get rid of assertions by encoding them into messages. We first define a mapping $p \mapsto \bar{p}$ from predicate symbols to free (*i.e.* without any deduction rule attached) function symbols. Assertions can be encoded into messages with function $[.]^m$ defined by induction as follows:

$$\begin{cases} [p(t_1, \ldots, t_n)]^m = \bar{p}(t_1, \ldots, t_n) \\ [a \wedge b]^m = \langle [a]^m, [b]^m \rangle \end{cases}$$

Conditions and guarantees encoding. We extend the encoding function $[\_]^m$ to processes as follows:

$$\begin{cases} [c(\varphi)]^m = \mathit{RCV}([\varphi]^m) \\ [g(\varphi)]^m = \mathit{SND}([\varphi]^m) \end{cases}$$

For efficiency reasons, we rewrite the processes obtained into equivalent one (w.r.t. reachability) using the rules:

$$\{\mathit{RCV}(m_1) \cdot \mathit{RCV}(m_2) \rightarrow \mathit{RCV}(\langle m_1, m_2 \rangle) \quad , \quad \mathit{SND}(m_1) \cdot \mathit{SND}(m_2) \rightarrow \mathit{SND}(\langle m_1, m_2 \rangle)\}$$

4 Specification of the orchestration problem

4.1 From Web Services to Security Protocols

In this paper we model composed web services using the HLPSL language [1] that was designed to express security protocols. In Subsection 4.1 we give a brief overview of the concepts of roles and transitions underlying HLPSL. In Subsection **??** 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 to solve composition and adaptation problems for web services.

Introducing HLPSL We refer to [1] for a presentation of the HLPSL language that has been originally designed to specify abstract participants (roles) in cryptographic protocols. It is based on the specification of a set of *roles* which are programs with parameters and on the instantiation of the parameters of these roles with *participants*. We believe this language to be sufficiently expressive to describe services as it permits one to specify, for each role a memory state in which a variable is instantiated with a term or an unbounded set of terms. The operational semantics of each role is described by a set of transition rules, each transition rule testing whether a condition on a received message and the local state is satisfied, and when this is the case, updates the local state and sends a response.

Example 1. An example of a secured service encoded by an HLPSL role is the timestamper service which is specified as follows in our case study:

```
role ts (TS: agent, PKTS: public_key, SND, RCV: channel (dy), Hash : hash_func)
played_by TS ≜
local N : text, M : message
transition
tS1. RCV(M') ⇒ N' := new () ∧SND(N'.{Hash(M'.N')}_inv(PKTS))
end role
```

This service has only one available operation, `tS1`, in which a new message M' is received. Upon reception, a new value for the timestamp N is created, and the service responds with this new value and with a proof of association of this value with the received message ($\{Hash(M'.N')\}\_inv(PKTS)$).

We can refine this example in order to show how to encode security constraints, such as guarantees:

Example 2. For example, we create a new relation symbol $TimestampOk(t_1, t_2, t_3)$ to mean that t_3 is the certified result of the attachment of timestamp t_2 on the message t_1. In the case study we have analyzed, the message t_3 is issued by a trusted timestamper service who creates t_2 and signs the hash of t_1 and t_2. This result in changing the transition of the timestamper service to:

tS1. *RCV*(*M'*) ⇒ *N'* := new() ∧ *SND*(
TimestampOk(*M'*. *N'*.{ *Hash*(*M'*. *N'*)}_inv(*PKTS*)). *N'*. *Hash*(*M'*. *N'*)_inv(*PKTS*))

One may similarly add atoms to received messages to ensure that the payload of these messages satisfies an assertion.

Security Relevant Aspects of Web Services A WSDL [32] specification specifies operations published by the service. Operations are defined by a name, by the type of input/output messages, and by bindings describing how an operation can be accessed. A WS-SecurityPolicy [23] 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. We consider a subset of the BPEL language that consists in the definition of a set of communication operations and of an ordering on these operations. The communications of a business partner are grouped into a *role*.

Assumptions and encoding. In this paper we assume that the schema defining the messages can be translated to first order terms. We also assume that the workflow of each BPEL role is totally ordered.

Each role is translated into an HLPSL role in which the communication operations are translated into a HLPSL transition. A $State$ variable is employed to ensure that transitions are fired sequentially. Each transition is also the translation of an operation in the WSDL description of the service. The messages in the HLPSL transition are the translation of the on-the-wire messages defined by the service description, possibly protected by additional cryptographic operations that represent the properties of the transport protocol.

4.2 Examples

Specification of the goal service We note that different timestamper services may output different proofs of timestamping: the hashing function may differ, or some extra information can be included in the proof,... The modeler needs however to know which timestamping proofs are accepted, and annotate the transitions of the available timestamper services accordingly.

We gather in a "`composition_goal`" service all the assertions that have to be satisfied by the goal service. In accordance with the HLPSL syntax, the variables employed in this role are local variables. The conjunction of constraints to satisfy is expressed by a set of transitions, and each transition checks that an assertion holds.

A final transition checks that all assertions are satisfied, and that the goal service can be a partner to the services it will have to communicate with. The encoding implies that if the intruder can send the message awaited by this transition then all the constraints are satisfied. Thus, we reduce the orchestration problem to the problem of firing this transition. This problem is then reduced to a secrecy problem, by giving away the secret in the response of the `composition_goal` role.

Example 3. In our case study, the transition in the `composition_goal` role that ensures that timestamps for the contracts are generated is written as follows:

subgoal_timestamped_contracts.

Recv(*TimestampOk*(*Hash*(*Contract.Client1*. *SignaturePolicy*).*TS1'.TS_Contract1'*). *TS1'.TS_Contract1'*. *TimestampOk*(*Hash*(*Contract.Client2.SignaturePolicy*). *TS2'.TS_Contract2'*).*TS2'.TS_Contract2'*)

$\wedge$ *Parameters_received* = ok $\Rightarrow$

Snd(timestamps_ok) $\wedge$ *Timestamped_contracts_received'* := ok

This transition stores the values of the timestamps ($TS1$ and $TS2$) and the proofs ($TS\_Contract1$ and $TS\_Contract2$) for later reuse. We have added a condition that ensures that the contract, the name of the clients and the signature policy have been received before:

gf.

Recv(parameters_received.signed_contracts_received.archive_checked. timestamps_ok.signature_checked) $\wedge$ *Parameters_received* = ok $\wedge$ *Signed_contracts_received* = ok $\wedge$ *Timestamped_contracts_received* = ok $\wedge$ *Archived_contracts* = ok $\Rightarrow$

Snd(composition_satisfied) $\wedge$ `secret` (composition_satisfied,success,{*Client1*,*Client2*})

Specification of the client

Example 4. In our case study, the goal service is a security server that has to communicate with a Business Portal as a "security server" partner. We extract from the BPEL, WSDL and WS-SecurityPolicy specification of the Business Portal the communication scenario between the business portal and the security server, and encode this scenario by an HLPSL role.

`role` bp (*S1*,*S2*,*BP*,*SS*,*ARC*: `agent`, *SigSent* : `hash_func`, *KBPSS*: `symmetric_key`, *PKS1*, *PKS2* : `public_key`, *SND*, *RCV*: `channel (dy)`, *Hash* : `hash_func`, *Contract*: `message`) `played_by` *BP* $\stackrel{\Delta}{=}$

`local` *State* : `nat`, *SignaturePolicy* : `message`

`init` *State* := 1

`transition`

bp1. *State* = 1 $\wedge$ *RCV*(start) $\Rightarrow$ *State*'= 2 $\wedge$ *SND*({*Contract.S1.PKS1.S2.PKS2*}_*KBPSS*)

bp2. *State* = 2 $\wedge$ *RCV*(start) $\Rightarrow$ *State*'= 3 $\wedge$ *SND*({*Hash*(*Contract*).*S1*}_*KBPSS*)

bp3. *State* = 3 $\wedge$ *RCV*({*Hash*(*Contract.S1.SignaturePolicy*')}_*KBPSS*) $\Rightarrow$ *State*'= 4 $\wedge$ *SND*(*S1*)

bp4. *State* = 4 $\wedge$

RCV(start) $\Rightarrow$ *State*'= 5 $\wedge$ *SND*(

{*SigSent*(*S1.Contract.*{*Hash*(*Contract.S1.SignaturePolicy*)}_inv(PKS1)).

{*Hash*(*Contract.S1.SignaturePolicy*)}_inv(*PKS1*)}_*KBPSS*)

bp5. *State* = 5 $\wedge$ *RCV*(start) $\Rightarrow$ *State*'= 6 $\wedge$ *SND*({*Hash*(*Contract*).*S2*}_*KBPSS*)

bp6. *State* = 6 $\wedge$ *RCV*({*Hash*(*Contract.S2.SignaturePolicy*)}_*KBPSS*) $\Rightarrow$ *State*'= 7 $\wedge$ *SND*(*S2*)

bp7. *State* = 7 $\wedge$ *RCV*(start) $\Rightarrow$ *State*'= 8 $\wedge$ *SND*(*SigSent*(*S2.Contract.*

{*Hash*(*Contract.S2.SignaturePolicy*)}_inv(*PKS2*)).{{*Hash*(*Contract.S2.SignaturePolicy*)}_inv(*PKS2*)}_*KBPSS*)

`end role`

The client service is annotated with two assertions, $SigSent(t_1, t_2, t_3)$ where t_1 is a client's name, t_2 is a contract, and t_3 is the signature of the contract by the client. We add to the `composition_goal` service a transition that checks that the two signed contracts have been received. Notice that in this case, since the interaction with the client does not involve any communication after this reception, we have omitted to add an extra assertion stating that the interaction with the business portal has finished.

subgoal_signed_contracts.
Recv(*SigSent*(*Client1.Contract.Signed_Contract1'. SigSentClient2Contract.*
Signed_Contract2'.Signed_Contract1'.Signed_Contract2') ∧ *Parameters_received* = ok ⇒
Snd(signed_contracts_received) ∧ *Signed_contracts_received'*:=ok

5 Experimental Results

This section describes the *digital contract signing* case study, provided by OpenTrust[1]. First we present informally the case study. Then we summarize the different kinds of constraints that were to be satisfied, and present the service community surrounding the goal orchestrator server. 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 two seconds with Avispa Tool [1].

5.1 Short presentation of the case study.

A *Business Portal* (BP) is provided to parties that plan to digitally sign a contract. The goal of this case study is to automatically compose a *security server* (SS) that will interact with this security portal as well as with available services to satisfy the security constraints.

Available Services. The service community that can be used by the SS is composed of:

Timestamper: An external service that provides a Time-stamping functionality. We abstract the protocol employed to communicate with this service with a simple payload exchange, and rely on an assertion (see Sect. 4) stating that this service is trusted to provide a timestamp;

PKI: A *Public Key Infrastructure* (PKI) is employed to check the validation keys of the customers. Again, we abstract away the protocols employed to communicate with this or these PKI, and rely on an assertion to characterize the PKI functionality.

Archiver: An archiver service is also accessible. We are doing coarse grain composition, and simply abstract this service with an assertion stating that this service is trusted for long-term storage.

We believe that generally speaking any type of abstract security functionality could be modelled in our framework. The presence of a given functionality in the service community is dependent on whether there exists an implementation of that functionality with which the goal service can communicate.

[1] `http://www.opentrust.com/`

Constraints. There were several constraints on the SS. We list here the main ones:

- The exchange between the BP and the SS must be secured using the HTTPS protocol. This is modeled by the sharing of a symmetric key $KBPSS$ between portal and security server (see the specification of BP in Sec. 4).
- The contracts have to be stored securely.

Client service. In addition to the compliance with the above constraints, the SS provider that we generate has to be able to be the *security server partner* in the BP process. To this end we have extracted the message exchange session with the *SS partner* from the BP definition, and imposed that the generated service interacts with the BP. In the process the customers were completely abstracted away as they only communicate with the BP.

We run CL-Atse [31], one of the back-end of Avispa Tool suite, with the HLPSL specification described above. It returns an attack on the secrecy of `composition_satisfied`, which corresponds to the trace of the messages exchanged between the component services (and the Business Portal). This trace can be directly translated to a composed service for Digital Contract Signing (see Fig. 1).

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.

In Fig. 1 we have abbreviated the most complex messages. We explain these abbreviations in the appendix.

5.2 Security Properties of the Composed Service

The trace computed for the composition and adaptation of services has been manually 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 the following security properties of the composition:

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, when all the data are collected. 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 business portal 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.

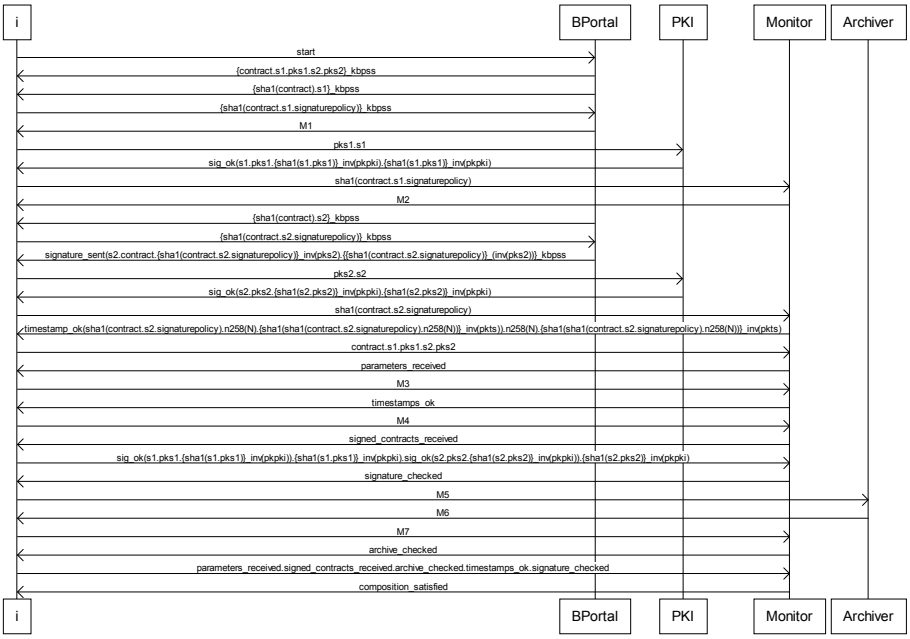

Fig. 1. Sequence Diagram for Digital Contract Signing (the intruder i stands for the security server)

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.

We have found that all these properties were satisfied when considering an active external intruder. We should stress here that by specification all communications were either local or secured (confidential and authentic).

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 enforce the composed service policy. We plan to pursue the development of a system integrating all these features in European Project Avantssar In particular we should relax some limitations we have concerning the number of service instances to appear in a composition and extend

the language of assertions to quantifier-free Horn clauses. An extension of the work we have presented will be to consider the negotiation of a partnership between two intruders. 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, `http://www.avispa-project.org/`
2. Barbon, F., Traverso P., Pistore, M., Trainotti,M. Run-Time Monitoring of Instances and Classes of Web Service Compositions, IEEE International Conference on Web Services (ICWS'06) pp. 63-71
3. Benatallah, B., Casati, F., Grigori, D., Hamid R., Nezhad, M., Toumani. F. Developing Adapters for Web Services Integration. CAiSE 2005: 415-429
4. Berardi, D., Cheikh, F., De Giacomo, G., Patrizi, F. Automatic Service Composition via Simulation. Int. J. Found. Comput. Sci. 19(2): 429-451 (2008)
5. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M. Automatic Composition of E-services That Export Their Behavior. ICSOC 2003:43-58.
6. Berardi, D., Calvanese, D., De Giacomo, G., Hull, R. and Mecella, M. Automatic composition of transition-based semantic web services with messaging. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 613–624. ACM, 2005.
7. Bultan, T., Fu, X., Hull, R., and Su, J. Conversation specification: a new approach to design and analysis of e-service composition. In *Proceedings of the international conference on World Wide Web, WWW 2003*, pages 403–410, 2003.
8. Cheikh, F., De Giacomo, G., Mecella, M. Automatic web services composition in trustaware communities. SWS 2006: 43-52.
9. Chevalier, Y., Vigneron. L. Strategy for Verifying Security Protocols with Unbounded Message Size. Journal of Automated Software Engineering, 11(2):141-166, April 2004.
10. Colomb, M., Di Nitto, E., and Mauri, M. CENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules Service-Oriented Computing - ICSOC 2006, 4th International Conference, Chicago, IL, USA, December 4-7, 2006,
11. Dolev, D., Yao, A. On the Security of Public-Key Protocols. IEEE Transactions on Information Theory **2**(29) (1983)
12. 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, 2006 (HOTWEB '06), 13-14 Nov. 2006.
13. Gagne, D., Sabbouh, M., Bennett, S., Powers, S. Using Data Semantics to Enable Automatic Composition of Web Services IEEE International Conference on Services Computing (SCC 2006), 18-22 September 2006, Chicago, Illinois, USA, pp. 438-444
14. Hassen, R., Nourine, L., Toumani, F. Protocol-Based Web Service Composition. ICSOC 2008: 38-53
15. He, D., Yang, J. Security Policy Specification and Integration in Business Collaboration IEEE International Conference on Services Computing (SCC 2007) pp. 20-27
16. Hung, P., Fung, C. Web Services Security and Privacy. IEEE International Conference on Services Computing (SCC 2007), 9-13 July 2007, Salt Lake City, Utah, USA
17. Kazhamiakin, R., Pistore, M., Santuari, L. Analysis of communication models in web service compositions. WWW 2006: 267-276.

18. Mitra, S., Basu, S., Kumar, R. Local and On-the-fly Choreography-based Web Service Composition, Web Intelligence, IEEE/WIC/ACM International Conference on 2-5 Nov. 2007 Page(s):521 - 527
19. Monfroy, E., Perrin, O., Ringeissen, C. Dynamic Web Services Provisioning with Constraints. OTM Conferences (1) 2008: 26-43
20. Nagarajan, M., Verma, K., Sheth, A.P., Miller, J.A.: Ontology driven data mediation in web services. Int. J. Web Service Res. **4** (2007) 104–126
21. Narayanan S., McIlraith, S. Simulation, verification and automated composition of web services. Proceedings of the 11th international conference on World Wide Web. Pages: 77 - 88, 2002.
22. 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`
23. 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`
24. Pathak, J., Basu, S., Lutz, R., Honavar, V. Parallel Web Service Composition in MoSCoE: A Choreography-Based Approach. ECOWS 2006: 3-12
25. Pistore, M., Marconi, A., Bertoli, P., Traverso, P. Automated Composition of Web Services by Planning at the Knowledge Level, International Joint Conference on Artificial Intelligence (IJCAI) 2005.
26. Sheth, A.P., Kashyap, V.: So far (schematically) yet so near (semantically). In Hsiao, D.K., Neuhold, E.J., Sacks-Davis, R., eds.: DS-5. Volume A-25 of IFIP Transactions., North-Holland (1992) 283–312
27. Rits, M., Rahaman, M.A. Secure Soap Requests in Enterprise SOA. In: ACSAC. (2006)
28. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc.14th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, June 2001.
29. Singaravelu, L., Pu, C. Fine-Grain, End-to-End Security for Web Service Compositions, IEEE International Conference on Services Computing (SCC 2007) pp. 212-219.
30. Tziviskou, C. Di Nitto, E. Logic-based Management of Security in Web Services, IEEE International Conference on Services Computing (SCC 2007) pp. 228-235.
31. Turuani, M. The CL-Atse Protocol Analyser. In Term Rewriting and Applications - Proc. of RTA, volume 4098 of Lecture Notes in Computer Science, pages 277-286, Seattle, WA, USA, 2006.
32. W3C. WSDL specification. `http://www.w3.org/TR/WSDL/`.
33. Wu, Z., Gomadam, K., Ranabahu, A., Sheth, A., Miller, J. Automatic Composition of Semantic Web Services Using Process Mediation. ICEIS (4) 2007: 453-462

A Abbreviations of Fig. 1

$$
\begin{aligned}
M_1 &= \{signature\_sent(s1 \cdot contract \cdot \{sha1(contract \cdot s1 \cdot signaturepolicy)\}\_inv(pks1)) \\
&\quad \cdot \{sha1(contract \cdot s1 \cdot signaturepolicy)\}\_(inv(pks1))\}\_kbpss \\
M_2 &= timestamp\_ok(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N) \\
&\quad \cdot sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\_(inv(pkts))) \cdot n267(N) \\
&\quad \cdot \{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}\_(inv(pkts)) \\
M_3 &= timestamp\_ok(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N) \\
&\quad \cdot \{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}\_inv(pkts)) \\
&\quad \cdot timestamp\_ok(sha1(contract \cdot s2 \cdot signaturepolicy) \cdot n258(N) \\
&\quad \cdot \{sha1(sha1(contract \cdot s2 \cdot signaturepolicy) \cdot n258(N))\}\_inv(pkts) \cdot n267(N) \\
&\quad \cdot \{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}\_inv(pkts) \cdot n258(N) \\
&\quad \cdot \{sha1(sha1(contract \cdot s2 \cdot signaturepolicy) \cdot n258(N))\}\_inv(pkts) \\
M_4 &= signature\_sent(s1 \cdot contract \cdot \{sha1(contract \cdot s1 \cdot signaturepolicy)\}\_inv(pks1)) \\
&\quad \cdot signature\_sent(s2 \cdot contract \cdot \{sha1(contract \cdot s2 \cdot signaturepolicy)\}\_inv(pks2) \\
&\quad \cdot \{sha1(contract \cdot s1 \cdot signaturepolicy)\}\_inv(pks1) \cdot \{sha1(contract \cdot s2 \cdot signaturepolicy)\}\_inv(pks2) \\
M_5 &= \{contract \cdot s1 \cdot signaturepolicy \cdot \{sha1(contract \cdot s1 \cdot signaturepolicy)\}\_(inv(pks1)) \\
&\quad \cdot n267(N) \cdot \{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}\_(inv(pkts)) \\
&\quad \cdot \{sha1(s1 \cdot pks1)\}\_inv(pkpki) \cdot \{sha1(contract \cdot s2 \cdot signaturepolicy)\}\_(inv(pks2)) \cdot n258(N) \\
&\quad \cdot \{sha1(sha1(contract \cdot s2 \cdot signaturepolicy) \cdot n258(N))\}\_inv(pkts) \cdot \{sha1(s2 \cdot pks2)\}\_inv(pkpki)\}\_kssarc \\
M_6 &= archive\_ok(contract \cdot s1 \cdot signaturepolicy \cdot \{sha1(contract \cdot s1 \cdot SignaturePolicy(3))\}\_inv(pks1) \\
&\quad \cdot n267(N) \cdot \{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}\_(inv(pkts)) \cdot \{sha1(s1 \cdot pks1)\}\_inv(pkpki) \\
&\quad \cdot \{sha1(contract \cdot s2 \cdot signaturepolicy)\}\_inv(pks2) \cdot n258(N) \\
&\quad \cdot \{sha1(sha1(contract \cdot s2 \cdot signaturepolicy) \cdot n258(N))\}\_inv(pkts) \cdot \{sha1(s2 \cdot pks2)\}\_inv(pkpki)) \\
M_7 &= archive\_ok(contract \cdot s1 \cdot signaturepolicy \cdot \{sha1(contract \cdot s1 \cdot signaturepolicy)\}\_inv(pks1) \\
&\quad \cdot n267(N) \cdot \{sha1(sha1(contract \cdot s1 \cdot signaturepolicy) \cdot n267(N))\}\_inv(pkts) \cdot \\
&\quad \{sha1(s1 \cdot pks1)\}\_inv(pkpki) \cdot \{sha1(contract \cdot s2 \cdot signaturepolicy)\}\_inv(pks2) \cdot n258(N) \\
&\quad \cdot \{sha1(contract \cdot s2 \cdot signaturepolicy \cdot n258(N))\}\_inv(pkts) \cdot \{sha1(s2 \cdot pks2)\}\_(inv(pkpki)))
\end{aligned}
$$