Test-Based Security Certification of Composite Services

MARCO ANISETTI, Università degli Studi di Milano, Italy
CLAUDIO ARDAGNA, Università degli Studi di Milano, Italy
ERNESTO DAMIANI, Khalifa University, UAE
GIANLUCA POLEGRI, Engineering Ingegneria Informatica S.p.A., Italy

The diffusion of service-based and cloud-based systems has brought to a scenario where software is often made available as services, offered as commodities over corporate networks or the global net. This scenario supports the definition of business processes as composite services, which are implemented via either static or runtime composition of offerings provided by different suppliers. Fast and accurate evaluation of services' security properties becomes then a fundamental requirement and is nowadays part of the software development process. In this paper, we show how the verification of security properties of composite services can be handled by test-based security certification, and built to be effective and efficient in dynamic composition scenarios. Our approach builds on existing security certification schemes for monolithic services and extends them towards service compositions. It virtually certifies composite services, starting from certificates awarded to the component services. We describe three heuristic algorithms for generating runtime test-based evidence of the composite service holding the properties. These algorithms are compared with the corresponding exhaustive algorithm to evaluate their quality and performance. We also evaluate the proposed approach in a real-world industrial scenario, which considers ENGpay online payment system of Engineering Ingegneria Informatica S.p.A. The proposed industrial evaluation presents the utility and generality of the proposed approach by showing how certification results can be used as a basis to establish compliance to Payment Card Industry Data Security Standard (PCI DSS).

CCS Concepts: • **Security and privacy → Distributed systems security**; **Web application security**; • **Software and its engineering → Software verification and validation**; • **Computer systems organization** → *Cloud computing*;

Additional Key Words and Phrases: Cloud, Model-Based Testing, Service-Oriented Architecture, Security Certification, Service Composition, Software-as-a-Service, Web Services

ACM Reference Format:
Marco Anisetti, Claudio Ardagna, Ernesto Damiani, and Gianluca Polegri. 2018. Test-Based Security Certification of Composite Services. 1, 1 (August 2018), 43 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Current trends in software distribution and provisioning envision services made available as commodities over the Internet or in the cloud marketplace. Business processes are increasingly implemented by either statically or dynamically composing many commodity services in the cloud (Business Process as a Service – BPaaS) [75, 76], each one providing a single functionality.

Authors' addresses: Marco Anisetti, Università degli Studi di Milano, Via Celoria 18, Milano, MI, 20131, Italy, marco.anisetti@unimi.it; Claudio Ardagna, Università degli Studi di Milano, Via Celoria 18, Milano, MI, 20131, Italy, claudio.ardagna@unimi.it; Ernesto Damiani, Khalifa University, Abu Dhabi, UAE, ernesto.damiani@kustar.ac.ae; Gianluca Polegri, Engineering Ingegneria Informatica S.p.A., Via S. Martino della Battaglia 56, Roma, RM, 00185, Italy, gianluca.polegri@eng.it.

XXXX-XXXX/2018/8-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Business process management and services are then rapidly converging with Platform-as-a-Service (PaaS) [57, 75, 76], providing a single, integrated cloud environment that combines application development and process management. According to Gartner,[1] BPaaS is one of the largest segments of the global cloud services market with $40.8 billion in 2016 and expected to reach $56.1 billion at the end of 2020. This market growth is following, on one side, the boost of microservice infrastructures, where extreme dynamic scenarios made of business processes composed of single-functionality services materialize, and, on the other side, the evolution of software development processes where after-testing, manual composition of services provided by different suppliers is implemented. In this vision of IT, services are either statically or dynamically integrated, remotely accessed, and continuously (re-)designed and released. To support this scenario we need accurate, efficient, low-cost, and robust assurance evaluation of service security.

Early research on security assurance has mainly focused on approaches where security properties of software/services are claimed by suppliers themselves, with little or no evidence [31, 63, 96]. Recently, security assurance (e.g., [9, 13, 22, 61, 62, 79, 83, 87]) is becoming part of the software development process, and focuses on efficent and effective independent evaluation of services and their properties. Among existing assurance approaches, certification is becoming increasingly popular to provide evidence that a given software/service system has the desired non-functional properties and behaves as expected [30]. Certification processes are traditionally managed by an accredited evaluation body called *certification authority*, in charge of evaluation activities and certificate issuing. Evaluation activities are carried out by an *accredited lab*, which focuses on collecting evidence supporting the certification of a given property. This evidence is collected by means of either testing activities or formal methods, and refers to a single component or system. Approaches to (cloud) service certification (e.g., [13, 54, 61, 87]) have been inspired by traditional software certification solutions (e.g., [30, 46, 92]) aimed at monolithic systems and do not usually consider runtime composition. Among traditional solutions, Common Criteria (ISO 15408) [46] is the most adopted and recognized International standard for software certification. It provides 7 assurance levels (EALs), representing 7 certification strengths. Each level describes the requirements a product must satisfy to get certified, including requirements on collection activities: *i)* certification at EAL1-EAL4 requires testing activities; *ii)* certification at EAL5-EAL7 requires (semi-)formal verification. 2224 software products are currently certified by Common Criteria, 68% of which are verified between EAL1 and EAL4. EAL4 is the biggest certification level with 36% of certified products and requires fine-grained testing of the software product. These numbers show that formal methods, though more trustworthy, are less adopted for software certification. This is due to the fact that they are more complex, more costly, less applicable, and less flexible than testing. This scenario is even exacerbated in dynamic scenarios like service composition in the cloud, where formal methods fall short of achieving the main requirements for cloud certification, making them inapplicable: low-cost adaptation and ability to react to composition events.

In this paper, we present a novel solution aimed to certify security properties of business processes, implemented as composite services and deployed at cloud application layer (BPaaS). While some architectural support for securing compositions has been described in the literature [50], there is currently no general way to derive security properties of a composition from the properties certified for individual components, regardless of the adopted composition model. Service composition in the cloud in fact introduces a loss of control on security and availability due to the dynamic, distributed and heterogeneous nature of cloud services. We consider Business Process Model and Notation (BPMN) [74], a generic and standard notation used for modeling business processes and service compositions, and propose a test-based security certification process that dynamically

[1] http://www.gartner.com/newsroom/id/3616417

generates BPMN-compliant compositions that hold a set of security properties. BPMN-compliant compositions can be implemented using both an executable BPMN (i.e. a BPMN executed within a framework for BPM execution like VRESCo [47]) or other executable languages such as Web Service Business Process Execution Language (WS-BPEL[2]) [4]. For the sake of simplicity but with no lack of generality, in this paper we use BPEL as the language for describing executable business processes. BPEL in fact has shown sufficient generality to cover composition of both SOAP and RESTful services [77], and is one of the most used languages for business process description [58]. Our basic idea is to produce a virtual test-based security certificate for a composition on the basis of the certificates of its component services. We call this certificate "virtual" since it does not involve any real testing activity on the composition; the test-based evidence proving a property is inferred from evidence originally used to certify the individual components (*evidence composition*). The trust in a virtual certificate is based on the trust the customer has in the automatic process implemented by the certification authority and in the real certificates held by the individual component services, coupled with the assumption that a customer prefers to have a certificate signed by a trusted certification authority rather than self-signed certificates or no certificates at all. Clearly, being based on virtual evidence and certificates, test-based virtual certification achieves a level of trust lower than the one achieved when certification of service compositions from scratch is adopted. This is the price we need to pay to *i)* accomplish the peculiarities of a cloud and service-based environment, where the high dynamics and flexibility of service composition make existing solutions, requiring certification from scratch at each service evolution, inapplicable (e.g., component service replacement/migration), *ii)* implement a repeatable certification, meaning that it can be re-executed to verify its quality. Our approach being part of the software development process aims to increase the efficiency of the certification process and its collection activities, and optimize the quality of the certification results and, in turn, of the composite services. We start by extending the BPEL process specification with security requirements and present a service selection process for the generation of security-enhanced BPEL process instances at runtime. Then, we consider the generation of a virtual security certificate for the BPEL instance with highest quality, prove that the corresponding problem is NP-hard, and propose three heuristic algorithms for its computation.

The contribution of this paper is therefore fourfold: *i)* the ability to certify security properties of service compositions, which are built by composing services either manually or at runtime, *ii)* the generation of high-quality virtual certificates for service compositions, where test-based evidence supporting certified security properties is produced on the basis of existing evidence in certificates of component services, *iii)* a quantitative approach for calculating the quality of virtual certificates, driving component service selection, and *iv)* the support for a multi-step chain of trust grounded in the certificates of the component services and rules defined by the certification authority.

This paper develops on our previous work [7] by providing *i)* a new and general-purpose approach to certification-aware service composition, *ii)* a new technique for the generation of machine-readable virtual certificates including virtual test cases, and *iii)* a quantitative evaluation of the quality of the virtual certification process and corresponding service composition. We provide an extensive experimental evaluation of our approach in terms of quality and performance. We further evaluate the generality and applicability of our approach in a real industrial scenario, which considers the certification of the ENGpay system for online payments of Engineering Ingegneria Informatica S.p.A., the largest Software and Information Technology services group in Italy (and one of the largest in Europe). The industrial evaluation shows how certification results can be used to establish compliance to internationally-recognized security standards. In particular, we prove

[2]BPEL in the following

ENGpay compliance with respect to Payment Card Industry Data Security Standard (PCI DSS) guidelines and regulations, concerning data security and privacy, and financial transactions.

The remainder of this paper is organized as follows. Section 2 presents the approach for single service certification. Section 3 describes a BPEL-based service composition using our reference scenario, and the corresponding algebra for composing service certificates. Section 4 describes our approach to the generation of virtual certificates for composite services starting from the certificates of component services. Section 5 presents the algorithm to produce certification-aware service compositions with virtual properties. Section 6 illustrates the heuristic algorithms for the generation of certified BPEL instances with highest certificate quality. Section 7 proposes the experimental evaluation of our approach. Section 8 presents the application of our approach to an industrial case study. Finally, Section 9 discusses related work and Section 10 gives our concluding remarks.

2 SECURITY CERTIFICATION OF SINGLE SERVICES

The approach presented in this paper extends our security certification scheme for monolithic services [13] to the certification of composite services. [13] considered a scenario where a service provider engages with a certification authority to certify the security properties of a single monolithic service. The certification authority manages a certification process that takes as input the property to be certified for the service and (a subset of) the service specifications, possibly including Web Services Description Language (WSDL) interface, the Web Services Conversation Language (WSCL) document, and the service implementation, and returns as output a certificate $C(p,m,e)$. $C(p,m,e)$ specifies certified property p, service model m, and a set of test-based evidence e supporting the property, as discussed below.

Security property. A security property p drives the certification process and all activities done by the certification authority. It is a pair $(\hat{p},Attr)$, where $p.\hat{p}$ is an abstract property (i.e., a label from a shared controlled vocabulary, such as confidentiality, integrity [27, 28, 37, 49]) and $p.Attr$ is a set of class attributes specifying the threats the service proves to counteract or the specific characteristics of the security function implemented by the service. For instance, property p_1=(*Confidentiality*,{`ctx`=in-storage, `algo`=DES, `key`=112}) refers to the confidentiality of data at rest using DES algorithm with key length of 112bits. A partial order can be straightforwardly defined over security properties based on attribute values, inducing a hierarchy $\mathcal{H}_P$ of properties as a pair $(\mathcal{P},\preceq_P)$, where $\mathcal{P}$ is the set of properties and $\preceq_P$ the partial order. Given two properties p_i and p_j, we write $p_i \preceq_P p_j$ if p_i is weaker than p_j. For instance, given p_1 above and p_2=(*Confidentiality*,{`ctx`=in-transit in-storage, `algo`=DES, `key`=112}) referring to the confidentiality of data at rest and in transit over the communication channel, using DES algorithm with key length of 112bits, $p_1 \preceq_P p_2$ since p_2 is stronger than p_1. In particular, p_2 extends p_1 by adding an additional context (i.e., in-transit) over which confidentiality must be preserved. Figure 1(a) shows an example of property hierarchy $\mathcal{H}_P$.

Service model. Service model m is a Symbolic Transition System (STS) [38] that specifies service behavior and execution flows as a finite state automaton. It can be used for automatic generation of test cases collecting the evidence in the certificates [26, 69]. Formally, an STS is a tuple $\langle \mathcal{S},s_1,\mathcal{V},\mathcal{I},\mathcal{A},\rightarrow\rangle$, where $\mathcal{S}$ is a set of states, $s_1 \in \mathcal{S}$ is the initial state, $\mathcal{V}$ is a set of internal variables, $\mathcal{I}$ is a set of interaction variables, $\mathcal{A}$ is a set of actions (web service operations), and $\rightarrow$ is a transition relation. Each transition relation consists of a set of edges connecting two states and labeled with an action, a guard (conditions on transition), and an update mapping.

In [13], a service model m can be defined at two levels of granularity depending on the amount of information (i.e., WSDL interface and/or WSCL document) available on the service. In this paper we consider the first level only (WSDL interface), that is, the least set of information a service provider

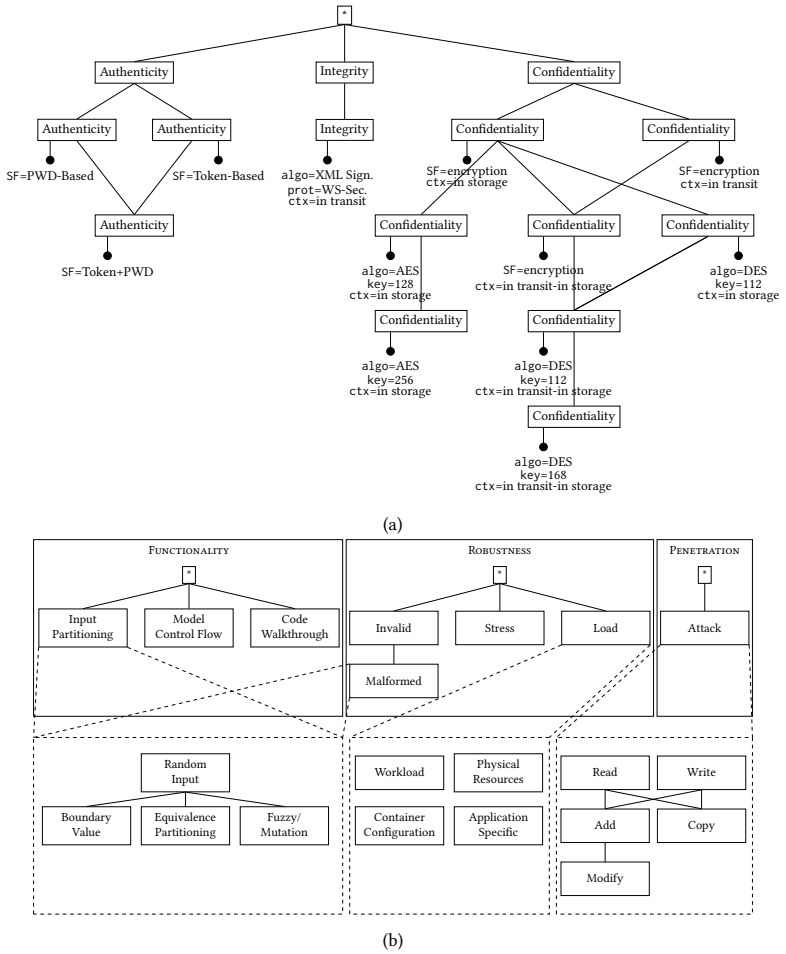


Fig. 1. Property (a) and test type (b) hierarchies

has to release to publish its service. The WSDL interface specifies the set of service operations and the methods for accessing them, and is used to define a WSDL-based model M_{wsdl} of the service. M_{wsdl} is an STS that consists of a set $\{m_{wsdl}\}$ of connected components, each one modeling a single service operation. Each m_{wsdl} is in turn modeled as an STS with three states representing the operation interface as follows: *i)* the initial state $s_1 \in \mathcal{S}$ when no input has been received yet; *ii)* the intermediate state $s_2 \in \mathcal{S}$ that is reached when the received input triggers the state transition, but no output has been generated yet; *iii)* the final state $s_3 \in \mathcal{S}$ that is reached whenever the output has been produced and returned to the counterpart. State s_2 of each $m_{wsdl} \in M_{wsdl}$ can be extended with additional states modeling the implementation of the corresponding operation. We remark that a service can be certified for a property holding for a subset of its operations in the WSDL. Connected components $m_{wsdl} \in M_{wsdl}$ of the component services are integrated to produce a BPEL-based model M_{bpel} of a composition. M_{bpel} is an STS $\langle \mathcal{S}, s_1, \mathcal{V}, \mathcal{I}, \mathcal{A}, \rightarrow \rangle$, where $\mathcal{S}$ is the union of all the states in the interface-based model of the integrated components. M_{bpel} can be seen as a generalization of the interface-based model in [13].

Test Evidence. Test evidence e specifies the set of test cases executed on the service, each with a reference to the WSDL operation under evaluation. It also includes a category and a type, as well as a set of test attributes and the results of test case execution. Test cases can be classified according to three categories of tests: *i)* category *functionality* including functional test cases based on valid input, *ii)* category *robustness* including test cases based on invalid and malformed input, and stress/load tests, *iii)* category *penetration* including test cases based on well-known security attacks. Test categories functionality and robustness are employed during a security certification process to check whether the needed security mechanisms are in place, functionally correct, and robust. Each category has a set of test attributes describing the set of test cases (e.g., the cardinality of the test set). Each category also includes a set of test types that specify how to generate the test cases needed to support security properties on a given service. We note that test types (e.g., random input, equivalence partitioning) correspond to test design techniques of [93]. More in detail, each category *cat* introduces a hierarchy $\mathcal{H}_{cat}$ of test types based on test attributes. $\mathcal{H}_{cat}$ is a pair $(\mathcal{T},\preceq_{cat})$, where $\mathcal{T}$ is the set of test types for *cat* and $\preceq_{cat}$ is a partial order relationship over $\mathcal{T}$. Given two test types $type(e_i)$ and $type(e_j)$, we write $type(e_i)\preceq_{cat} type(e_j)$, if $type(e_i)$ is an abstraction of $type(e_j)$ according to their test attributes ta_k. Figure 1(b) shows an example of test hierarchies $\mathcal{H}_{cat}$.

Certificate. A certificate $C(p,m,e)$ generated as a result of our certification process includes a property p, a service model m, and a test evidence e. For simplicity (but with no lack of generality), we assume that the certification authority will release a different certificate for each m_{wsdl} (i.e., for each operation) in the service model. Security certificates $C(p,m,e)$ enable a procurement process where services are selected on the basis of their security certificates.

In the following, when clear from the context, we will refer to operations using the corresponding service name ws_i.

3 BPEL-BASED SERVICE COMPOSITION

Our reference scenario is a service-based environment where services are composed according to functional and non-functional (i.e., certified security properties) requirements. It includes the following parties: *i)* a *certification authority* that certifies security properties of services; *ii)* a *service provider* that implements and distributes certified services; *iii)* a *process owner*, the client of our approach, that implements a business process by selecting and composing certified services;[3] *iv)* a *service registry* that publishes a set of certified services (e.g., implemented as Eureka registry in case of RESTful services); *v)* a *customer* that accesses a business process.

We use BPEL as the language for describing executable composite services. BPEL is an XML-based language that permits to implement a business process using a collection of services, specifying the order in which service operations are invoked, the data to be exchanged at each step of the composition, and the (functional) conditions under which a service is selected and integrated within the business process [4]. BPEL defines executable processes that consist of a set of activities (e.g., *invoke*, *receive*, and *reply*) combined using different structures to form a coherent system. Most BPEL engine supports runtime service composition, where component services within registries are dynamically selected and composed on the basis of functional requirements [68]. We formally model a BPEL service composition as a graph as follows.

Definition 3.1 (BPEL graph). A BPEL graph $G(V,E)$ is a direct acyclic graph having a root $v_r \in V$, a vertex $v_i \in V_I \subseteq V$ for each service operation invocation, two additional vertices $v_c, v_m \in V_\otimes \subset V$ for

[3] We note that the process owner can be a developer aiming to design a static composition of services with certified security properties.

Table 1. eFlight component services

Service	Operation	Description
Airline (AI)	*flightDetails* `flightOffer`(*query*) *orderID* `bookFlight`(*flightID*,*cc*)	Allows customers to browse available flights Allows customers to book a flight using their credit card
Payment Service (PS)	*paymentID* `payOrder`(*orderID*,*cc*)	Allows customers to pay for a transaction
Secure Storage (SS)	*success* `write`(*data*)	Stores data in a remote server

each alternative ($\otimes$) structure modeling the alternative execution (*choice*) of operations and the retrieval (*merge*) of the results, respectively, and two additional vertices $v_f, v_j \in V_{\oplus} \subset V$ for each parallel ($\oplus$) structure modeling the contemporary execution (*fork*) of operations and the integration (*join*) of their results, respectively.

We note that $\{v_r\} \cup V_I \cup V_{\otimes} \cup V_{\oplus} = V$, and v_c, v_m, v_f, and v_j model branching for alternative/parallel structures. We also note that root v_r represents the BPEL orchestrator, that is, the set of operations exposed in the WSDL by the process owner. For simplicity but with no lack of generality, we assume the BPEL orchestrator and its model to just include external service invocations, with no internal functionalities implemented using script languages that need to be certified. In fact, any internal BPEL functionality can be also executed by an external service, thus introducing no real limitations to the applicability of our approach.

As an example, we shall consider a relatively simple BPEL-based business process implementing a flight reservation service (*eFlight*), which allows customers to browse and compare different offers, book a flight, and pay for it over the Internet. The eFlight process owner acts as an orchestrator and composes a set of partner services, provided by different service providers, to implement eFlight. In particular, it relies on n airline services selling flight tickets, m payment services to pay for the selected flights, and an external storage service to store customer and flight information. Table 1 presents the operations of each component service in detail. When a request to book a flight is sent to eFlight, a call to operation `flightOffer` of all airline services is made. The result from each airline is reported to the customer in a tabular form, to enable comparison. The customer can then book and pay for the preferred flight at eFlight, using one of the available payment services. In this case, eFlight first invokes operation `bookFlight` of the airline service providing the selected offer, and then operation `payOrder` of the selected payment service. For each purchase, eFlight stores the transaction data using operation `write` of the secure storage service. Figure 2 shows the BPEL graph for the eFlight service with a single payment service *PS*.

In the following, we use eFlight to clarify concepts and definitions in the paper. Then, in Section 8, we test the utility and generality of our approach in a real-world industrial scenario, which considers the PaymentService of eFlight extended on the basis of Engineering ENGpay system for online payments.

4 VIRTUAL CERTIFICATE OF COMPOSITE SERVICES

The core idea of our approach is to empower certification authorities to virtually certify a composite service starting from the certificates of the component services. Composite service certificates are "virtual", since they do not involve any real testing. The trust in a virtual certificate is mainly based on the trust a customer has in the automatic process implemented by the certification authority and in the certificates held by the individual component services. These conditions are however necessary but not sufficient conditions to implement a trust relation in a virtual certificate, and are coupled with an additional assumption inspired by the Zermelo's theorem (a.k.a., well-ordering assumption), which is equivalent to an axiom of choice, as follows.

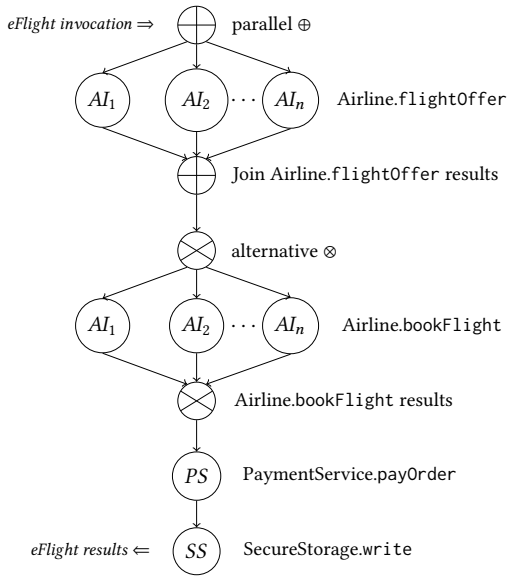


Fig. 2. eFlight BPEL graph

Assumption 4.1. *Given a set of virtual certificates (including the empty certificate), there exists a well-ordering with its domain modeling the user's trust in certificates.*

The well-ordering in assumption 4.1 models a classic behavior of human beings that prefer to have a certificate signed by a trusted organization rather than no certificate at all. The well-ordering approach implemented in this paper is presented in Section 6.

A virtual certificate is a certificate denoted as $C^*_{ij}(p^*_{ij}, m^*_{ij}, e^*_{ij})$ where the virtual property p^*_{ij}, model m^*_{ij}, and evidence e^*_{ij} are generated using an algebra for certificate composition. In the following we describe how to generate the virtual property, model and evidence of C^*_{ij} by applying our algebra operators between a couple of services ws_i and ws_j having certificates C_i and C_j, respectively.

4.1 Algebra for certificate composition

Our algebra of certificates mimics the algebra of service composition, meaning that each composition of two services ws_i and ws_j triggers the application of the corresponding operator in the algebra of certificates. We consider five main operators, namely, sequence $\odot$, alternative $\otimes$, parallel $\oplus$, loop $\oslash$, containment τ. The syntax is defined according to the following BNF-like notation.

$$C \ ::= \ \epsilon \mid C \mid \ \oslash C \mid C \odot C \mid C \otimes C \mid C \oplus C \mid C \tau C$$

C represents a security certificate, while ϵ is an empty certificate. The above notation shows how a certificate of a given composite service can be generated by iteratively applying the algebra operators. Our syntax is disambiguated by defining precedence and associativity rules for each operator. In particular, operators $\oslash$, $\odot$, $\otimes$, and $\oplus$ are left associative, while τ is non-associative. The precedence is defined as follows: *i)* τ has precedence 0; *ii)* $\otimes$ and $\oplus$ have precedence 1; *iii)* $\oslash$ and $\odot$ have precedence 2.

Before discussing the semantics of the algebra and its operators, we introduce the concept of environment to describe relations in a domain of interest by means of clauses. Our environment is defined as follows.

Definition 4.1 (Environment). An environment *env* is a set of clauses of two types: *i)* facts f that hold in a given domain of interest and *ii)* boolean rules over certificates C and facts f that specify constraints for certificate composition.

We note that *facts* model domain-specific information that can be used in rule definition, while *rules* define constraints on the composition of certificates. In other words, facts describe true assertions about the BPEL process (e.g., all component services share the same authorization database) and on the component services (e.g., data are stored in an encrypted form and distributed as they are), while rules define restrictions such as *if property p_i of C_i and p_j of C_j are composed using operator $\oplus$, then property p_{ij} holds.*

Composite certificates can be generated by composing certificates through the algebra operators as follows.

Sequence $\odot$. It composes two certificates C_i of ws_i and C_j of ws_j in a sequence. $[C_i \odot C_j]_{env}$ mimics a composition where ws_j is executed after ws_i.

Alternative $\otimes$. It composes two certificates C_i of ws_i and C_j of ws_j in an alternative. $[C_i \otimes C_j]_{env}$ mimics a composition where either ws_i or ws_j are executed.

Parallel $\oplus$. It composes two certificates C_i of ws_i and C_j of ws_j in a parallel. $[C_i \oplus C_j]_{env}$ mimics a composition where both ws_i and ws_j are executed in parallel.

Loop $\oslash$. It composes two certificates by iteratively composing the same certificate C. $[\oslash C]_{env}$ mimics a composition where the same ws is executed a given number of times. In the following, $\oslash$ is considered as a sequence of $\oslash$ services with the same certificate C.

Containment τ. It composes two certificates C_i of ws_i and C_j of ws_j in a containment relation. $[C_i \tau C_j]_{env}$ mimics a basic composition pattern where ws_j is called within ws_i, meaning that ws_i assumes the role of container and ws_j uses container-level functionalities (e.g., signature, encryption) to secure the message exchange.[4] We note that operator containment has not a direct map to BPEL constructs because it is applied to a specific service before BPEL process is even considered.

4.2 Virtual Properties

The virtual property p^*_{ij} in C^*_{ij} depends on the properties p_i and p_j in the component service certificates C_i and C_j, respectively, and operator $op \in \{\odot, \otimes, \oplus, \tau\}$ used to compose ws_i and ws_j. Property composition is not restricted to a specific class of properties, but it depends on the behavior the specific property assumes in a given domain. To this aim, our approach to virtual property generation is based on a set of ad hoc rules modeling expectations in a specific domain and a default rule settling scenarios in which no ad hoc rules apply to the composition of two properties. These rules identify all pairs of properties that can be composed.

More in detail, a virtual property is generated by first applying a set of *ad hoc* rules in the form $[p_k \; op \; p_t]_{cond} = p^*_{kt}$, where $p_k = (\hat{p}_k, \{a_{k,1}, \ldots, a_{k,n}\})$, $p_t = (\hat{p}_t, \{a_{t,1}, \ldots, a_{t,n}\})$, $p^*_{kt} = (\hat{p}^*_{kt}, \{a_{kt,1}, \ldots, a_{kt,n}\})$, and *cond* is the set of conditions (verified using facts in the algebra in Section 4.1) under which the rule is applicable. We note that each property in the rule must define the entire set of possible attributes *Attr*; value *null* is assigned to non-specified attributes. We also note that each

[4] A service container is either a middleware or a PaaS service providing functionalities (i.e. security features like WS-Security [71]) to manage the life cycle of services and the runtime infrastructure to support service specifications [13, 21].

attribute $a_{kt,i}$ in p^*_{kt} is computed as a function f_i of corresponding attributes $a_{k,i}$ and $a_{t,i}$ (i.e., $a_{kt,i}$=$f_i(a_{k,i},a_{t,i})$), or by directly assigning a value val to it (i.e., $a_{kt,i}$=val). Our ad hoc rules are defined by trustworthy experts and represent the behavior of security properties when the services holding them are integrated in a composition via a given composition operator.

If no ad hoc rules apply for the composition of p_i and p_j, a *default* rule is then triggered. The default rule considers property hierarchy $\mathcal{H}_P$, and is such that given p_i and p_j the resulting virtual property p^*_{ij} is the *least upper bound* (lub) in $\mathcal{H}_P$. Virtual property p^*_{ij} is calculated according to the following definition.

Definition 4.2 (Composite properties p^*_{ij}). Given $p_i,p_j\in\mathcal{P}$ to be composed using operator $op\in\{\odot,\otimes,\oplus,\tau\}$, the virtual property $p^*_{ij}\in\mathcal{P}$ is computed as follows:

(1) if an ad hoc rule $[p_k\ op\ p_t]_{cond}$=p^*_{kt} is applicable, that is, *cond* is satisfied by the facts in Definition 4.1, $p_k\preceq_P p_i$, and $p_t\preceq_P p_j$, p^*_{ij} is calculated according to it and is such that $p^*_{kt}\preceq_P p^*_{ij}$;
(2) else, p^*_{ij} is the *lub* of p_i and p_j in $\mathcal{H}_P$, such that *i)* $p^*_{ij}\preceq_P p_i$ and $p^*_{ij}\preceq_P p_j$, and *ii)* $\forall p\in\mathcal{P}: p\preceq_P p_i$ and $p\preceq_P p_j \Rightarrow p\preceq_P p^*_{ij}$.

Condition 1 states that in case an ad hoc rule is applicable, at least p^*_{kt} is guaranteed as the virtual property. Condition 2 instead applies whenever Condition 1 fails and provides a virtual property based on *lub* that is always weaker or at most equal to p_1 and p_2. In particular, if $p_1\preceq_P p_2$ ($p_2\preceq_P p_1$, resp.) p^*_{12}=p_1 (p^*_{12}=p_2, resp.). We note that, in case multiple rules apply or multiple *lub* exist, different properties are virtually certified for the service.

In this paper, we focus on Confidentiality, Integrity, Authentication (CIA) properties, which are at the basis of a security certification process. Figure 1 shows some examples of CIA properties and their distribution in a property hierarchy. We note that other security properties can be defined by composing CIA properties, such as privacy and non-repudiation. We also note that our approach, being generic, can address composition of any classes of properties, including properties (e.g., trust) whose composition usually requires more complex algebraic structure than the ones in Section 4.1. New algebraic structures can in fact be modeled within ad hoc rules defining the expectations on the composition process, in a specific domain and for a specific property.

Example 4.3 (Composition of p_i and p_j using an ad hoc rule – 1). Let us assume $Attr$={`algo`, `key`, `ctx`} as the set of all attributes $Attr$. Let us then consider ad hoc rule $[p_i\odot p_j]_{cond}$=p^*_{ij}, where p_i=(*Confidentiality*, {`algo`=null, `key`=null, `ctx`=in transit-in storage}), p_j=*Any* meaning that any property is allowed, `cond`=encrypted-disclosure meaning that stored information is always disclosed in encrypted form, and p^*_{ij}=(*Confidentiality*, {`algo`=$val(a_{i,algo})$, `key`=$val(a_{i,key})$, `ctx`=in transit-in storage}), with val denoting the function that returns as output the value of the property attribute given to it as input. Virtual property p^*_{ij} is awarded to the corresponding composite service $ws_i\odot ws_j$, since all information between ws_i and ws_j will be transmitted and stored in an encrypted form. Then, if we consider a service with property p_1=(*Confidentiality*,{`algo`=3*DES*,`key`=168bit,`ctx`=in transit-in storage) (i.e., $p_i\preceq_P p_1$) and assume that *cond* holds based on facts in Definition 4.1, the ad hoc rule is applied and virtual property p^*_{12}=(*Confidentiality*,{`algo`=3*DES*,`key`=168bit,`ctx`=in transit-in storage}) generated for $ws_i\odot ws_j$. We note that p^*_{12} is such that $p^*_{ij}\preceq_P p^*_{12}$.

Example 4.4 (Composition of p_i and p_j using an ad hoc rule – 2). Let us consider ad hoc rule $[p_i\tau p_j]_{cond}$=p^*_{ij}, where p_i=(*Authenticity*,{`algo`=null,`key`=null,`ctx`=null}),p_j=(*Integrity*, {`algo`=null, `key`=null, `ctx`=null}), `cond`=action-identity-link meaning that a link between each action and the identity of the user executing the action is kept and cannot be falsified, and p^*_{ij}=(*Non repudiation*, `ctx`=signature-based-integrity}). We note that `cond` restricts access to the private key of the user only upon authentication. Then, if we consider a container with property p_1=(*Authenticity*,{`algo`=biometric,

key=null, ctx=null}), a service with property p_2=(*Integrity*,{algo=RSA,key=null,ctx=envelop}) and assume that *cond* holds based on facts in Definition 4.1, the ad hoc rule is applied and virtual property p^*_{12}=(*Non repudiation*,ctx=signature-based-integrity}) generated for $ws_i \odot ws_j$. Focusing on eFlight and, in particular, on service *PS* in Table 1, property *non repudiation* is generated iff, according to condition cond, an authentication based on biometric techniques is implemented in the container, a privacy key is retrieved after authentication, and *PS* signs all payment transactions with the retrieved key.

Example 4.5 (Composition of p_i and p_j using default rule). Let us consider properties p_1=(*Confidentiality*, {algo=DES, key=112bit, ctx=in storage}) and p_2=(*Confidentiality*, {algo=AES, key=128, ctx=in storage}) of services *PS* and *SS* in Table 1 to be composed in a sequence. Suppose that there are no ad hoc rules for $p_1 \odot p_2$. Then, the default rule is applied, and p^*_{12}=(*Confidentiality*, {algo=encryption, ctx=in storage}) generated using $\mathcal{H}_P$ in Figure 1(a).

4.3 Virtual Model

The virtual model m^*_{ij} in C^*_{ij} depends on models m_i and m_j in the component service certificates C_i and C_j, respectively, and operator $op \in \{\odot, \otimes, \oplus, \tau\}$ used to compose ws_i and ws_j. We note that the virtual model m^*_{ij} is generated by merging the STS of the services according to the following definition.

Definition 4.6 (m^*_{ij}). Given m_i and m_j, $m^*_{ij}=m_i\ op\ m_j$ is an STS $\langle \mathcal{S}_{ij}, s_{ij}, \mathcal{V}_i \cup \mathcal{V}_j, \mathcal{I}_i \cup \mathcal{I}_j, \mathcal{A}_i \cup \mathcal{A}_j, \rightarrow_i \cup \rightarrow_j \rangle$, where:

(1) if $op=\odot$, $\mathcal{S}_{ij}=\mathcal{S}_i \cup \mathcal{S}_j$, where the last vertex in the interface part of m_i and the first of m_j are substituted by a single vertex;
(2) if $op=\otimes$, $\mathcal{S}_{ij}=\mathcal{S}_i \cup \mathcal{S}_j$, where the first vertex in the interface part of m_i and m_j are substituted by a single vertex;
(3) if $op=\oplus$, $\mathcal{S}_{ij}=\mathcal{S}_i \cup \mathcal{S}_j$, where the first vertex in the interface part of m_i and m_j are substituted by a single vertex. The same applies to the last vertex of m_i and m_j.
(4) if $op=\tau$, the model of the container m_i does not have any direct impact on the composite model, which is the same as m_j. Each state in m_j relying on functionalities of container ws_i (see [13] for further details) is graphically represented as a double circle.

We remark that Definition 4.6 considers integration of connected components m in interface-based models, that is, models with three states of the interface. For composition of more complex models, there exist scenarios in which m_i or m_j has more than one last vertex (i.e., leaf nodes). In this case, the process in Definition 4.6 is applied for each leaf node. We also note that $\mathcal{S}_{ij}$ can contain the vertices of the service operation implementation, if available.

Example 4.7. Figure 3 shows an example of virtual model generation for operators $\odot$, $\otimes$, $\oplus$ and τ. For this example, we assume two three-state STSs with no model of the implementation for services ws_1 and ws_2, where $\mathcal{S}_1=\{s_1,s_2,s_3\}$ and $\mathcal{S}_2=\{s'_1,s'_2,s'_3\}$. Figure 3(a) shows the resulting model for $ws_1 \odot ws_2$, where states s_3 and s'_1 are integrated in s^*; Figure 3(b) shows the model for $ws_1 \otimes ws_2$, where states s_1 and s'_1 are integrated in s^*; Figure 3(c) shows the model for $ws_1 \oplus ws_2$, where states s_1 and s'_1 are integrated in s^*_1, and s_3 and s'_3 in s^*_3; Figure 3(d) shows the model for $ws_1\, \tau\, ws_2$, with ws_1 the service container, where $\mathcal{S}_{12}=\mathcal{S}_2$ and states in $\mathcal{S}_{12}$ that relies on functionality of ws_1 are denoted with double circles. The input actions are denoted with $?f(in)$, while the corresponding output actions are denoted with $!f(out)$. Guards specifying conditions on transitions can also be associated with each action.

Example 4.8. As an example of virtual model generation for complex service models, let us assume that the models in Figure 3(b) and Figure 3(a) need to be composed in a sequence. Following

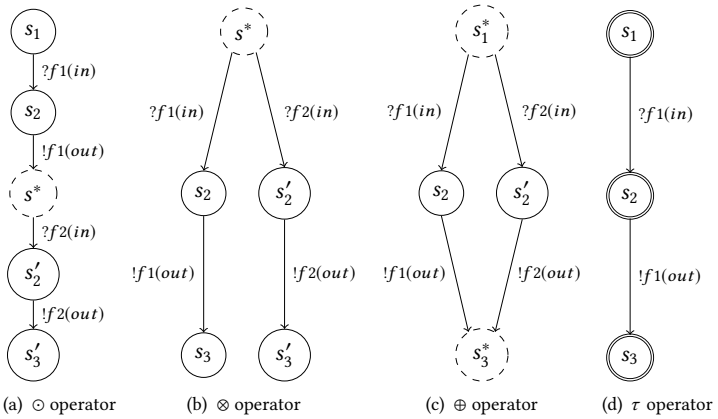


Fig. 3. An example of virtual service models

our discussion, the merging process is applied for each leaf; in particular, the model in Figure 3(a) is first attached to node s_3 in Figure 3(b) (i.e., states s_3 in Figure 3(b) and s_1 in Figure 3(a) are integrated in a single node) and is then attached to node s'_3 in Figure 3(b) (i.e., states s'_3 in Figure 3(b) and s_1 in Figure 3(a) are integrated in a single node).

4.4 Virtual Evidence

Virtual evidence e^*_{ij} is the set $\{\langle cat(e^*_{ij}), type(e^*_{ij}), ta(e^*_{ij}), tc(e^*_{ij}), tr(e^*_{ij})\rangle\}$ in C^*_{ij}, where $cat(e^*_{ij})$ is a virtual category, $type(e^*_{ij})$ a virtual type, $ta(e^*_{ij})$ a set of test attributes, $tc(e^*_{ij})$ a set of test cases, and $tr(e^*_{ij})$ the results of test case execution. It depends on the evidence e_i and e_j in the component service certificates C_i and C_j, respectively, and operator $op \in \{\odot, \otimes, \oplus, \tau\}$ used to compose ws_i and ws_j. We discuss virtual evidence generation on the basis of operators in $\{\odot, \otimes, \oplus, \tau\}$ assuming e_i and e_j to contain a single tuple $\langle cat(e), type(e), ta(e), tc(e), tr(e)\rangle$.

Operator $\odot$. Let us first consider a composition $ws_i \odot ws_j$. Virtual evidence is generated according to the concepts *equivalence classes* and *equivalence class relationship*. Equivalence classes are defined for all service inputs and use the test category to model the entire set of possible values, including: *i)* valid values in category functionality, *ii)* invalid values in category robustness, and *iii)* attack values in category penetration. We denote with $\mathcal{F}_j$ the set $\{\mathcal{F}_{j,x}\}=\{ec_{jx1},\ldots,ec_{jxn}\}$ of all equivalence classes (one for each input variable x) for service ws_j. An equivalence relationship $\approx_t$ is then defined on the domain D of test case outputs $f_i(\cdot)$ and corresponding test case inputs d_j. From $\approx_t$, we pass to the quotient $D/\approx_t$ of D over $\approx_t$ as the set of equivalence classes that form a partition $\mathcal{F}$ of D. Clearly, given D and $\mathcal{F}$, if $(f_i(\cdot),d_j) \in D \times D$, and $f_i(\cdot)$ and d_j belong to the same equivalence class, we write $f_i(\cdot) \approx_t d_j$.

We note that, although each output $f_i(\cdot)$ and corresponding input d_j take values in the same domain D, the equivalence classes logically referring to $f_i(\cdot)$ may differ from the ones of d_j. For instance, a token returned as output by a service login in case of success has a single equivalence class of valid values; a token given as input to a payment service has three equivalence classes, that is, the set of existing and valid tokens, the set of not existing and valid tokens, the set of reused tokens. In the following, when not mandatory, we will consider a 1-to-1 equivalence relation where the equivalence classes of $f_i(\cdot)$ and d_j are the same and have a direct mapping.

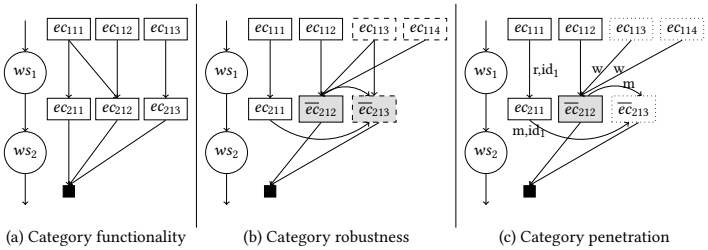


Fig. 4. Virtual evidence generation assuming a single input variable for both ws_1 and ws_2. Equivalence classes of category functionality are denoted with solid rectangles, category robustness with dashed rectangles, category penetration with dotted rectangles. The filled square represents the end of a BPEL flow, that is, the output of the last service invocation when virtual test cases are generated. Special equivalence classes, where test cases are generated before the BPEL end, are denoted with gray background. Arrows between equivalence classes represent the relationship between inputs/outputs of services.

When we consider a sequential service composition, a virtual test case $t^* \in tc(e^*_{ij})$ is generated if a sequence of test cases covering an entire path in the composition can be integrated. There are however some *special equivalence classes* (denoted $\overline{ec}_{jxk}$) in $\mathcal{F}_j$ that, when reached, trigger the generation of a virtual test case covering a subpath of the composition that starts from its root. In the following, when clear from the context, we denote test cases of the form $(\{d_i\},\{f_i(\cdot)\})$ using the corresponding equivalence classes $(\{ec_{i**}\},\{ec_{j**}\})$, with $ec_{i**} \in \mathcal{F}_i$ and $ec_{j**} \in \mathcal{F}_j$, and $*$ denoting any input variable and equivalence class. The certification authority is responsible for the definition of the equivalence classes of each service invocation in the composition. Indeed, a standard way of defining them increases trust in the virtual certification process and permits to provide a consistent comparison of virtual certificates associated with different functionally-equivalent services.

Given e_i and e_j, the virtual evidence e^*_{ij} is computed based on test categories as follows.

- *Category Functionality.* If $cat(e_i)=cat(e_j)=Functionality$, $cat(e^*_{ij})=Functionality$, and $type(e^*_{ij})$ is computed as the *lub* between $type(e_i)$ and $type(e_j)$ using $\mathcal{H}_{cat}$ in Figure 1(b). The set of virtual test cases $tc(e^*_{ij})$ is then generated by merging pairs of test cases $tc(e_i)$ and $tc(e_j)$ according to the following definition.

  Definition 4.9 ($tc(e^*_{ij})$). Given two test cases $t_i=(\{d_i\},\{f_i(\cdot)\}) \in tc(e_i)$ and $t_j=(\{d_j\},\{f_j(\cdot)\}) \in tc(e_j)$, a virtual test case $t^*_{ij} \in tc(e^*_{ij})$ is generated *iff* for each output $f_i(\cdot)$ of t_i having a corresponding input d_j of t_j, $f_i(\cdot) \approx_t d_j$, that is, they belong to the same input equivalence class of ws_j.

  We note that the calculation of $tc(e^*_{ij})$ applies to the Cartesian product $tc(e_i) \times tc(e_j)$. We also note that outputs $f_i(\cdot)$ of t_i and inputs d_j of t_j can be partially overlapped. Outputs having no corresponding input (and vice versa) are not consider in Definition 4.9. Figure 4(a) shows an example of equivalence classes for category functionality, where $\mathcal{F}_1=\{\{ec_{111},ec_{112},ec_{113}\}\}$ are the equivalence classes for the input of ws_1 and $\mathcal{F}_2=\{\{ec_{211},ec_{212},ec_{213}\}\}$ the ones for the input of ws_2. As an example, given the equivalence classes in Figure 4(a), a test case executed on ws_1 with input in ec_{111} and output in ec_{211} (denoted $t_1=(ec_{111},ec_{211})$) can be integrated with a test case case executed on ws_2 with input in ec_{211} and any output (denoted $t_2=(ec_{211},*)$) , generating a virtual test case t^*_{12} as the sequence of the above two test cases.
- *Category Robustness.* If $cat(e_i) \vee cat(e_j)=Robustness$ and $cat(e_i) \wedge cat(e_j) \neq Penetration$, $cat(e^*_{ij})=Robustness$. If both $cat(e_i)$ and $cat(e_j)$ have category robustness, the virtual test type $type(e^*_{ij})$ is calculated as the *lub* of $type(e_i)$ and $type(e_j)$ using $\mathcal{H}_{cat}$ in Figure 1(b). Otherwise, $type(e^*_{ij})$ assumes the type of the evidence with category robustness. Virtual test cases are then generated by

combining $tc(e_i)$ and $tc(e_j)$ as follows. Two test cases $t_i \in tc(e_i)$ and $t_j \in tc(e_j)$ form a test case $t^*_{ij} \in tc(e^*_{ij})$, if each output $f_i(\cdot)$ of t_i and corresponding input d_j of t_j are such that $f_i(\cdot) \approx_t d_j$ (Definition 4.9), and one of the following conditions are satisfied *i)* at least one input d_j of t_j belongs to an input partition in category robustness (e.g., invalid input), *ii)* at least an input of t_j (with no correspondence to outputs of t_i) belongs to an input partition in category robustness (e.g., invalid input), *iii)* at least an output $f_i(\cdot)$ of t_i belonging to category functionality is logically replaced by an output $f'_i(\cdot) \approx_t d_j$ belonging to an input partition in category robustness (e.g., invalid input). Condition *iii)* is similar to the *input mutation* testing approach adopted in software testing literature [5]; a mutation, denoted $\rightsquigarrow$, exists from each equivalence class of category functionality to each of category robustness. Condition *iii)* can be reformulated as: there exists a mutation $\rightsquigarrow$ such that $\big(f_i(\cdot) \rightsquigarrow f'_i(\cdot)\big) \approx_t d_j$. Figure 4(b) shows an example of equivalence classes for category robustness, where $\mathcal{F}_1$=$\{\{ec_{111}, ec_{112}, ec_{113}, ec_{114}\}\}$ are the equivalence classes for the input of ws_1 and $\mathcal{F}_2$=$\{\{ec_{211}, \overline{ec}_{212}, \overline{ec}_{213}\}\}$ the ones for the input of ws_2. Classes ec_{113}, ec_{114}, and $\overline{ec}_{213}$ refer to category robustness, and classes $\overline{ec}_{212}$ and $\overline{ec}_{213}$ trigger test case generation when reached. As an example, given the equivalence classes in Figure 4(b), a test case t_1=$(ec_{113}, \overline{ec}_{212})$ executed on ws_1 can be used either to generate a virtual test case t^*_1=t_1 or as a starting point for the generation of a test case following mutation $\overline{ec}_{212} \rightsquigarrow \overline{ec}_{213}$. In the latter case, if a test case t_2=$(\overline{ec}_{213}, *)$ executed on ws_2 exists, a virtual test case t^*_{12} is generated. We note that stress/load testing, which aims to verify the robustness of a service while varying load conditions, are based on test cases of category functionality sent at high rates.

- *Category Penetration.* If $cat(e_i) \vee cat(e_j)$=*Penetration*, $cat(e^*_{ij})$=*Penetration*. For test case generation, we use an approach similar to the one for category robustness. The only difference is in condition *iii)* where, in a penetration scenario, the mutation of an equivalence class in another one is driven by the set of capabilities c (e.g., read, write, modify) needed to an adversary to implement a given attack [13]. Condition *iii)* is then reformulated as there exists a mutation $\overset{c}{\rightsquigarrow}$ such that $(f_i(\cdot) \overset{c}{\rightsquigarrow} f'_i(\cdot)) \approx_t d_j$, with $f'_i(\cdot)$ an attack equivalence class (e.g., SQL injection), and the test type of t_j is stronger than capability c (i.e., $c \preceq_{cat} type(e_j)$). This condition must hold for at least one pair $f_i(\cdot)$ and d_j. We note that capability c is taken from test types in Figure 1(b), and virtual test type $type(e^*_{ij})$ is generated by taking $type(e_i) \cup type(e_j)$ on the basis of hierarchy $\mathcal{H}_{cat}$, where $type(e_i)$ or $type(e_j)$ are initialized to *null* if their category is not penetration. We note that, when a specific vulnerability can be exploited by a set of valid requests executed with temporal constraints, the certification authority must report them as ad hoc relations between equivalence classes. These relations are denoted with $\overset{c,id}{\rightsquigarrow}$ and link together two input partitions of either the same or different operations, to the aim of certifying attack *id*. Figure 4(c) shows an example of equivalence classes for category penetration, where $\mathcal{F}_1$=$\{\{ec_{111}, ec_{112}, ec_{113}, ec_{114}\}\}$ are the equivalence classes for the input of ws_1 and $\mathcal{F}_2$=$\{\{ec_{211}, \overline{ec}_{212}, \overline{ec}_{213}\}\}$ the ones for the input of ws_2. Classes ec_{113}, ec_{114}, and $\overline{ec}_{213}$ refer to category penetration, and classes $\overline{ec}_{212}$ and $\overline{ec}_{213}$ trigger test case generation when reached. As an example, given the equivalence classes in Figure 4(c), a test case t_1=$(ec_{113}, \overline{ec}_{212})$ executed on ws_1 can be used either to generate a virtual test case t^*_1=t_1 of type *write* or as a starting point for the generation of a test case following mutation $\overline{ec}_{212} \overset{m}{\rightsquigarrow} \overline{ec}_{213}$. In the latter case, if it exists a test case t_2=$(\overline{ec}_{213}, *)$ executed on ws_2, a virtual test case t^*_{12} of type *modify* is generated based on $\overline{ec}_{212} \overset{m}{\rightsquigarrow} \overline{ec}_{213}$. Finally, following mutation labeled with id_1, it is possible to generate a virtual test case for attack id_1.

We note that the virtual attributes $ta(e^*_{ij})$ are generated on the basis of test attributes $ta(e_i)$ and $ta(e_j)$, and test cases $tc(e^*_{ij})$. For instance, test attribute `cardinality` is equal to the number of test cases in $tc(e^*_{ij})$. We also note that, since the evidence is virtual and no test cases are really executed on the composite service, $tr(e^*_{ij})=\emptyset$. A virtual test case $t^*_{ij} \in tc(e^*_{ij})$ generated following a BPEL path composed of n invocations is a set $\{(\{ec_{1xk}\},\{ec_{2x'k'}\}),\ldots,(\{ec_{nx^nk^n}\},*)\}$, where each $(\{ec_{1xk}\},\{ec_{2x'k'}\})$ refers to a test case and wildcard symbol $*$ represents any equivalence class or, in other words, the end of the BPEL flow.

Example 4.10. We now provide an example for each of the categories discussed above.

Let us consider a sequence between `bookFlight` of AI_1 and `payOrder` of PS in Table 1. The output of `bookFlight` is the encryption of an alphanumeric string *orderID* of 16 characters for local flights and of 32 characters for international ones, which is then given as input to `payOrder`. The set of equivalence classes for *orderID* are the empty string, the set of encryption of 16-character *orderID*, the set of encryption of 32-character *orderID*, and the set of invalid *orderID*.

Suppose now that both services have been certified for category *Functionality*, and C_{AI} has test type *Input partitioning.Equivalence partitioning* and C_{PS} *Input partitioning.Random input*. The set of test cases for AI contains t_1 that returns a valid encryption of a 16-character *orderID* and t_2 that returns a valid encryption of a 32-character *orderID*; the set of test cases for PS contains t_3 and t_4 that take as input a valid encryption of a 16-character *orderID*. The resulting virtual evidence e^* belongs to category *Functionality* and type *Input partitioning.Random input*, and has two test cases t^*_{13} and t^*_{14}.

Then, suppose that: *i)* service AI has been certified for category *Functionality*, while service PS have been certified for category *Robustness*; *ii)* C_{AI} has test type *Input partitioning.Random input*, while C_{PS} has test type *Invalid.Equivalence partitioning*, that is, it has been tested against invalid input (invalid data format). The set of test cases for AI contains t_1 and t_2 that receive as input a valid *flightID* and return as output a valid *orderID*; the set of test cases for PS contains t_3 and t_4 that take as input invalid *orderID* (e.g., null, wrong charset) returning error. The resulting virtual evidence e^* belongs to category *Robustness* and type *Invalid.Equivalence partitioning*, and has four test cases t^*_{13}, t^*_{14}, t^*_{23}, t^*_{24}, where the valid output of t_1 and t_2 are substituted with the invalid input of t_3 and t_4.

Finally, suppose that equivalence classes for *orderID* include the class of SQL injection attacks. Also, suppose that there is a mutation labeled with capability *modify* (i.e., $\overset{m}{\rightsquigarrow}$) between the valid and SQL injection classes, meaning that an SQL Injection attack can be exploited by an attacker having modify privileges on message `payOrder` containing *orderID*. At this point, let us consider two test cases t_1 and t_2 for operation `bookFlight` of service AI returning two valid *orderID*, and a single test case t_3 for operation `payOrder` of service PS, with $type(t_3)=modify$, taking as input an *orderID* whose value is in the partition of SQL injection attacks. The resulting virtual evidence e^* belongs to category *Penetration* and type *Attack.Modify*, and has two test cases t^*_{13} and t^*_{23}.

Operators $\otimes$, $\oplus$, τ. Let us consider e_i and e_j to be composed using operators $\otimes$, $\oplus$, τ. The virtual evidence e^*_{ij} is calculated as follows:

- *Operators $\otimes$ and $\oplus$.* Given e_i and e_j to be composed, the virtual evidence $e^*_{ij}=e_i \cup e_j$.
- *Operator τ.* Operator τ is special-type operator. It is applied a priori before the BPEL process is evaluated and generates evidence following two distinct approaches.
  In the first approach, the contained service inherits the certificate C_i of the container. The virtual evidence e^*_{ij} is then generated by selecting relevant evidence e_i in C_i, according to the container functionality used by the service and restrictions defined in conditions *cond* of property p (Section 4.2). We note that, for this to work, evidence e_i refers to a prototype

service that is used for testing purposes only. For instance, this service is used to test the encryption and signature functionalities of the container. Concretely, let us consider a service to be certified for confidentiality in transit using AES. Given a container certified to support AES encryption algorithm within WS-Security specifications, the service specifies support for a WS-Policy requiring the AES-based encryption of data exchanged over the Internet. In this scenario, virtual evidence is produced for the composite service by including the test cases used by the container to certify the support for AES encryption algorithm within WS-Security specifications.
In the second approach, container evidence e_i and service evidence e_j are composed in a sequence (operator $\odot$) to produce the virtual evidence e^*_{ij}, that is, $e_i \tau e_j = e_i \odot e_j$. Concretely, let us consider a service to be certified for confidentiality in transit-in storage. Virtual evidence can be generated by composing in a sequence the evidence in a container certificate for property confidentiality in transit and the one in a service certificate for property confidentiality at rest.

5 CERTIFICATION-AWARE SERVICE SELECTION AND COMPOSITION

Our approach produces service compositions holding a specific security property starting from the certificates of the component services. Component services are first selected according to their certificates and then composed to certify the target security property. Certificates of component services are finally integrated using the operators in Section 4 to produce a virtual certificate for the entire composition.

In the following, starting from the BPEL graph in Definition 3.1, we describe: *i)* how a certification authority can annotate the BPEL graph to ensure a specific property and permit the consequent generation of a virtual certificate (BPEL template, Definition 5.1) and *ii)* how to select suitable candidate services to instantiate a runnable composition satisfying the BPEL template (BPEL instance, Definition 5.2).

5.1 BPEL Template

Given the BPEL graph in Definition 3.1, we use annotations [15, 94] to let certification authority specify requirements in terms of certificates for each component service, to ensure a specific virtually certifiable property for the composition. More specifically, we use annotations with two different objectives: *i)* security annotations on individual BPEL invocations $v_i \in V_I$ in the BPEL graph, as a way to obtain a composition having the desired security property, *ii)* test equivalence classes $\mathcal{F}_i$ on individual BPEL invocations $v_i \in V_I$ in the BPEL graph (Definition 3.1), as a way to support the virtual test case generation for the composition (see Section 4.4).[5] Our annotation process involves two labeling functions: *i)* a labeling function $\lambda: V_I \rightarrow L_S$ that associates a set of security requirements $\mathcal{R} \in L_S$ with each invocation $v_i \in V_I$, *ii)* a labeling function $\gamma: V_I \rightarrow L_F$ that associates a set $\mathcal{F}_i \in L_F$ of equivalence classes with each invocation $v_i \in V_I$.

We formally define a BPEL template as follows.

Definition 5.1 (BPEL template). Given a BPEL graph $G(V,E)$, a BPEL template $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$ is a direct acyclic graph with two labeling functions: *i)* λ that assigns a label $\lambda(v_i)$, corresponding to the security requirements to be satisfied by the service represented by v_i, for each vertex $v_i \in V_I$; *ii)* γ that assigns a label $\gamma(v_i)$, corresponding to the equivalence classes defined for each vertex $v_i \in V_I$.

[5] We note that, although not explicitly formalized, each invocation vertex in V_I is also annotated with a set of functional requirements, to be evaluated before security requirements are considered.

```
<annotation id="ann01" ref="payOrder">
    <property>
        <abs>Confidentiality</abs>
        <algo>AES</algo>
        <key>128 bit</key>
        <ctx>in storage</ctx>
    </property>
    <model>Interface</model>
    <evidence>
        <category>Functionality</functionality>
        <type>Input partitioning.Random Input</type>
        <attrs>
            <attr id="cardinality">100</attr>
        </attrs>
    </evidence>
</annotation>
```

(a)

```
<annotation id="ann02" ref="write">
    <property>
        <abs>Confidentiality</abs>
        <algo>AES</algo>
        <key>128 bit</key>
        <ctx>in storage</ctx>
    </property>
    <model>Interface</model>
    <evidence>
        <category>Functionality</functionality>
        <type>Input partitioning</type>
    </evidence>
</annotation>
```

(b)

Fig. 5. An example of security annotation for operations `payOrder` (a) and `write` (b) of eFlight BPEL graph in Figure 2

The security annotation models requirements in the form of $\mathcal{R}[p,m,e]$. We note that $\mathcal{R}.p$ specifies the least security property the service must hold according to hierarchy $\mathcal{H}_P$, $\mathcal{R}.m$ defines the requirements on the service model used to certify a service, and $\mathcal{R}.e$ specifies the requirements on the evidence supporting each property. Being defined by the certification authority, each security annotation in the BPEL template contains the least set of requirements for each corresponding invocation such that p holds. We note that different annotations can exist for a single BPEL template; in the following, for simplicity but with no lack of generality, we assume a single annotation. We also note that given the set of ad hoc rules, property hierarchy $\mathcal{H}_P$, and property p to be certified for the composition, $\lambda(v_i)$ can be computed automatically for each $v_i \in V_I$, such that p is supported. However the definition of an automatic BPEL template annotation approach is outside of the scope of this paper.

The equivalence class annotation models a set $\mathcal{F}_i$ of $\mathcal{F}_{i,x}=\{ec_{ix1},\ldots,ec_{ixn}\}$. We note that each $\mathcal{F}_{i,x}$ depends on the input parameter x and on the specific transition (v, v_i); in other words, different paths could lead to different $\mathcal{F}_{i,x}$ for the same parameter x.

We note that both security annotation $\lambda(v_i)$ and equivalence class annotation $\gamma(v_i)$ can be expressed as XML fragments. An example of XML-based security annotations $\lambda(v_i)$ is provided in Figure 5 for operation `payOrder` (Figure 5(a)) and for operation `write` (Figure 5(b)). The XML fragment for operation `payOrder` prescribes the integration of a service certified for property *Confidentiality* of data *in storage* with *AES* algorithm and at least key length equal to or greater than $128bit$, using a WSDL model. The selected service must have been tested with at least 100 test cases of category *Functionality* and type *Input partitioning.Random input*. Specific examples of $\gamma(v_i)$ annotations and corresponding equivalence classes $\mathcal{F}_i$ are provided in Figure 4 and Example 4.10.

5.2 BPEL Instance

Given the BPEL template we define our BPEL instantiation technique as a function that takes as input a BPEL template $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$ and different sets of candidate services, each satisfying the functional requirements of one service operation invocation, and returns as output a BPEL instance $G'(V', E, \gamma)$. In $G'(V', E, \gamma)$, every invocation $v_i \in V'_I$ contains a service instance, and every branching $v \in V_\otimes \cup V_\oplus$ is maintained as it is. We formally define our BPEL instance as follows.

Definition 5.2 (BPEL instance). Let $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$ be a BPEL template, a BPEL instance $G'(V', E, \gamma)$ is a direct acyclic graph, where $v_r = v'_r$, for each vertex $v \in V_\otimes \cup V_\oplus$ it exists a corresponding vertex

$v' \in V'_{\otimes} \cup V'_{\oplus}$, and for each $v_i \in V_I$ it exists a corresponding vertex $v'_i \in V'_I$ instantiated with a real service ws_i having certificate C_i, such that the following conditions hold:

(1) ws_i satisfies functional requirements in $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$;
(2) C_i satisfies $\lambda(v_i)$.

Condition 1 is needed to preserve the process functionality, as it simply states that each service ws_i in the BPEL instance must satisfy functional requirements associated with the corresponding invocation in the BPEL template. Condition 2 states that a service ws_i can be integrated within the BPEL instance *iff* its security certificate C_i satisfies the security requirements $\lambda(v_i)$ associated with the corresponding vertex v_i in the BPEL template. We note that, there are no conditions on equivalence classes $\gamma(v_i)$, since they are only used as a support for the computation of the virtual evidence of G'.

The BPEL instance in Definition 5.2 is generated by traversing the graph $G^{\lambda,\gamma}$ with a *breadth-first* search. The algorithm starts from the root vertex v_r which is added as root v'_r of G'. Root v_r in fact already contains the instance of the orchestrator. Then for each vertex $v \in V_{\otimes} \cup V_{\oplus}$, a corresponding vertex $v' \in V'_{\otimes} \cup V'_{\oplus}$ is generated. Finally, similarly to the work in [12], for each vertex $v_i \in V_I$, a two-step selection approach is applied as follows.

(1) *Matching algorithm:* It matches requirements $\mathcal{R}[p, m, e]$ against service certificates $C(p, m, e)$, and returns as output a set of compatible services whose certificates satisfy $\mathcal{R}$. Formally, let us consider a set WS_i of candidate services ws_j, each one having certificate C_j, and $\lambda(v_i)$ as the security requirements $\mathcal{R}_i$ for the invocation at vertex v_i in the BPEL template. Assuming that functional matching is successful for each $ws_j \in WS_i$, security requirements $\lambda(v_i)$ are considered. The matching process is successful if C_j satisfies $\lambda(v_i)$; otherwise, ws_j is discarded and not considered for selection. The matching algorithm returns a set $WS'_i \subseteq WS_i$ of compatible services, which represent the possible candidates for selection.
(2) *Comparison Algorithm:* Upon retrieving a set of compatible services, it produces a ranking of these services (partial order) according to their certificates, and identifies the best service that satisfies the security requirements. Formally, compatible services $ws_j \in WS'_i$ are ranked in a partial order on the basis of their certificate C_j. The best service ws_i is then selected and integrated in $v'_i \in V'_I$. There are many ways of generating this partial order as reported in [12]. We present the approach used in this paper in Section 6.

When all vertices $v \in V$ have been visited, G' contains a service instance for each operation invocation in $G^{\lambda,\gamma}$. G' represents the real composite service to be certified. We note that if the above selection approach satisfies $\lambda(v_i)$ for each $v_i \in V_I$ of $G^{\lambda,\gamma}$, the composite service will inherit at least property p used as the target for its definition, according to theorem 5.3.

THEOREM 5.3. *[BPEL instantiation] Given a BPEL Template $G^{\lambda,\gamma}(V,E,\lambda,\gamma)$ defined so that a property p is guaranteed for the composition, any BPEL instance $G'(V', E, \gamma)$ obtained using Definition 5.2 guarantees at least p.*

PROOF. We proof the theorem by induction.

Base case: Let us consider a simple BPEL Template $G_1^{\lambda,\gamma}(V, E, \lambda, \gamma)$, including one invocation $v_1 \in V_I$, defined to guarantee a target property p_t. According to Definition 5.2, $v'_1 \in V'_I$ is instantiated with a service ws_1, forming a BPEL instance G'_1, such that ws_1 satisfies functional requirements in $G_1^{\lambda,\gamma}$ and C_1 satisfies $\lambda(v_1)$, meaning that $\lambda(v_1).p = p_t \preceq_P C_1.p$.

Inductive step: Let us consider a BPEL Template $G_k^{\lambda,\gamma}$, including k invocations $v_1, \ldots, v_k \in V_I$ composed using operators $op \in \{\odot, \otimes, \oplus\}$ and annotated with properties $\lambda(v_1).p, \ldots, \lambda(v_k).p$, respectively. $G_k^{\lambda,\gamma}$ is defined to guarantee a target property p_t for the composition. If a BPEL instance G'_k

is generated by instantiating invocations $v_1, \ldots, v_k$ in $G_k^{\lambda,\gamma}$ with services $ws_1, \ldots, ws_k$ following Definition 5.2, thus guaranteeing at least p_t, then G'_{k+1} guarantees at least p_t when k+1 invocations $v_1, \ldots, v_k, v_{k+1} \in V_I$ are considered in $G_{k+1}^{\lambda,\gamma}$ and the additional invocation v_{k+1} contributes to the support of p_t.

Specifically, G'_k and ws_{k+1}, with properties $C^*_k.p$ and $C_{k+1}.p$, are integrated in the BPEL instance G'_{k+1} generated as the instantiation of the BPEL template $G_{k+1}^{\lambda,\gamma}$, with $p_t \preceq_P C_k.p$ and $\lambda(v_{k+1}).p \preceq_P C_{k+1}.p$.

According to Definition 4.2, property $C^*_{k+1}.p^*$ is calculated using either an ad hoc rule $[p_1\ op\ p_2]{=}p$, such that $p_1 \preceq_P p_t \preceq_P C_k.p$ and $p_2 \preceq_P \lambda(v_{k+1}).p \preceq_P C_{k+1}.p$, or hierarchy $\mathcal{H}_P$. In the first case, by definition, $p_t \preceq_P p^*$, while in the second case $p_t \preceq_P p^*$ because p_t=lub(p_t,$\lambda(v_{k+1}).p$) and p^*=lub($C_k.p$,$C_{k+1}.p$). □

Example 5.4 (BPEL instantiation). Let us consider a BPEL template $G^{\lambda,\gamma}$ for services *PS* and *SS* composed in a sequence, and annotated with the XML fragments in Figure 5. The BPEL template is defined by the certification authority to guarantee at least p=(*Confidentiality*,{`algo`=DES, `key`=128, `ctx`=in storage}) for the composition. Let us then consider a valid BPEL instance $G'(V', E, \gamma)$ integrating two service instances ws_j and ws_k, with certified properties $C_j.p$=(*Confidentiality*,{`algo`=DES, `key`=256, `ctx`=in storage}) and $C_k.p$=(*Confidentiality*,{`algo`=DES, `key`=256, `ctx`=in storage}), respectively. Following Definition 4.2, a virtual property p^*_{ik}=(*Confidentiality*,{`algo`=DES, `key`=256, `ctx`=in storage}) greater than property p ensured by the BPEL template is calculated using hierarchy $\mathcal{H}_P$. p^*_{jk} has in fact higher key length than p.

5.3 Virtual Certificate Generation

Virtual certification is a process owned by a certification authority, which receives as input a BPEL instance $G'(V',E)$ and iteratively composes pairs of (virtual) certificates C_i and C_j to generate a virtual certificate $C^*_{G'}$ for the composition. The virtual certification process proceeds as follows: *i)* all pairs of services $ws_i \odot ws_j$ in a sequence are selected, their certificate C_i and C_j composed in a virtual one C^*_{ij}, and their respective vertices $v_i, v_j \in V'$ substituted with a single vertex v_{ij} annotated with certificate C^*_{ij}; *ii)* when no service sequences are left, a pair of services in either an alternative ($ws_i \otimes ws_j$) or a parallel ($ws_i \oplus ws_j$) are composed, a virtual certificate C^*_{ij} generated, and their vertices $v_i, v_j \in V'$ substituted with a single vertex v_{ij} annotated with certificate C^*_{ij}; *iii)* the process is repeated until G' is reduced to a graph having a single vertex v with virtual certificate $C^*_{G'}$. $C^*_{G'}$ is finally awarded to G'. Given two certificates C_i and C_j, a virtual certificate C^*_{ij} is generated through a three-step approach which composes: *i)* the property of C_i and C_j (Section 4.2), *ii)* the model of C_i and C_j and, in particular, the model of the operations of ws_i and ws_j involved in the composition (Section 4.3), *iii)* the evidence of C_i and C_j (Section 4.4).

The trustworthiness of virtual certificate $C^*_{G'}$ is discussed in the following proposition.

Proposition 5.5 (Virtual certificate trustworthiness). *The trustworthiness of virtual certificate $C^*_{G'}$ relies on well-ordering assumption (assumption 4.1) and the trust a specific customer has on the correctness of the requirements specified by the certification authority in the BPEL template and on the BPEL instantiation process (theorem 5.3).*

We remark that as a result, a multi-step *chain of trust* is built, grounded in the certificates of the atomic services: anyone who trusts the authorities who signed the certificates of the component services and the authority computing the virtual certificate for the BPEL composition, will trust the virtual certificate itself; then, the virtual certificate awarded to the composition can be recursively used as a basis to virtually certify another orchestration where the composition is just a component.

6 HEURISTIC APPROACHES TO BPEL INSTANCE GENERATION

In the previous section we described our general approach based on matching and comparison, aimed to generate and virtually certify a BPEL instance (see Sections 5.2 and 5.3). Here we refine the approach to produce the BPEL instance with the "best" virtual certificate.

The first step in this direction is the definition of a metric of quality of the BPEL instance, which measures the quality of its virtual certificate. We propose to measure the certificate quality by evaluating the quality of its virtual evidence. The main reasons supporting this idea are as follows: *i)* our approach guarantees a minimum virtual property for each BPEL instance (see Theorem 5.3), *ii)* the virtual model is not used for test case generation, while test cases are generated from the ones of component service,[6] *iii)* virtual evidence is the only artifact evaluating the strength of the certified property.

Given a BPEL instance $G'(V', E, \gamma)$ and its virtual certificate $C^*_{G'}$, we define a *quality metric* $\theta(G')$ that evaluates the quality of $C^*_{G'}$. $\theta(G')$ measures the coverage of equivalence classes in G' using the test cases in $C^*_{G'}$ and is calculated as follows.

Definition 6.1 ($\theta(G')$). Given G' and $tc(e^*_{G'}) \in C^*_{G'}$, $\theta(G')$ is calculated as

$$\theta(G') = \sum_{\forall v'_i \in V'_I} |\{ec_{ixk} \in \gamma(v'_i) | \exists t \in tc(e^*_{G'}) \; s.t. \; (ec_{ixk}, *) \in t\}|, \tag{1}$$

where ec_{ixk} is the k-th equivalence class for the x-th input variable of ws_i instantiating vertex v'_i.

We note that in case a BPEL instance G' is composed of a single vertex v'_i instantiated with service ws_i, $\theta(G')$ is calculated as the number of input equivalence classes in $\gamma(v'_i)$ covered by test cases in C_i of ws_i.

We also define a general purpose quality metric $\Theta(G')$ as the normalization of $\theta(G')$. $\Theta(G')$ is computed as the ratio between $\theta(G')$ and the total number of equivalence classes in G' according to the following definition.

Definition 6.2 ($\Theta(G')$). Given G' and $\theta(G')$, $\Theta(G')$ is calculated as

$$\Theta(G') = \frac{\theta(G')}{\sum_{\forall v'_i \in V'_I} |\gamma(v'_i)|}, \tag{2}$$

where the numerator represents the equivalence classes covered by test cases in $C^*_{G'}$ and the denominator the number of equivalence classes in G'.

While $\theta(G')$ is used to generate the best BPEL instance, $\Theta(G')$ can be used to compare different functionally equivalent BPEL instances.

The problem of computing a BPEL instance G' that satisfies security annotations in the BPEL Template and maximizes the certification quality metric $\theta(G')$ can be formally defined as follows.

Problem 6.1. *(Best BPEL Instance) Given a BPEL Template $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$ and a set WS_i of candidate services for each invocation $v_i \in V_I \subseteq V$, find a maximum BPEL instance $G'(V', E, \gamma)$ that satisfies the following requirements:*

(1) *G' satisfies functional requirements in $G^{\lambda,\gamma}$ (Definition 5.2, Condition 1);*
(2) *G' satisfies security annotations in $G^{\lambda,\gamma}$ (Definition 5.2, Condition 2);*
(3) *$\nexists$ a BPEL instance G'' such that G'' satisfies $G^{\lambda,\gamma}$ and $\theta(G'') > \theta(G')$.*

[6] We remark that the assumption *the better the model, the higher the certification quality* [13] at the basis of certificate comparison in [12] does not hold for the certification of composite services.

A BPEL instance G' represents the best BPEL instance iff the replacement of a service instance associated with a given invocation $v'_i \in V'_I$ either violates functional requirements for the corresponding v_i in $G^{\lambda,\gamma}$ (Condition 1), violates $\lambda(v_i)$ in $G^{\lambda,\gamma}$ (Condition 2), or produces a lower $\theta(G')$ (Condition 3).

The Best BPEL Instance problem is NP-hard, as proven by the following theorem.

Theorem 6.3. *The Best BPEL Instance problem is NP-hard.*

Proof. The proof is a reduction from the NP-hard problem of the Knapsack, formulated as follows. *Given a set of items S={1,. . .,n}, where item i has size s_j and value val_i, and knapsack capacity Cp, find the subset $S' \subset S$ that maximizes the value of $\sum_{i \in S'} val_i$ given that $\sum_{i \in S'} s_i \leq Cp$, that is, it fits in a knapsack of size Cp.*

Given a BPEL template $G^{\lambda,\gamma}(V, E, \lambda, \gamma)$ and the set WS of available services, the correspondence between the Max-Instance problem and the Knapsack problem can be defined as follows.

First, we assume each service $ws_j \in WS$ to be composed of different operations all with the same certificate C_j with evidence of category functionality. In the following, for simplicity, we refer to an operation using its corresponding service ws_j; we also consider $G^{\lambda,\gamma}$ that consists of a sequence of n services and is annotated with the same set of equivalence classes for each invocation. Then, let $|V_I|$ be the knapsack capacity Cp. The set WS of services includes a service ws_j for each item in S, with size s_i=1. Each ws_j is a candidate service for at least one vertex $v_i \in V_I$ and its value depends on the service associated with the instance of the preceding vertex v'_{i-1} in G'. In particular, the value val_j of ws_j, if selected to instantiate $v'_i \in V'_I$ in G', is calculated as $val_j = \theta(v'_i) = |\theta(v'_{i-1} \odot v'_i)| - |\theta(v'_{i-1})|$, meaning that val_j for ws_j is calculated as the intersection between the equivalence classes $\gamma(v_i)$ covered by the inputs of test cases in C_j and the ones covered by the outputs of test cases in the certificate of the service selected for v'_{i-1}.[7] We note that val_j calculated for candidate services ws_j associated with root v_r of the BPEL graph is equal to $\theta(v'_r)$, that is, it is the number of input equivalence classes in $\gamma(v_r)$ covered by test cases in C_j of ws_j. We also assume that for each service the same set of equivalence classes are covered both in input and output; as a consequence, when a service is selected to instantiate two consecutive invocations, the whole set of equivalence classes are covered for the second invocation. We finally assume that for each possible combination of ws_{i-1} and ws_i in a sequence $(ws_{i-1} \odot ws_i)$, the produced value val_i for ws_i is less than/equal to val_i achieved when $ws_{i-1} = ws_i$.

Each vertex $v_i \in V_I$ in $G^{\lambda,\gamma}$ is associated with a subset of compatible services $WS_i \subseteq WS$, where every service in WS_i has a certificate C that satisfies $\lambda(v_i)$. Let us now consider an instance set $\mathcal{W} = \{ws_1, \ldots, ws_n\} \subseteq WS$, with $n \leq Cp$, where each service in $\mathcal{W}$ is used to instantiate at least one $v'_i \in V'_I$ in G'. If the selected set produces a BPEL instance G' with maximum quality $\theta(G') = \sum_{v'_i \in V'_I} val_i$, then it represents a solution S' for the corresponding knapsack problem. Since the integration of a new service does not increase the total number $\theta(G')$ of covered equivalence classes, we will be able to to find the minimum subset of services that represent a valid BPEL instance G' and therefore maximize the quality metric θ. We therefore start from a BPEL template $G^{\lambda,\gamma}$ and try to found an instance $\mathcal{W}$ of services in S, such that all invocation $v_i \in V_I$ are instantiated by services in $\mathcal{W}$ and related annotations $\lambda(v_i)$ satisfied. Let us consider an instance set $\mathcal{W} = \{ws_1\}$, with $ws_1 \in WS_1$. If the selected set is such that a valid G' is produced, then it represents a solution S' for the corresponding knapsack problem, where the instance set maximizes the value θ of G'. If a valid G' is not found, meaning that at least one instantiation of v_i fails, we consider a new instance set $\mathcal{W} = \{ws_1, ws_2\}$, with $ws_1 \in WS_1$ and $ws_2 \in WS_2$, that tries to find a set of at most two services (i.e., a valid solution

[7] The definition of service value val_j that depends on the preceding service is equivalent to the scenario in which different instances of ws_j (one for each preceding service) are available and have different θ.

```
INPUT
G^{λ,γ}(V, E, λ, γ): BPEL template
WS_i: set of candidate services
      for v_i∈V_I⊆V (i:=1,. . .,n)

OUTPUT
G'(V', E, γ): BPEL instance

MAIN_LOCAL
for each v_i∈V_I do
   v'_i:=best_candidate_local(WS_i);
return G'

BEST_CANDIDATE_LOCAL(WS)
best_candidate←ws_1∈WS;
for each ws_j∈WS do {
   v_tmp←ws_j;
   if θ(best_candidate)<θ(v_tmp)
     best_candidate:=v_tmp;
}
return best_candidate;

MAIN_SBS
v_r:=best_candidate_local(WS_r);
v_r.visited:=true;
instance_generation(children(v_r));
return G';

INSTANCE_GENERATION(V)
for each v_i∈V do
   if v_i.visited≠true {
     v'_i:=best_candidate_sbs(v_i,WS_i);
     v_i.visited:=true;
   }
for each v_i∈V do
   if |children(v_i)|>0
     instance_generation(children(v_i));

BEST_CANDIDATE_SBS(v,WS)
if |parent(v)|>1
  v'_a:=∪_{∀v_i∈parent(v)} v'_i;
else if |parent(v)|=1
       v'_a:=v' s.t. v' is the instance
                    of parent(v);
best_candidate←ws_1∈WS;
for each ws_j∈WS do {
   v_tmp←ws_j;
   if θ(parent(v'_a) ⊙ best_candidate)
     <θ(parent(v'_a) ⊙ v_tmp)
     best_candidate:=v_tmp;
}
return best_candidate;
```

Fig. 6. Local Heuristic and Step-By-Step Heuristic

could include different operations of the same service) that covers the BPEL instance maximizing its quality. The number of terms in the instance set is iteratively increased by one until a solution to the problem is found or $|\mathcal{W}|=Cp$. □

We now introduce two heuristics, *local* and *step-by-step*, balancing quality and efficiency of the solution, which differs in the algorithm used for service selection. According to Problem 6.1, both heuristics receive as input a BPEL template $G^{\lambda,\gamma}(V,E,\lambda,\gamma)$ and a set WS_i of candidate services for each invocation $v_i \in V_I \subseteq V$, and returns as output a BPEL instance $G'(V',E,\gamma)$. The *local* heuristic selects for each invocation $v_i \in V_I$ the best service ws_i such that $\theta(v'_i)$ is maximum among all candidate services. The *step-by-step* heuristic first selects a candidate service ws_r for root vertex v'_r of the BPEL instance such that $\theta(v'_r)$ is maximum among candidate services;[8] then it traverses $G^{\lambda,\gamma}$ using a *breadth-first* search. At each invocation $v_i \in V_I$, a service ws_i is selected such that it maximizes the quality of the subgraph of G' that has been already traversed. We now describe in detail our heuristic algorithms, which are shown in Figure 6.

6.1 Local Heuristic

Local heuristic selects each service to be integrated in G', independently. Let us consider WS_i as the set of candidate services for vertex $v_i \in V_I$ of BPEL Template $G^{\lambda,\gamma}$. The local approach can be described as follows. We iteratively consider all invocation vertices $v_i \in V_I$ (**for each** cycle in function **main_local**) and instantiate (denoted with $\leftarrow$) each of them using $ws_i \in WS_i$ such that $\theta(v'_i)$ is maximized (function **best_candidate_local**). In case more than one service have maximum $\theta(v'_i)$, the first one is selected.

6.2 Step-By-Step Heuristic

The step-by-step heuristic first starts from root vertex v_r of the BPEL and selects ws_r such that $\theta(v'_r)$ is maximized (first instruction in function **main_sbs**). The rest of the BPEL graph is then traversed using a *breadth-first* search (function **instance_generation**) to select services for all

[8] We remark that, in a general case, the root vertex has only one candidate that represents the BPEL orchestrator.

invocations as follows. For each $v_i \in V_I$, candidate services $ws_j \in WS_i$ are evaluated with respect to the services already selected for v_i's parents (function **best_candidate_sbs**) as follows:

- In case v_i has only one parent v in $G^{\lambda,\gamma}$ ($|parent(v)|=1$), we consider composition $ws \odot ws_j$ where ws is the service selected for the parent's vertex v'_a in G' and $ws_j \in WS_i$ is one candidate service for vertex v'_i in G'. The best candidate service $ws_i \in WS_i$ is then selected such that $\theta(v'_a \odot v'_i)$ is maximized, and corresponding node v_i in the BPEL template is marked as *visited*.
- In case v_i has multiple parents in $G^{\lambda,\gamma}$ ($|parent(v)|>1$), first the corresponding vertices in G' are logically collapsed into a single parent vertex v'_a, with a single logical service ws_a and virtual certificate C^*_a, where the virtual test cases computed for each parent are merged in a single set of virtual test cases $tc(e^*_a)$. We then consider composition $ws_a \odot ws_j$, where $ws_j \in WS_i$ is one candidate service for vertex v'_i. The best candidate service $ws_i \in WS_i$ is selected such that $\theta(v'_a \odot v'_i)$ is maximized and corresponding node v_i in the BPEL template is marked as *visited*.

Finally, a third heuristic (*hybrid heuristic*) is defined as an hybridization of the two heuristics described in this section. The hybrid heuristic takes as input the services selected by local and step-by-step heuristics in their executions, and applies an exhaustive search of the optimal solution over them.

An overview of our experimental results is presented in Section 7, where performance and quality of the three heuristics are discussed and compared.

7 EXPERIMENTAL EVALUATION

We experimentally evaluated our certification approach for composite services. In the following of this section, we first present the testing infrastructure adopted in our experiments (Section 7.1). We then discuss the performance (execution time) of our heuristic algorithms and of the whole certification process (Section 7.2). We finally evaluate the quality of the virtual certificates generated according to our heuristics based on *quality metric* Θ (Section 7.3).

7.1 Testing Infrastructure

We designed and developed a Java-based testing infrastructure that implements the certification-aware service composition approach in Section 5, by extending our prototype described in [7] with the heuristics for the generation of virtual certificates. Our infrastructure implements local, step-by-step, and hybrid heuristics to assess their effectiveness and efficiency, in terms of both quality Θ of the generated virtual certificate (see Definition 6.2) and the execution time performance for its computation. To better evaluate the performance and quality of our heuristic algorithms, it also implements the exhaustive algorithm that computes the optimum solution by calculating all possible certificates given the BPEL template and all candidate services (Problem 6.1). Finally, to support complete and automatic experiments, the testing infrastructure provides *i)* a *BPEL template generator* that given a configuration file produces a BPEL template and *ii)* a *certificate generator* that, according to a BPEL template, generates a set of security certificates $C(p,m,e)$ for each invocation $v_i \in V_I$. We note that each certificate is logically associated with a service. We also note that each pair of service/certificate is a candidate for v_i, that is, it satisfies conditions (1) and (2) in Definition 5.2. BPEL template and certificate generators work as follows.

BPEL template generator produces BPEL templates $G^{\lambda,\gamma}$ with a sequential structure (worst case scenario) annotated with security requirements λ and equivalence classes γ. It receives as input a configuration file specifying an interval on the number of invocations ($|V_I|$) and an interval on the number of equivalence classes for each invocation. We note that the number of input parameters, requirements on attack equivalence classes, and special equivalence classes have less impact on

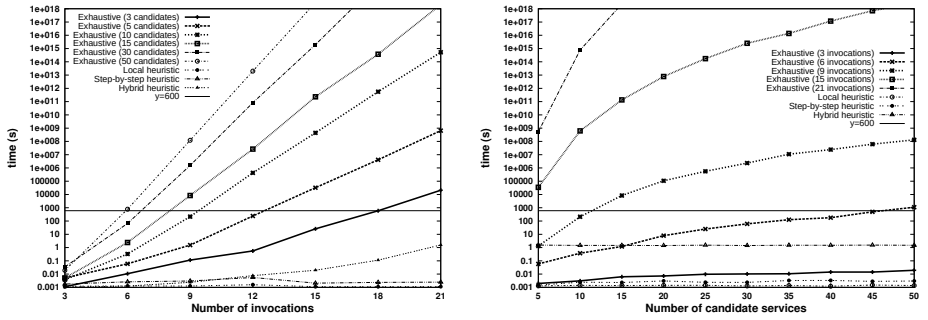


Fig. 7. Execution time (log scale) varying (a) the number of invocations and (b) the number of candidate services

our experiments and therefore are randomly generated. It then selects a random number n of invocations, and generates the corresponding nodes v_i with random security annotation $\lambda(v_i)$ and equivalence classes annotation $\gamma(v_i)$ for v_i according to the configuration file. The equivalence classes of each node v_i are randomly connected to one or more classes of child node v_{i+1}, with i=1,. . .,n−1, to represent the relationship between inputs/outputs of services (see Figure 4).

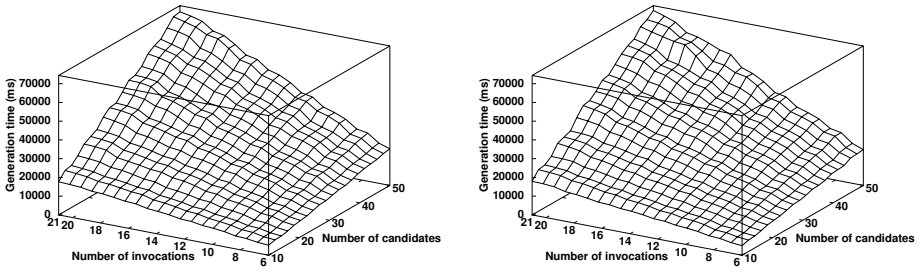
Certificate generator considers each invocation v_i in $G^{\lambda,\gamma}$ independently and produces a set of certificates, each referring to a service, that address the security annotation $\lambda(v_i)$. In particular, for each certificate, it first generates a property p and a model m that satisfy $\lambda(v_i)$. It then provides an evidence e whose category and type satisfy $\lambda(v_i)$, and with a random set of test cases generated as discussed in Section 4.4.

All our experiments have been run on a workstation equipped with an Intel Core 2 Duo CPU 2.4 GHz, 4GB RAM, 1TB disk, 7200RPM, 32MB cache, and Mac OS X 10.7.4, and installing the complete testing infrastructure. All experiments have been repeated 10 times and the results shown here are the average over the 10 executions.

7.2 Performance Evaluation

We evaluated the performance of our approach on different system configurations, varying the number of invocation nodes between 3 and 21, and the number of candidate services for each invocation between 5 and 50. For all configurations, we considered the hierarchies in Figure 1, 20 ad hoc rules, a rate of 100% of candidates (worst case) that satisfy the security annotation of the corresponding invocation, and a number of input equivalence classes varying between 3 and 30. Recalling that the results discussed in the following have been computed as the average of the values obtained with 10 runs for each configuration, we note that each run generated a new BPEL template and corresponding service certificates from scratch.

Let us first consider the performance of the three heuristics and the exhaustive algorithm evaluating their execution time. Figure 7(a) compares their execution time increasing the number of invocations from 3 to 21 (step 3). The exhaustive algorithm has been evaluated in 6 settings varying the number of candidate services for each invocation in 3, 5, 10, 15, 30, 50, while the heuristics considered the worst case of 50 candidates. Figure 7(b) compares the execution time increasing the number of candidate services from 5 to 50 (step 5). The exhaustive algorithm has been evaluated in 5 settings varying the number of invocations in 3, 6, 9, 15, 21, while the heuristics considered the



(a) (b)

Fig. 8. Certification process execution time with (a) local and (b) step-by-step heuristics

worst case of 21 invocations. We note that the execution time of the exhaustive algorithm is reported only for configurations requiring less than 10 minutes, and estimated for configurations over that threshold. In general, all heuristics considerably outperformed the exhaustive algorithm, which required exponential time in the number of invocations and candidate services, with a stronger dependence on the number of invocations. The execution time of local heuristic remained below 1.3 ms, the one of the step-by-step heuristic below 3.38 ms, and the one of the hybrid heuristic below 1566.83 ms. The total execution time of the hybrid heuristic is given by the execution time of local and step-by-step heuristics plus an additional overhead given by the application of the exhaustive algorithm to the set of candidate services derived from local and step-by-step heuristic execution. Although hybrid heuristic is exponential in the worst case, our experiments showed low overhead due to the fact that the set of selected services for local and step-by-step heuristics were very similar and overlapped in average of the 64.2%.

Figure 8(a) and Figure 8(b) present the execution time of the whole certification process, from BPEL instance to certificate generation, on the basis of local and step-by-step heuristics, respectively. The generation time is pseudo-linear in the number of invocations and candidates, and requires 71.989s for local heuristic and 74.678s for step-by-step heuristic in the worst case with 21 invocations and 50 candidates for each invocation. Taking the average of all measurements in our experiments, 93.33% of the time is devoted to BPEL instance generation (of which 0.32% is devoted to heuristic execution), while the remaining 6.67% to virtual certificate production. Our results show that *i)* the number of candidates has the highest impact on performance and *ii)* the overhead given by a security certification-aware service composition is manageable in practice. Also, it is important to note that we considered a very challenging and unusual case as our worst case, in which we have 21 invocations in the BPEL instance and 50 candidates for each invocation that are far greater than real composition scenarios. Moreover, since the greatest part of the certification process execution time is devoted to BPEL instance generation, it can be technically reduced by implementing optimized algorithms or by simply deploying our process on a more powerful workstation. To conclude, these results show that the performance of our approach is suitable to certify service compositions generated at runtime, and can accomplish the dynamics and runtime requirements of a service-based infrastructure.

7.3 Quality Evaluation

Quality evaluation aims to verify the quality of the virtual certificate produced by our heuristics, and in turn of our process, with respect to the best certificate that would have been achieved by using the exhaustive search. Our goal is to show how many times the heuristics reach the optimum, and when the optimum is not reached the amount of preserved quality. We consider optimum quality $\overline{\Theta}(G')$ as the quality achieved by our exhaustive algorithm when all possible combinations of services and corresponding certificates are considered (see Definition 6.2). The quality of an heuristic can then be informally defined as the rate $\frac{\Theta_h(G')}{\overline{\Theta}(G')}$, where $\Theta_h(G')$ is the quality achieved by the heuristic itself. We evaluated the certificate quality guaranteed by our three heuristics for BPEL instance G' on all configurations discussed in the previous sections.

Summarizing our results, all heuristics showed a good quality close to the optimum obtained with the exhaustive search. As expected, hybrid heuristic achieved the best results, while the local one the worst. More in detail, the local heuristic was able to produce BPEL instances with an average quality of 67.9% with respect to the optimum; it was able to produce the optimum in 15.6% of the cases, and when the optimum was not identified, the quality was on average 62% of the optimum. The step-by-step heuristic was able to produce BPEL instances with an average quality of 82.8% with respect to the optimum; it was able to produce the optimum in 33% of the cases, and when the optimum was not identified, the quality was on average 74.3% of the optimum. The hybrid heuristic was able to produce BPEL instances with an average quality of 84.3% with respect to the optimum; it was able to produce the optimum in 42.2% of the cases, and when the optimum was not identified, the quality was on average 72.8% of the optimum. The results achieved by the hybrid heuristic are due to the fact that the union of the candidates selected by local and step-by-step heuristics is often a super-set of the candidates selected by the exhaustive algorithm. These results together with corresponding performance results show that the hybrid heuristic could be a viable approach in cases where there are stronger requirements on quality and weaker requirements on performance.

8 A REAL-WORLD INDUSTRIAL SCENARIO: ENGPAY ONLINE PAYMENT SYSTEM

We further evaluated our approach in a real-world industrial scenario based on the ENGpay payment system of Engineering Ingegneria Informatica S.p.A. The evaluation in this section has a threefold value: *i)* it demonstrates the applicability of our approach to a real industrial scenario; *ii)* it shows the utility of the proposed approach by discussing how certification evidence can be used as a basis to establish a semi-automatic and dynamic solution for compliance verification; *iii)* it proves the generality of the approach and its suitability for certifying complex systems which mix services and traditional software components. We considered a specific deployment of ENGpay composite system, where the certified component services were available and already composed at evaluation time, and focused on the virtual certification process. We started from the definition of the BPEL graph of the Engineering system to the generation of the virtual certificate used for supporting compliance verification against Payment Card Industry Data Security Standard (PCI DSS) [1]. We note that the discussion in this section is valid independently by how the services are selected to produce a BPEL instance, either according to our heuristics in Section 6 or by manually selecting them as assumed in this section. In the following, we first describe PCI DSS standard (Section 8.1) and the ENGPay system architecture (Section 8.2). We then discuss the virtual certification process and its support to PCI DSS compliance verification (Section 8.3). We finally discuss the utility of our approach with respect to traditional compliance approaches and its applicability to real-world software (Section 8.4).

Table 2. PCI DSS Requirements [78].

Build and maintain a secure network	1. Install and maintain a firewall configuration to protect cardholder data 2. Do not use vendor-supplied defaults for system passwords and other security parameters
Protect cardholder data	3. Protect stored cardholder data 4. Encrypt transmission of cardholder data across open, public networks
Maintain a vulnerability management program	5. Use and regularly update anti-virus software on all systems commonly affected by malware 6. Develop and maintain secure systems and applications
Implement strong access control measures	7. Restrict access to cardholder data by business need-to-know 8. Assign a unique ID to each person with computer access 9. Restrict physical access to cardholder data
Regularly monitor and test networks	10. Track and monitor all access to network resources and cardholder data 11. Regularly test security systems and processes
Maintain an information security policy	12. Maintain a policy that addresses information security

8.1 PCI DSS

PCI DSS is a global standard established by financial organizations to protect cardholder data and information linked to users' personal data. According to requirements of PCI DSS, all enterprises and operators who store, process, or transmit cardholder data should meet stringent security standards. In a nutshell, meeting the PCI DSS requirements means that data, such as credit card details and PIN numbers, are processed, managed, transmitted, and stored by data controllers/processors following the appropriate procedures and standards. This minimizes the risks of frauds, both for the card holder and for the point of sale where a payment is accepted. The PCI DSS requirements are summarized in Table 2. PCI DSS requirements in Table 2 can be classified in three classes: environmental/generic requirements (requirements 1, 2, 5, 10, 12), development process requirements (requirements 6, 11), and functional requirements (requirements 3, 4, 7, 8, 9).

8.2 ENGPay payment system

ENGPay is an ePayment system built around a payment hub that provides support for managing any merchants, back-end systems, and innovative payment services. It provides the merchants with a flexible instrument that adapts to the type of product and to the considered users. ENGPay system is composed of the following set of components/services: *i) Auth* (AU) is a service providing authentication functionality to merchants, *ii) Payment Hub* (PH) is the core service of ENGPay, managing a variety of credit sources to satisfy all users' needs concerning the settlement of their accounts, *iii) Acquirer* (AC) is a third party service associated with Merchant responsible for credit card payment authorization, credit card plafond management (on behalf of authorized merchants), and interactions with card issuer bank, *iv) Issuer* (IS) is a third party service provided by the card issuer bank responsible for managing card holder bank account, *v) Selfcare Data Storage* (SD) is a service providing CRM/self-care interface supporting merchants' CRM and customers. We note that, in the original ENGPay specifications, AU and SD were part of service *PH* and specified as traditional software components. We also note that *Acquirer* and *Issuer* together manage the post-trading Cleaning & Settlement process. For simplicity and without loss of generality, we made the following assumption on ENGPay specifications.

(1) *ENGPay system*: we consider a specific ENGPay business process related to payment authorization, modeled as a BPEL composition.
(2) *ENGPay components*: to test the generality of our approach and its applicability to real-world software components, our model of ENGPay specifies AU and SD as being services themselves. This is achieved in practice by defining a proper interface around them and by adding operations for network message management, leaving their implementation almost unchanged. BPEL modeling and service interface are just auxiliary structures which do not impact on compliance verification.

Table 3. ENGPay component services and corresponding operations for payment authorization.

Service	Operation	Description
Auth (AU)	*MerchantToken* `merchantAuth`(*usr, pwd*)	Authenticates the merchant webstore
PaymentHUB (PH)	*OrderId* `receiveTransReq`(*MerchantToken,OrderD*)	Allows customers to send payment requests for an order
	TotalPrice `paymentReg`(*PaymentD, OrderId*)	Allows to collect customers' payment data (e.g. card info)
AcquirerIssuer (AS)	*TransResult* `transAuth`(*PaymentD,TotalPrice*)	Authorizes and executes the transaction request
SelfcareSD (SD)	*SDRes* `write`(*MerchantToken, OrderId, TransResult*)	Securely stores personal transaction information
Parameter/output	**Description**	**Format**
(usr, pwd)	Merchant credentials obtained upon registration	Strings with constraints
MerchantToken	Merchant authentication token	String of 256 characters
OrderD	Order details including products and prices	Structured data with items of multiple types
OrderId	Order identification number	Unique numeric id
PaymentD	Payment details (e.g., credit card number)	Structured data with items of multiple types
TotalPrice	Total cost for the transaction (including fees)	Number
TransResult	Transaction result	String of 256 characters specifying the success/error details
SDRes	The result of selfcare storing	Number (error code)

(3) *ENGPay Cleaning & Settlement process*: we unify Acquirer *AC* and Issuer *IS* in a single service *AcquirerIssuer* (AS) dealing with all the aspects of the Cleaning & Settlement process.

Table 3 describes the set of operations of the payment authorization process, which includes a description of the input/output parameters and corresponding data format. Figure 9 shows the BPEL model of the payment authorization process.[9] Let us consider a *Client* (e.g., the customer of eFlight service in Section 3) that starts a transaction at a *Merchant* website (e.g., the Airline in Section 3 registered to ENGPay). Upon receiving the payment order by the *Client*, the *Merchant* interacts with the ENGPay system for payment authorization (*Auth*.`merchantAuth`) providing its authentication credentials. In the case of positive authentication, ENGPay starts the payment authorization process by executing *PaymentHUB*.`receiveTransReq`. The latter is responsible for communicating the order id (*OrderId*) and the available payment options (e.g., credit card, paypal, bank transfer) back to the Merchant website. ENGPay collects the client's payment details (*PaymentD*) using *PaymentHUB*.`paymentReg` for the authorized order and, depending on the *Merchant*, selects the correct *AcquirerIssuer*. *AcquirerIssuer* receives the payment details and total price (*TotalPrice*), and proceeds with the transaction authorization (*AcquirerIssuer*.`transAuth`). Finally, ENGPay stores the results of the transaction (*TransResult*) in a secure data storage (*SelfcareSD*.`write`) and forwards the final response to *Client* via *Merchant*.

8.3 Certification-based compliance verification

We show how our certification process can be adopted as the basis to verify compliance of ENGPay payment authorization process against PCI DSS standard. For simplicity and without loss of generality, we made the following assumptions on compliance verification.

(1) *PCI DSS*: we consider PCI DSS requirements 3 and 4 in Table 2 (protect card holder data), which impacts on ENGPay payment authorization process. According to the Engineering security guideline, the requirement 3 has been extended with a check on the integrity of data in transit.
(2) *Compliance verification*: we describe a certification-based compliance verification for PCI DSS requirements 3 and 4, while providing the details of virtual certificate generation for just one specific security annotation related to requirement 3.

[9]We note that, although this BPEL has been defined on the basis of ENGPay specifications, it can be considered as a representative of a generic payment system.

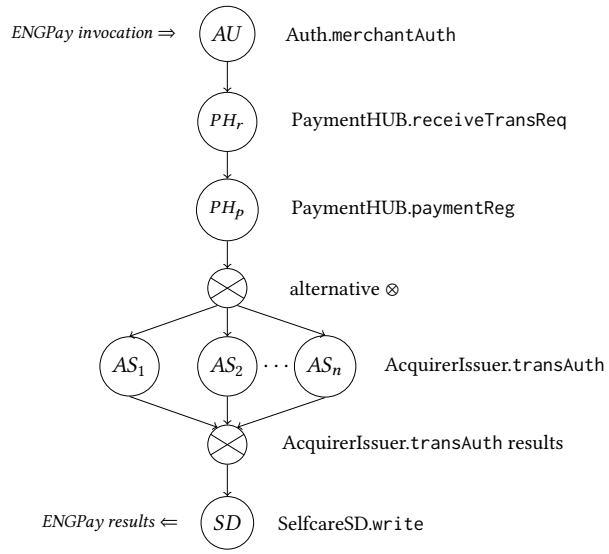


Fig. 9. BPEL graph of ENGPay payment authorization based on operations in Table 3.

Table 4. Mapping between PCI DSS Requirements, security property annotation on the BPEL graph of ENGPay, and target virtual property.

Req	Security property	BPEL operations	Virtual property
3	Integrity of data in transit	All operations	Integrity of data in transit
3	Integrity of data in storage	*SelfcareSD*.Write	Integrity of data in storage
4	Confidentiality of data in transit	All operations	Confidentiality of data in transit
4	Confidentiality of data in storage	*SelfcareSD*.Write	Confidentiality of data in storage

(3) *Virtual certificate generation*: we provide some concrete examples of virtual evidence generation focusing on a subset of representative test equivalence classes. For the sake of conciseness, we privileged the description of a realistic certification, sacrificing the description of a full certification process.

In the following, we discuss in detail the virtual certification of ENGPay for the verification of requirement 3, including the evidence generation process, and summarized the verification of requirement 4.

Verification of requirement 3. The first step of our compliance verification process is the definition of the security annotation (see Section 5.1) for the BPEL graph in Figure 9. Table 4 provides simplified security annotations where, for each of the considered requirements, it is specified a subset of BPEL operations in Table 3 with the security property they must support and the virtual property produced as a result of a successful certification process. Security annotations have been inspired by existing work on PCI DSS compliance verification of Assuria Log Manager (ALM) and CyberSense Enterprise Scanner (http://www.assuria.com/). We note that, since PCI DSS

requirements are more abstract than security annotations, multiple security annotations (and BPEL template) can be defined for each requirement.

Being focused on requirement 3, our compliance verification process configures and executes two virtual certification processes focusing on the certification of two different properties as follows.

Table 5. Equivalence class annotation for BPEL graph of ENGPay authorization system. (a) Legenda, (b) equivalence classes for property *Integrity of data in transit*, (c) certificates of ENGPay component services. Equivalence classes denoted with "-" are unused, gray cells represent special equivalence classes for intermediate test case generation.

Legenda				
$[< val >]$=parameter value *val*	s=valid and correct signature	s'=valid but incorrect signature	$\overline{s}$=invalid signature	*any_s*=any signature
5y=5 years	n=CC plafond	*sw*=signature wrapping attack	*reply*=reply attack	*ACL*=access control list

(a)

Integrity of data in transit							
	ec / Parameter	Functional					
		1	2	3	4	5	6
1	*(usr, pwd)*	$[\in ACL]_s$	$[\in ACL]_{s' \neq s}$	$[\notin ACL]_s$	$[\notin ACL]_{s' \neq s}$	-	-
2	*MerchantToken*	$[\in \exists]_s$	$[\in \exists]_{s' \neq s}$	$[\in expired]_s$	$[\in expired]_{s' \neq s}$	$[\in \nexists]_s$	$[\in \nexists]_{s' \neq s}$
3	*OrderD.TotalCost*	$[= [0-9]^*.[0-9]^2]_s$	$[= [0-9]^*.[0-9]^2]_{s' \neq s}$	-	-	-	-
4	*OrderId*	$[= [0-9]^5]_s$	$[= [0-9]^5]_{s' \neq s}$	-	-	-	-
5	*PaymentD.CC*	$[\in Visa]_s$	$[\in Visa]_{s' \neq s}$	$[\in MC]_s$	$[\in MC]_{s' \neq s}$	$[\in AE]_s$	$[\in AE]_s$
6	*PaymentD.ExpDate*	$[> Today + 5y]_s$	$[> Today + 5y]_{s' \neq s}$	$[< Today]_s$	$[< Today]_{s' \neq s}$	$[< Today \wedge > Today + 5y]_s$	$[< Today \wedge > Today + 5y]_{s' \neq s}$
7	*TotalPrice*	$[> 0 \wedge < n]_s$	$[> 0 \wedge < n]_{s' \neq s}$	$[> n]_s$	$[> n]_{s' \neq s}$	–	–
8	*TransResult*	$[= [A-Za-z0-9]^{256}]_s$	$[= [A-Za-z0-9]^{256}]_{s' \neq s}$	-	-	-	-

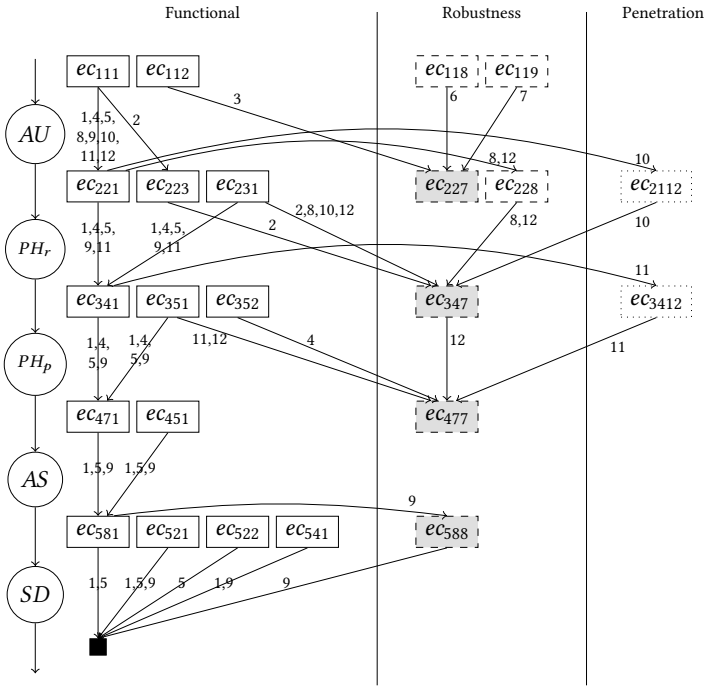
	ec / Parameter	Robustness				Penetration	
		7	8	9	10	11	12
1	*(usr, pwd)*	$[\in (\emptyset, any) \vee (any, \emptyset)]_{any\_s}$	$[\in invalid]_{any\_s}$	$[any\ value]_{\overline{s}}$	-	sw	-
2	*MerchantToken*	$[= \emptyset]_{any\_s}$	$[\neq [A-Za-z0-9]^{256}]_{any\_s}$	$[any\ value]_{\overline{s}}$	-	sw	$reply$
3	*OrderD.TotalCost*	$[= \emptyset]_{any\_s}$	$[< 0]_{any\_s}$	$[= [0-9]^*.[0-9]^2]_{any\_s}$	$[any\ value]_{\overline{s}}$	sw	-
4	*OrderId*	$[< 0]_{any\_s}$	$[\neq [0-9]^5]_{any\_s}$	$[any\ value]_{\overline{s}}$	-	sw	$reply$
5	*PaymentD.CC*	$[= \emptyset]_{any\_s}$	$[\notin \{Visa, MC, AE\}]_{any\_s}$	$[any\ value]_{\overline{s}}$	-	sw	-
6	*PaymentD.ExpDate*	$[= \emptyset]_{any\_s}$	$[\neq dd/mm/aaaa]_{any\_s}$	$[any\ value]_{\overline{s}}$	-	sw	-
7	*TotalPrice*	$[< 0]_{any\_s}$	$[any\ value]_{\overline{s}}$	-	-	sw	-
8	*TransResult.Descr*	$[\neq [A-Za-z0-9]^{256}]_{any\_s}$	$[any\ value]_{\overline{s}}$	-	-	sw	-

(b)

Simplified Certificate related to Integrity of data in transit annotation					
1	C_{AU}(MerchantAuth)	p=$(Integrity, \{ctx = in-transit, algo = MD5\})$			
		cat()=*Functional*	$t_1 = (ec_{111}, ec_{221})$	$t_2 = (ec_{111}, ec_{223})$	$t_3 = (ec_{112}, \overline{ec}_{227})$
		cat()=*Robustness*	$t_4 = (ec_{118}, \overline{ec}_{227})$	$t_5 = (ec_{119}, \overline{ec}_{227})$	-
2	C_{PH_r}(ReceiveTransReq)	p=$(Integrity, \{ctx = in-transit, algo = MD5\})$			
		cat()=*Functional*	$t_6 = ((ec_{221}, ec_{231}), ec_{341})$	$t_7 = ((ec_{223}, ec_{231}), \overline{ec}_{347})$	
		cat()=*Robustness*	$t_8 = ((ec_{221} \overset{m}{\rightsquigarrow} ec_{228}, ec_{231}), \overline{ec}_{347})$	-	-
		cat()=*Penetration*	$t_9 = ((ec_{221} \overset{m}{\rightsquigarrow} ec_{2212}, ec_{231}), ec_{347})$	-	-
3	C_{PH_p}(PaymentReg)	p=$(Integrity, \{ctx = in-transit, algo = MD5\})$			
		cat()=*Functional*	$t_{10} = ((ec_{341}, ec_{351}), ec_{471})$	$t_{11} = ((ec_{341}, ec_{352}), \overline{ec}_{477})$	-
		cat()=*Robustness*	$t_{12} = ((\overline{ec}_{347}, ec_{351}), \overline{ec}_{477})$	-	-
		cat()=*Penetration*	$t_{13} = ((ec_{3412}, ec_{351}), \overline{ec}_{477})$	-	-
4	C_{AI_1} and C_{AI_2} (FwdTransAuth)	p=$(Integrity, \{ctx = in-transit, algo = MD5\})$			
		cat()=*Functional*	$t_{14} = ((ec_{471}, ec_{451}), ec_{591})$	$t_{15} = ((ec_{473}, ec_{451}), ec_{591})$	-
		cat()=*Penetration*	$t_{16} = ((ec_{471}, \overline{ec}_{4511}), ec_{581})$	-	-
5	C_{SS}(write)	p=$(Integrity, \{ctx = in-transit, algo = SHA1\})$			
		cat()=*Functional*	$t_{17} = ((ec_{581}, ec_{521}, ec_{541}), *)$	$t_{18} = ((ec_{581}, ec_{522}, ec_{541}), *)$	-
		cat()=*Robustness*	$t_{19} = ((ec_{581} \overset{m}{\rightsquigarrow} ec_{587}, ec_{521}, ec_{541}), *)$	-	-
		cat()=*Penetration*	$t_{20} = ((ec_{581}, \overline{ec}_{5211}, ec_{541}), *)$	-	-

(c)

- *Property integrity of data in transit.* Upon the definition of the security annotation (row 1 in Table 4), we define the equivalence class annotation that associates a set of equivalence classes with each input parameter of the involved BPEL invocation. Table 5(b) shows equivalence class annotation. Each equivalence class is defined as the range of values that can be assumed by the corresponding parameter, and belongs to either functionality, robustness, or penetration category. Each class in Table 5(b) is denoted with $ec_{<par><part>}$, where <par> refers to the specific input parameter and <part> to the partition number. For instance, ec_{65} refers to the fifth equivalence class of input parameter *PaymentD.ExpDate* that collects credit cards with a valid-format expiration date, which is either lower than today or greater that 5 years. We note that, since we are testing property integrity of data in transit, equivalence classes also consider signatures associated with each parameter. For instance, let us consider the equivalence class of existing tokens for parameter *MerchantToken* (ec_{21}). This equivalence class results in two equivalence classes when the signature is considered, that is, an existing token with valid and correct signature s (ec_{21}=$[\in \exists]_s$) and an existing token with valid but incorrect signature $s' \neq s$ (ec_{22}=$[\in \exists]_{s' \neq s}$). The same discussion holds for equivalence classes representing expired (ec_{23}, ec_{24}) and not-existing (ec_{25}, ec_{26}) tokens. Equivalence classes of category robustness include invalid values which can be rejected before executing the operation. For instance, robustness equivalence classes for *MerchantToken* specify an empty (ec_{27}=$[= \emptyset]_{any}$) and invalid (ec_{28}=$[\neq [A-Za-z0-9]^{256}]_{any}$) token with any signature and any token with invalid signature (ec_{29}). Finally, each equivalence class of type *Penetration* contains a pointer to a type of attack, which must then be represented with a complete attack flow [13]. For instance, the penetration equivalence class *reply* on *MerchantToken* abstracts all operations needed to implement a reply attack such as the operations to guess or stole a valid token.
  A BPEL instance (see Section 5.2) is then generated according to the BPEL template. In our case, we have a possible instantiation only (see Figure 9), which integrates the fixed set of ENGPay component services. Table 5(c) shows an overview of the certificates of component services including: *i)* security property p, *ii)* a subset of test cases t specified as a pair of input and output equivalence classes in Table 3(b). We denote each equivalence class as $ec_{<op><par><part>}$ where <*op*> is the operation id (row number) in Table 5(c), <*par*> is the parameter id (row number) in Table 5(b), and <*part*> is the partition id (column number) in Table 5(b). For instance, ec_{5211} refers to the equivalence class *sw* (signature wrapping) for parameter *MerchantToken* parameter of operation `write` of service SD. We note that, in Table 5(c), all test cases associated with *cat*()=*Functional* belong to *type*()=*Input Partitioning.Random input*, all test cases associated with *cat*()=*Robustness* belong to *type*()=*Invalid.Random input*, and all test cases associated with *cat*()=*Penetration* belong to *type*()=*Attack*.
  Certification-based compliance verification then starts following the approach presented in Section 5.3 and aims to produce a virtual certificate according to the security annotation in row 1 of Table 4. The virtual certificate is generated by composing certificates of component services according to our algebra as follows: $C_{AU} \odot C_{PH_r} \odot C_{PH_p} \odot C_{AS} \odot C_{SD}$, assuming for conciseness a single service *AS*. Virtual property *integrity of data in transit* is first generated, because the properties of the component services in Table 5(c) satisfy the security annotation in Table 4. Virtual model is then generated as the composition of the single service models in corresponding certificates, according to the methodology in Section 4.3. Finally, virtual evidence is produced by composing the test cases of single certificates in Table 5(c), following equivalence class annotation in Table 5(b) (see Section 4.4). Figure 10 shows the output of the virtual test case generation process in terms of *i)* testing flows virtually executed on the composition (Figure 10(a)) and *ii)* corresponding virtual test cases (Figure 10(b)). Each testing



(a)

$t_1^*=t_1 \odot t_6 \odot t_{10} \odot t_{14} \odot t_{17}$	$t_2^*=t_2 \odot t_7$	$t_3^*=t_3$
$t_4^*=t_1 \odot t_6 \odot t_{11}$	$t_5^*=t_1 \odot t_6 \odot t_{10} \odot t_{14} \odot t_{18}$	$t_6^*=t_4$
$t_7^*=t_5$	$t_8^*=t_1 \odot t_8$	$t_9^*=t_1 \odot t_6 \odot t_{10} \odot t_{14} \odot t_{19}$
$t_{10}^*=t_1 \odot t_9$	$t_{11}^*=t_1 \odot t_6 \odot t_{13}$	$t_{12}^*=t_1 \odot t_8 \odot t_{12}$

(b)

Fig. 10. Virtual evidence generation for PCI DSS compliance verification of ENGPay: (a) testing flows, (b) virtual test cases

flow is a path connecting equivalence classes of operation calls in the BPEL graph; each path can be retrieved by following annotations on arrows connecting two equivalence classes. We note that some arrows have multiple annotations since they are involved in multiple virtual testing flows. Virtual testing flows represent an overview of testing activities that must be done, either physically or virtually, to certify ENGPay implementation.

As an example, let us consider the virtual flow labeled with 1, resulting in the virtual test case t_1^* of category functionality. According to Figure 10(b), virtual test case t_1^* integrates test cases t_1, t_6, t_{10}, t_{14}, and t_{17} in a sequence on the basis of the specified equivalence classes. For instance, test case t_1=(ec_{111},ec_{221}) is integrated with test case t_6=((ec_{221},ec_{231}),ec_{341}) on the basis of equivalence class ec_{221}. Two interesting examples involve testing flows 8 and 12 of category robustness, resulting in virtual test cases t_8^* and t_{12}^*, respectively. Test case t_8^* integrates t_1 and t_8, through a mutation between ec_{221} and ec_{228}, and is generated thanks to $\overline{ec}_{347}$, an equivalence class triggering generation of virtual test cases covering a subpath

of the composition that starts from its root. Test case t^*_{12} extends t^*_8 with t_{12}, includes $\overline{ec}_{347}$ treated as a normal equivalence class, and is generated thanks to $\overline{ec}_{377}$.

- *Property integrity of data in storage.* Based on the security annotation in Table 4 (row 2), the verification process is reduced to a simple check on the availability of a certificate proving property *integrity of data in storage* for operation `write` of service SD. Assuming such certificate exists for SD.`write`, all security annotations related to requirement 3 in Table 4 are satisfied and the compliance to PCI DSS requirement 3 verified.

Verification to requirement 4. Similarly to the above process, compliance verification of requirement 4 requires two virtual certification processes. The first process aims to prove property *confidentiality of data in transit*. Based on the security annotation in Table 4 (row 3), the process is very similar to the one discussed for property *integrity of data in transit* with the main difference that test cases verify the correctness of the encryption mechanism, rather than the one of the signature mechanism. The second process aims to prove property *confidentiality of data at rest*. Based on the security annotation in Table 4 (row 4), the verification process is reduced to a simple check on the availability of a certificate proving property *confidentiality of data in storage* for operation `write` of service SD. Assuming both certification processes are successful, all security annotations related to requirement 4 in Table 4 are satisfied and the compliance to PCI DSS requirement 4 verified.

The satisfaction of requirements 3 and 4 allows to prove the PCI DSS high-level requirement "Protect Cardholder Data".

8.4 Discussion

The experimental evaluation of our security certification scheme has been conducted along three main lines.

The first line (Section 7), considering a simulation-based approach, evaluated the scalability, performance, and quality of the proposed heuristics. It provided the diversity in terms of services and their composition that can be given only by processes specifically designed for testing purposes. This evaluation in fact considered huge scenarios with composite services orchestrating up to 21 component services and 50 alternative candidate services for each component service, making the evaluation of our solution and heuristics stronger. These numbers are clearly not possible with real-world runtime compositions. Also, a simulation approach with random generation of all experimental building blocks provides an independent, objective, and impartial evaluation.

The second line (in this section) evaluated the applicability of our approach to a real industrial scenario, which mixes traditional software components and services. The experiments proved the suitability of our approach for the certification of the ENGPay payment system, and showed that the certification of systems involving traditional software components can be achieved with a simple adaptation process. This process consists of the definition of a proper interface around software components and by adding operations for network message management. The modeling effort requested to adapt ENGpay system specifications towards a BPEL workflow required standard competences to Engineering employees and completed in few days. The main effort was spent in the labeling of the BPEL template with equivalence class and security annotations, while the modeling effort was less demanding. In particular, we spent 3 days to analyze the interactions between components and label the interfaces accordingly. Then, after a first evaluation, we spent additional 2 days in the refinement of our labeling to make it finer-grained.

The third line evaluated the utility of our solution by showing how our certification-based approach can be used to provide verifiable evidence of a service behavior at the basis of existing compliance and certification standards. In particular, we implemented a certification-based compliance verification process, verifying ENGPay payment system against PCI DSS standard.

9 RELATED WORK

The problem of certifying that a given system has some properties and behaves as expected is a time-honored research topic, which has been considered well before the advent of IT and distributed systems. Today, people live in certified houses, drive certified cars, and trade goods with certified retailers. With the advent of IT, distributed systems, and cloud infrastructures, people realized that they interact with previously unknown IT services on which they have little or no information. In this context, certification can increase the confidence of the users that their life, processes, and data are treated safely and as expected.

9.1 Service certification

Early certification schemes focused on non-functional properties (e.g., security) of software and provided powerful, while static and rigid, solutions that well adapted to software-based systems. The most important example of such certification schemes, which today is widely adopted, is Common Criteria (ISO 15408) [46]. Common Criteria provides a framework to specify, design, and evaluate security properties, and is used in many critical scenarios as for instance to certify products for the US Army and US Government. With the advent of service-based and cloud-based systems, certification schemes are entering a new phase, and new solutions have been presented to support the dynamic nature of services and runtime processes (e.g., service discovery and selection) at the basis of such systems. Existing approaches to service certification have focused on non-functional properties of single services (e.g., [6, 8, 13, 36, 43, 52, 55, 70]), also considering service evolution [8, 36, 55]. Some works (e.g., [10, 11, 87]) also addressed the problem of trustworthy service certification in cloud environments providing test-based approaches similar to the one in this paper. For instance, Anisetti at al. [10] proposed a test-based certification scheme driven by requirements defined by the certification authority and the model of the service under certification. To increase certification trustworthiness, the proposed solution implemented an automatic approach to the verification of consistency between models and requirements. It also provided a solution to certificate life cycle management including an automatic and incremental approach to certificate adaptation, addressing the multi-layer and dynamics nature of the cloud. To further strengthen certification trustworthiness, the correctness of the corresponding certification model must be carefully verified. Anisetti at al. [11] proposed a certiiňĄcation scheme where the verification of the system model correctness is based on real and synthetic service execution traces in operational environments, according to time, probability, and coniňĄguration constraints, and attack iňĆows. In this paper, we consider and extend the above approaches towards composite service certification. The first approach to composite service certification has been proposed by Anisetti at al. [13], which considered the certification of composite services in a containment relationship, where a service can inherit part of the security properties certified on its container. However, further analysis is required when more complex compositions like orchestrations in this paper or choreography in [80] are taken into account.

9.2 Service composition

A recent survey on service composition by Lemos et al. [58] underlines the impact of service compositions in everyday computing tasks, providing a comprehensive composition taxonomy with a particular emphasis on the role of composition languages. A composition language is

responsible to define the components and the composition logic, and to execute the composite services. Among the languages presented in [58], BPEL stands out as a language covering SOAP services, for which it has been originally designed, as well as RESTful services via "BPEL for REST" in [77]. Being designed for BPEL, our approach is suitable for both design-time and deployment-time component selection, and is generic enough to be easily adapted to any BPM description languages (e.g., BPMN v2) or automata-based description of compositions like the one based on State Transition System used by Zhao et al. [44] for RESTful services. Furthermore, our approach partially covers the gaps in the engineering of component services pointed out by Lemos et al. [58], providing transparency of non-functional properties of heterogeneous component services by means of certification, and a suitable way for generating testing plans for composite services.

Other works specifically focused on providing solutions to verify service compatibility in terms of functionalities and interfaces (e.g., [48, 88, 89]). In the work of Hwang et al. [48], a composition approach that preserves system functionalities is extended with the notion of reliability of component services. Similar to our approach, the authors adopt a Finite State Machine model for Web services to map constraints on sequences of operation invocations and a BPEL-based model for the composition of web services. They focus on dynamically and incrementally selecting a web service to maximize the likelihood of successful execution in a failure-prone operating environment. In this context, they introduce the concept of *aggregated reliability*, which is a metric to measure the probability that a given state in the composite web service will lead to successful execution in an error-prone environment. Najafi et al. [72] propose to add a Service Composition Certify (SCC) component directly into the SOA model. The scope is to certify client-defined service composition models. The SCC-based certification is mainly focused on compatibility checking of component services, and on policy-based verification of clients' expectations. The proposed policies are not suitable for security specification, while they allow to check functional behavior of a given composition. Bennara et al. [18] focus on RESTful linked services composition and propose a composition engine with a unified method for service discovering enriched with semantics.

Much in line with the work presented in this paper, research has focused on checking and verifying non-functional properties of composite services and on security annotations for service compositions (e.g., [15, 17, 51, 81, 94]). Souza et al. [15] first express a composition as a BPMN annotated with non-functional requirements, and then translate it into a BPEL satisfying the annotations. The work in [81], instead, provides a solution based on patterns and logical rules for building and maintaining secure compositions at runtime. Our approach differs from all the above ones since it does not rely on external components, expresses XML-based security annotations directly in the BPEL file, and implements a test-based security certification scheme for composite services exploiting the test-based evidence included in the certificates of the component services. Vo et al. [94] propose a framework targeting web service compositions with the specific scope of securing data. Using this framework, composite service developers specify security policies for received data, which are applied and updated during the BPEL execution. The approach in [94] considers non-functional properties of compositions from a developer point of view, supporting developers in the design and deployment of security-aware compositions. She at al. [86] provide an access control solution supporting composition-time validation. The paper develops a three-step composition protocol that includes information flow control. Historical information is used to identify candidate compositions and execute policy evaluation on selected candidates only, thus reducing evaluation costs.

9.3 Automatical test of composition

Another line of research has analyzed the problem of modeling and automatically testing service compositions. Mateescu and Rampacek [67] define an approach to model BPEL-based business

processes and web services, and to analyze them using standard verification tools for concurrent systems. Bentakauk et al. [19] propose a solution using STS-based testing and STM solver to check the functional conformance of the composite service with respect to its specifications and/or client requirements. Gao et al. [40] study the problem of testing service composition adopting an extended BPEL4WS model named probabilistic timed interface automata for Web service (PTIA4WS). Yu et al. [95] present a framework for the definition and test-based verification of dependencies between processes under test and their partner services. Many works (e.g., [24, 56, 59, 73]) have proposed solutions to select and generate test cases for service compositions. Differently from these works, our solution provides an approach that automatically generates a security certificate with virtual test-based evidence for runtime compositions, starting from the certificates of the component services. In other words, no runtime testing is needed.

9.4 Formal model -based certification

Many approaches exist for formal verification of systems and satisfaction of security properties, ranging from logic such as BAN [23] or Temporal Logic [82] to formal languages and automaton analysis such as AVISPA [14] and AVANTSSAR [16]. This latter category typically uses the concept of Safety/Lifeness and Hypersafety/Hyperlifeness [29] to describe properties within a dynamic system. These properties however have a behavioral background, rather than a security-related motivation.

Another widely acknowledged approach for the specification of security properties in terms of a formal model is non-interference. Mantel [66] gave a good insight into this topic. Yet, non-interference uses the concepts of local view and initial knowledge implicitly, while within SeMF these are specified separately for each agent, thus enabling to differentiate between agents with different capacities. Furthermore, SeMF includes the notion of trust to capture mechanisms based on Trusted Third Parties (i.e., in particular trust establishment by way of certification). The model based composition of systems is a field of growing research activity in the last decade. Tout et al. [90] developed a methodology for the composition of web services with security. They use BPEL language for the specification of web service compositions and extend it to the specification of security properties, which are independent from the business logic. However, their approach does not address verification of security properties in such compositions. Rossi [84] presented a logic-based technique for verifying both security and correctness properties of multilevel service compositions. Service compositions are specified in terms of behavioral contracts, which provide abstract descriptions of system behaviors by means of terms of a process algebra. Multi-party service compositions are modeled as the parallel composition of such contracts. Modal mu-calculus formulae are used to characterize non-interference and compliance (i.e., deadlock and livelock free) properties. With this approach it can be proven that data transmitted to a web service remains confidential in case this service is part of a composition. The approach is further able to reveal confidentiality breaches caused by dependencies of actions within different components of a composition. Fuchs and Gürgens [39] introduced an approach for the composition of system models. It derives security proofs from specific conditions concerning the component interface.

Another line of research is based on rely-guarantee reasoning. Approaches in this line are mainly applied to functional verification of concurrent/distributed programs compositions [42, 45, 60, 85]. Some of the rely-guarantee reasoning approaches addressed the security of the compositions specifically for few well-defined situations like safety in shared-memory, information flows, protocols of key exchanges, to name but a few. Garg et al. [41] proposed a formal framework for compositional reasoning about secure systems based on an extension of rely-guarantee reasoning for system correctness. This approach considers interface-confined adversaries, addresses the problem considering a multi-thread scenario, and shows its applicability to a very specific security property

related to authorization at file system level. Datta et al. [33] focused on safety properties, which are presented as possibly amenable to compositional reasoning. The authors pointed out the need of local interface and global reasoning. While local interface reasoning can be applied in cloud service environment, global reasoning cannot as also pointed out by Dustdar et al. [35].

To the best of our knowledge, the generality, dynamics, and adaptability required by a certification framework for the cloud, make any holistic global assumptions unfeasible and the practical applicability of formal methods limited. In addition, some of the rely-guarantee approaches are based on white-box or code-level inspection of the composition, which is not always possible. Rely-guarantee reasoning is then applicable for some static service compositions in a controlled environment and for specific properties, but do not scale and generalize to a dynamic composition.

9.5 QoS-aware composition

Other approaches have focused on QoS-aware composition of Web Services [2, 3, 20, 53, 65] showing a level of similarity with security-aware composition. Aggarwal et al. [2] present a semantic web-based approach for achieving constraint-driven Web service composition. The authors present a Constraint-Driven Web Service Composition tool developed in the METEOR-S project. The tool allows process designers to bind Web Services to an abstract process described as BPEL4WS, based on business and process constraints, and generate an executable process. Berbner et al. [20] present an heuristic approach with the goal of selecting Web Services so that the overall QoS and cost requirements of the composition are satisfied. More recently, Alrifai et al. [3] propose a hybrid solution that combines global optimization with local selection techniques, to produce composite services addressing user constraints on end-to-end QoS. Although the proposed approach to service selection is similar to the one used in this paper, the optimization strategy in [3] is not suitable for a scenario that considers security requirements and service certificates. Ma et al. [65] deal with more complex contexts where compositions involve reconfigurable services (increasing the decision space). To find the optimal composition, they adopt a Compositional Decision-making Process (CDP) and combine it with multi-objective evolutionary algorithm (MOEA), obtaining the compositional MOEA (CMOEA). CMOEA explores optimal solutions of individual component services and uses this knowledge to derive optimal QoS-driven composition solutions.

Recently, some effort has been done on optimal QoS-aware selection of services [25, 34, 97]. Chen et al. [25] study the problem of QoS-aware service composition from a general Pareto optimal angle, and propose a parallel approach for improving the performance of the multi-objective optimal searching algorithm. Deng at al. [34] propose the correlation-aware service pruning (CASP), a method for service selection. CASP manages QoS correlations by accounting for all services that may be integrated, and by pruning services that are not the optimal candidates based on the QoS metrics and the correlation with adjacent tasks. These approaches consider just simple compositions made by sequences of services and simple QoS metrics mainly based on response time and service price. Zheng et al. [97] present an approach to QoS evaluation for composite services handling any QoS probability distributions and basic compositional patterns (i.e., sequence, conditional, parallel and loop).

As far as cloud computing is concerned, the testing notion at the basis of our certification evidence generation becomes fundamental to achieve optimum composite services [32, 64, 91]. Dastjerdi et al. [32] focus on functional and QoS aspects of cloud service composition with particular emphasis on usability, from users point of view, in terms of definition of compositional requirements. The paper presents a semantic-aware framework and algorithms aimed to simplify cloud service composition for non-expert users. The framework adopts a repository of cloud services defined using Web Service Modeling Language (WSML) and enriched with expert knowledge as the basis for cloud service compatibility checking. In addition, to minimize the users' effort in expressing their

preferences, the framework adopts a combination of evolutionary algorithms and fuzzy logic for composition optimization. Tsai et al. [91] propose an approach to facilitate service composition and testing in the cloud. Composition and testing activities are mainly focused on functional aspects, while non-functional aspects are not considered. The service composition is supported by service injection mechanisms allowing users to define interfaces and dependencies at design time, and inject service implementation later. The selection of component services is left to the cloud provider, which is responsible to execute testing activities at runtime for discovering a composition suitable for the client. Lu et al. [64] propose a global trust service composition approach based on random QoS and trust evaluation, considering the multi-criteria assessment of service quality. The paper considers a cloud scenario and is aimed at ensuring the stability and reliability of QoS attributes for satisfying the need of customers, providers, and third-parties.

The approach in this paper implements a certificate-aware service composition, where testing evidence is taken from existing certificates of single services without requiring real testing activities on the composition. It supports fully dynamic composition without a priori knowledge on the candidate services. Our approach is complementary to other solutions described in this section that rely on just-in-time composition evaluation. In fact, these approaches can be applied after a BPEL instance is created and certified using the technique put forward in this paper, to evaluate and confirm the quality of the composition by means of testing and/or monitoring.

10 CONCLUSIONS

When critical business processes are implemented as dynamic composite services in cloud- and SOA-based environments, it is important to be able to guarantee an adequate level of security assurance for these services. Composite services are implemented by orchestrating single services provided by different suppliers, which are often not under the control of business process owners. This scenario provides several advantages in terms of flexibility, costs, and interoperability, but it increases the risk of new security threats that might prevent customers from using such "untrusted" services.

The assurance solution presented in this paper allows to evaluate the security properties of BPEL-based service compositions, managing the intrinsic dynamics of services. It provides an accurate, low-cost, and robust test-based security certification scheme that supports process owners in the certification of composite services. Our scheme is based on the idea to produce a virtual test-based security certificate for the composition, where the test-based evidence proving a property is inferred from the evidence used to certify the individual components.

Although we focused on BPEL-based processes, our scheme can be easily integrated with any orchestration approach, thus supporting dynamic and certification-aware selection and composition of services. The high level of automation and the low certification effort provided/required by our virtual certification process makes it a viable approach for highly-dynamic service composition scenarios. Also, it can be used as a tool for producing verifiable evidence of a service behavior supporting existing compliance and certification standards.

ACKNOWLEDGMENTS

This work was partly supported by the program "Piano sostegno alla ricerca" funded by Università degli Studi di Milano. We would also like to thank Domenico Presenza (Engineering Ingegneria Informatica S.p.A.) for its continuous support towards the compliance verification of ENGPay system.

REFERENCES

[1] R. Accorsi, L. Lowis, and Y. Sato. 2011. Automated certification for compliant cloud-based business processes. *Business & Information Systems Engineering* 3, 3 (2011), 145–154.

[2] R. Aggarwal, K. Verma, J. Miller, and W. Milnor. 2004. Constraint Driven Web Service Composition in METEOR-S. In *Proc. of the 2004 IEEE International Conference on Services Computing (SCC 2004)*. Shangai, China.

[3] M. Alrifai, T. Risse, and W. Nejdl. 2012. A hybrid approach for efficient Web service composition with end-to-end QoS constraints. *ACM Transactions on the Web (TWEB)* 6, 2 (June 2012), 1–31.

[4] A. Alves et al. 2007. *Web Services Business Process Execution Language Version 2.0*. OASIS. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[5] J.H. Andrews, L.C. Briand, Y. Labiche, and A.S. Namin. 2006. Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria. *IEEE Transactions on Software Engineering* 32, 8 (August 2006), 608–624.

[6] M. Anisetti, C.A. Ardagna, and E. Damiani. 2011. Fine-Grained Modeling of Web Services for Test-Based Security Certification. In *Proc. of the IEEE International Conference on Services Computing (SCC 2011)*. Washington, DC, USA.

[7] M. Anisetti, C.A. Ardagna, and E. Damiani. 2013. Security Certification of Composite Services: A Test-Based Approach. In *Proc. of the 20th IEEE International Conference on Web Services (ICWS 2013)*. San Francisco, CA, USA.

[8] M. Anisetti, C.A. Ardagna, and E. Damiani. 2015. A Test-Based Incremental Security Certification Scheme for Cloud-Based Systems. In *Proc. of the 12th IEEE International Conference on Services Computing (SCC 2015)*. New York, NY, USA. short paper.

[9] M. Anisetti, C. Ardagna, E. Damiani, and F. Gaudenzi. 2016. A certification framework for cloud-based services. In *Proc. of the ACM Symposium on Applied Computing (SAC 2016)*. Pisa, Italy.

[10] M. Anisetti, C.A. Ardagna, E. Damiani, and F. Gaudenzi. 2017. A semi-automatic and trustworthy scheme for continuous cloud service certification. *IEEE Transactions on Services Computing* (2017).

[11] M. Anisetti, C.A. Ardagna, E. Damiani, N. El Ioini, and F. Gaudenzi. 2018. Modeling time, probability, and configuration constraints for continuous cloud service certification. *Computers & Security (COSE)* 72, Supplement C (2018), 234–254.

[12] M. Anisetti, C.A. Ardagna, E. Damiani, and J. Maggesi. 2012. Security Certification-Aware Service Discovery and Selection. In *Proc. of 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2012)*. Taipei, Taiwan.

[13] M. Anisetti, C.A. Ardagna, E. Damiani, and F. Saonara. 2013. A Test-based Security Certification Scheme for Web Services. *ACM Transactions on the Web (TWEB)* 7, 2 (May 2013), 1–41.

[14] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. Drielsma, P. Héam, O. Kouchnarenko, and et al. J. Mantovani. 2005. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proc. of the 17th International Conference on Computer Aided Verification (CAV-2005)*. Edinburgh, Scotland.

[15] A.R.R. Souza et al. 2009. Incorporating Security Requirements into Service Composition: From Modelling to Execution. In *Proc. of the 7th International Joint Conference on Service Oriented Computing (ICSOC-ServiceWave 2009)*. Stockholm, Sweden.

[16] AVANTSSAR project. 2018. *The AVANTSSAR project IST-2001-39252*. http://www.avantssar.eu/.

[17] Y. Bai, Y. Zhang, Y. Zhou, and L.T. Yang. 2011. A non-functional property based service selection and service verification model. In *Proc. of the 8th International Conference on Ubiquitous Intelligence and Computing (UIC 2011)*. Banff, Canada.

[18] M. Bennara, M. Mrissa, and Y. Amghar. 2014. An approach for composing RESTful linked services on the web. In *Proc. of the 23rd International Conference on World Wide Web (WWW 2014)*. Seoul, Republic of Korea.

[19] L. Bentakouk, P. Poizat, and F. Zaïdi. 2009. A Formal Framework for Service Orchestration Testing Based on Symbolic Transition Systems. In *Proc. of the 21th IFIP Internation Conference on Testing of Communicating Systems (TESTCOM 2009) and the 9th International Workshop on Formal Approaches to Testing of Software (FATES 2009)*. Eindhoven, The Netherlands.

[20] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz. 2006. Heuristics for QoS-aware Web Service Composition. In *Proc. of the IEEE International Conference on Web Services (ICWS 2006)*. Chicago, IL, USA.

[21] B. Bernhard. 2003. Web services container. http://www.google.com/patents/US20030033369 US Patent App. 10/215,722.

[22] B. Bertholon, S. Varrette, and P. Bouvry. 2011. Certicloud: A Novel TPM-based Approach to Ensure Cloud IaaS Security. In *Proc. of the 4th IEEE International Conference on Cloud Computing (CLOUD 2011)*. Washington, DC, USA.

[23] M. Burrows, M. Abadi, and R. Needham. 1990. A Logic of Authentication. *ACM Transactions on Computer Systems* 8, 1 (February 1990), 18–36.

[24] A. Cavalli, T.-D. Cao, W. Mallouli, E. Martins, A. Sadovykh, S. Salva, and F. Zaïdi. 2010. WebMov: A Dedicated Framework for the Modelling and Testing of Web Services Composition. In *Proc. of the 8th IEEE International Conference on Web Services (ICWS 2010)*. Miami, FL, USA.

[25] Y. Chen, J. Huang, C. Lin, and J. Hu. 2015. A partial selection methodology for efficient qos-aware service composition. *IEEE Transactions on Services Computing* 8, 3 (2015), 384–397.

[26] T.S. Chow. 1978. Testing Software Design Modeled by Finite-State Machines. *IEEE Transactions on Software Engineering* 4, 3 (May 1978), 178–187.
[27] L. Chung and J.C.P. Leite. 2009. Conceptual Modeling: Foundations and Applications. In *On Non-Functional Requirements in Software Engineering*, A.T. Borgida, V.K. Chaudhri, P. Giorgini, and E.S. Yu (Eds.). Springer-Verlag, Berlin, Heidelberg, 363–379.
[28] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos. 2000. *Non-Functional Requirements in Software Engineering, vol. 5.* Springer, Heidelberg.
[29] M.R. Clarkson and F.B. Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.
[30] E. Damiani, C.A. Ardagna, and N. El Ioini. 2009. *Open source systems security certification.* Springer, New York, NY, USA.
[31] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. 2004. Web Services on Demand: WSLA-driven Automated Management. *IBM Systems Journal* 43, 1 (January 2004), 136–158.
[32] A.V. Dastjerdi and R. Buyya. 2014. Compatibility-aware Cloud Service Composition Under Fuzzy Preferences of Users. *IEEE Transactions on Cloud Computing* 2, 1 (January 2014), 1–13.
[33] A. Datta, J. Franklin, D. Garg, L. Jia, and D. Kaynar. 2011. On adversary models and compositional security. *IEEE Security & Privacy* 9, 3 (2011), 26–32.
[34] S. Deng, H. Wu, D. Hu, and J.L. Zhao. 2016. Service selection for composition with QoS correlations. *IEEE Transactions on Services Computing* 9, 2 (2016), 291–303.
[35] S. Dustdar and P. Fenkam. 2004. Formally designing web services for mobile team collaboration. In *Proc. of the 30th Euromicro Conference 2004.* Rennes, France.
[36] M. Egea, K. Mahbub, G. Spanoudakis, and M.R. Vieira. 2015. A certification framework for cloud security properties: the monitoring path. In *Accountability and Security in the Cloud*, M. Felici and C. Fernandez-Gago (Eds.). Springer, 63–77.
[37] R. Focardi and R. Gorrieri. 2004. Classification of Security Properties (Part II: Network Security). In *Foundations of Security Analysis and Design*, R. Focardi and R. Gorrieri (Eds.). Springer Berlin / Heidelberg.
[38] L. Frantzen, J. Tretmans, and T.A.C. Willemse. 2006. A Symbolic Framework for Model-Based Testing. In *Proc. of the 6th International Workshop on Formal Approaches to Testing and Runtime Verification (FATES/RV 2006).* Seattle, WA, USA.
[39] A. Fuchs and S. Gürgens. 2013. Preserving confidentiality in component compositions. In *Proc. of the International Conference on Software Composition (SC 2013).* Budapest, Hungary.
[40] H. Gao and Y. Li. 2011. Generating Quantitative Test Cases for Probabilistic Timed Web Service Composition. In *Proc. of the IEEE Asia-Pacific Services Computing Conference (APSCC 2011).* Jeju, Korea.
[41] D. Garg, J. Franklin, D. Kaynar, and A. Datta. 2010. Compositional system security with interface-confined adversaries. *Electronic Notes in Theoretical Computer Science* 265 (September 2010), 49–71.
[42] C.S. Gordon, M.D. Ernst, D. Grossman, and M.J. Parkinson. 2017. Verifying Invariants of Lock-Free Data Structures with Rely-Guarantee and Refinement Types. *ACM Transactions on Programming Languages and Systems* 39, 3, Article 11 (May 2017), 54 pages.
[43] B. Grobauer, T. Walloschek, and E. Stocker. 2011. Understanding Cloud Computing Vulnerabilities. *IEEE Security & Privacy* 9, 2 (March-April 2011), 50–57.
[44] Z. Haibo and D. Prashant. 2009. Towards Automated RESTful Web Service Composition. In *Proc. of the IEEE International Conference on Web Services (ICWS 2009).* Los Angeles, CA, USA.
[45] I.J. Hayes, C.B. Jones, and R.J. Colvin. 2013. *Reasoning about concurrent programs: Refining rely-guarantee thinking.* Computing Science, Newcastle University.
[46] D.S. Herrmann. 2002. *Using the Common Criteria for IT security evaluation.* Auerbach Publications.
[47] W. Hummer, P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. 2011. VRESCo–Vienna Runtime Environment for Service-oriented Computing. In *Service Engineering*, S. Dustdar and F. Li (Eds.). Springer, 299–324.
[48] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, and C.-H. Chen. 2008. Dynamic Web Service Selection for Reliable Web Service Composition. *IEEE Transactions on Services Computing* 1, 2 (April 2008), 104–116.
[49] C. Irvine and T. Levin. 1999. Toward a taxonomy and costing method for security services. In *Proc. of the 15th Annual Conference on Computer Security Applications (ACSAC 1999).* Phoenix, AZ, USA.
[50] F. Kerschbaum and P. Robinson. 2009. Security architecture for virtual organizations of business web services. *Journal of Systems Architecture (JSA)* 55, 4 (April 2009), 224–232.
[51] K.M. Khan, A. Erradi, S. Alhazbi, and J. Han. 2012. Security oriented service composition: A framework. In *Proc. of the 8th International Conference on Innovations in Information Technology (IIT 2012).* Al Ain, UAE.
[52] K.M. Khan and Q. Malluhi. 2010. Establishing Trust in Cloud Computing. *IT Professional* 12, 5 (September-October 2010), 20–27.

[53] J.M. Ko, C.O. Kim, and I.-H. Kwon. 2008. Quality-of-service Oriented Web Service Composition Algorithm and Planning Architecture. *Journal of Systems Software* 81, 11 (November 2008), 2079–2090.
[54] D. Kourtesis, E. Ramollari, D. Dranidis, and I. Paraskakis. 2010. Increased reliability in SOA environments through registry-based conformance testing of Web services. *Production Planning & Control* 21, 2 (June 2010), 130–144.
[55] M. Krotsiani, G. Spanoudakis, and C. Kloukinas. 2015. Monitoring-Based Certification of Cloud Service Security. In *Proc. of the International Symposium on Secure Virtual Infrastructures, Cloud and Trusted Computing 2016 (C&TC 2015)*. Rhodes, Greece.
[56] M. Lallali, F. Zaidi, A. Cavalli, and I. Hwang. 2008. Automatic Timed Test Case Generation for Web Services Composition. In *Proc. of the 6th IEEE European Conference on Web Services (ECOWS 2008)*. Dublin, Ireland.
[57] A. Landro. 2013. *The benefits of cloud-based BPM.* http://tinyurl.com/hk9jy9g.
[58] A.L. Lemos, F. Daniel, and B. Benatallah. 2016. Web Service Composition: A Survey of Techniques and Tools. *Comput. Surveys* 48, 3 (February 2016), 33:1–33:41.
[59] B. Li, D. Qiu, H. Leung, and D. Wang. 2012. Automatic Test Case Selection for Regression Testing of Composite Service Based on Extensible BPEL Flow Graph. *Journal of Systems Software* 85, 6 (June 2012), 1300–1324.
[60] H. Liang, X. Feng, and M. Fu. 2014. Rely-Guarantee-Based Simulation for Compositional Verification of Concurrent Program Transformations. *ACM Transactions on Programming Languages and Systems* 36, 1, Article 3 (March 2014), 3:1–3:55 pages.
[61] Sebastian Lins, Pascal Grochol, Stephan Schneider, and Ali Sunyaev. 2016. Dynamic Certification of Cloud Services: Trust, but Verify! *IEEE Security & Privacy* 14, 2 (2016), 66–71.
[62] S. Lins, S. Schneider, and A. Sunyaev. 2016. Trust is Good, Control is Better: Creating Secure Clouds by Continuous Auditing. *IEEE Transactions on Cloud Computing* PP, 99 (2016), 1–1.
[63] Yutu Liu, Anne H. Ngu, and Liang Z. Zeng. 2004. QoS Computation and Policing in Dynamic Web Service Selection. In *Proc. of the 13th International World Wide Web Conference on Alternate Track Papers & Posters (WWW Alt. 2004)*. New York, NY, USA.
[64] W. Lu, X. Hu, S. Wang, and X. Li. 2014. A Multi-Criteria QoS-aware Trust Service Composition Algorithm in Cloud Computing Environments. *International Journal of Grid & Distributed Computing* 7, 1 (2014), 77–88.
[65] H. Ma, F. Bastani, I.-L. Yen, and H. Mei. 2013. QoS-driven service composition with reconfigurable services. *IEEE Transactions on Services Computing* 6, 1 (April 2013), 20–34.
[66] H. Mantel. 2000. Possibilistic definitions of security-an assembly kit. In *Proc. of the 13th IEEE Computer Security Foundations Workshop (CSFW 2000)*. Cambridge, UK.
[67] R. Mateescu and S. Rampacek. 2008. Formal Modeling and Discrete-Time Analysis of BPEL Web Services. In *Advances in Enterprise Engineering I*, J.L.G. Dietz (Ed.). Lecture Notes in Business Information Processing, Vol. 10. Springer Berlin Heidelberg, 179–193.
[68] B. Medjahed, A. Bouguettaya, and A.K. Elmagarmid. 2003. Composing Web services on the Semantic Web. *The VLDB Journal* 12, 4 (November 2003), 333–351.
[69] S. Mouchawrab, L.C. Briand, Y. Labiche, and M. Di Penta. 2011. Assessing, Comparing, and Combining State Machine-Based Testing and Structural Testing: A Series of Experiments. *IEEE Transactions on Software Engineering* 37, 2 (March 2011), 161–187.
[70] A. Munoz and A. Mana. 2013. Bridging the GAP between Software Certification and Trusted Computing for Securing Cloud Computing. In *Proc. of the 9th IEEE World Congress on Services (SERVICES 2013)*. Santa Clara, CA, USA.
[71] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. 2006. *Web Services Security: SOAP Message Security 1.1.* OASIS. http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.
[72] M. Najafi, K. Sartipi, and N. Archer. 2018. *Formal Verification and Validation of Web Service Composition Certifier.* http://www.cas.mcmaster.ca/~najafm/journal4.pdf, Accessed in date August 2018.
[73] Y. Ni, S.-S. Hou, L. Zhang, J. Zhu, Z.J. Li, Q. Lan, H. Mei, and J.-S. Sun. 2013. Effective Message-Sequence Generation for Testing BPEL Programs. *IEEE Transactions on Services Computing* 6, 1 (April 2013), 7–19.
[74] OMG 2011. *Business Process Model and Notation (BPMN) – Version 2.0.* OMG. http://www.omg.org/spec/BPMN/2.0/PDF/.
[75] OpenText 2016. *BPM in the Cloud.* OpenText. http://tinyurl.com/jfx4pn5, Accessed in date October 2016.
[76] Oracle 2015. *Oracle Process Cloud Service.* Oracle. http://tinyurl.com/jj3skoa.
[77] C. Pautasso. 2009. RESTful Web service composition with BPEL for REST. *Data & Knowledge Engineering* 68, 9 (2009), 851–866.
[78] PCI Security Standards Council 2015. *Payment Card Industry (PCI) Data Security Standard – Requirements and Security Assessment Procedures – Version 3.1.* PCI Security Standards Council. http://csrc.nist.gov/publications/secpubs/rainbow/std001.txt.
[79] S. Pearson. 2011. Toward Accountability in the Cloud. *IEEE Internet Computing* 15, 4 (July-August 2011), 64–69.
[80] C. Peltz. 2003. Web services orchestration and choreography. *Computer* 36, 10 (October 2003), 46–52.

[81] L. Pino and G. Spanoudakis. 2012. Finding Secure Compositions of Software Services: Towards A Pattern Based Approach. In *Proc. of the 5th IFIP Internation Conference on New Technologies, Mobility & Security (NTMS 2012)*. Istanbul, Turkey.
[82] A. Pnueli. 1977. The temporal logic of programs. In *Proc. of the 18th Annual Symposium on Foundations of Computer Science (SFCS 1977*. Washington, DC, USA.
[83] H. Rasheed. 2014. Data and Infrastructure Security Auditing in Cloud Computing Environments. *International Journal of Information Management* 34, 3 (June 2014), 364–368.
[84] S. Rossi. 2010. Model Checking Adaptive Multilevel Service Compositions. In *Proc. of the International Conference on Formal Aspects of Component Software (FACS 2010)*.
[85] D. Sanán, Y. Zhao, Z. Hou, F. Zhang, A. Tiu, and Y. Liu. 2017. CSimpl: A Rely-Guarantee-Based Framework for Verifying Concurrent Programs. In *Proc. of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2017)*. Uppsala, Sweden.
[86] W. She, I.-L. Yen, B. Thuraisingham, and E. Bertino. 2013. Security-Aware Service Composition with Fine-Grained Information Flow Control. *IEEE Transactions on Services Computing* 6, 3 (July 2013), 330–343.
[87] P. Stephanow, G. Srivastava, and J. Schutte. 2016. Test-based cloud service certification of opportunistic providers. In *Proc. of the 9th IEEE International Conference on Cloud Computing (CLOUD 2016)*. San Francisco, CA, USA.
[88] H.N. Talantikitea, D. Aissanib, and N. Boudjlidac. 2009. Semantic Annotations for Web Services Discovery and Composition. *Computer Standards and Interfaces* 31, 6 (November 2009), 1108–1117.
[89] W. Tan, Y. Fan, and M.C. Zhou. 2009. A Petri Net-Based Method for Compatibility Analysis and Composition of Web Services in Business Process Execution Language. *IEEE Transactions on Automation Science and Engineering* 6, 1 (January 2009), 94–106.
[90] H. Tout, A. Mourad, H. Yahyaoui, C. Talhi, and H. Otrok. 2012. Towards a BPEL model-driven approach for Web services security. In *Proc. of the 10th Annual International Conference on Privacy, Security and Trust (PST 2012)*. Paris, France.
[91] W.-T. Tsai, P. Zhong, J. Balasooriya, Y. Chen, X. Bai, and J. Elston. 2011. An approach for service composition and testing for cloud computing. In *Proc. of the 10th International Symposium on Autonomous Decentralized Systems (ISADS 2011)*. Kobe, Japan.
[92] USA Department of Defence 1985. *Department Of Defense Trusted Computer System Evaluation Criteria*. USA Department of Defence. http://csrc.nist.gov/publications/secpubs/rainbow/std001.txt.
[93] E. van Veenendaal. 2018. *Standard glossary of terms used in Software Testing Version 2.2*. International Software Testing Qualifications Board. http://www.astqb.org/educational-resources/glossary.php, Accessed in date August 2018.
[94] H.D. Vo, D.C. Phung, V.Q. Dung, and V.-H. Nguyen. 2012. Securing Data in Composite Web Services. In *Proc. of the 4th International Conference on Knowledge and Systems Engineering (KSE 2012)*. Danang, Vietnam.
[95] J. Yu, J. Han, S.O. Gunarso, and S. Versteeg. 2013. A Business Protocol Unit Testing Framework for Web Service Composition. In *Proc. of the 25th International Conference on Advanced Information Systems Engineering (CAiSE 2013)*. Valencia, Spain.
[96] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. 2004. QoS-aware middleware for Web services composition. *IEEE Transactions on Software Engineering* 30, 5 (May 2004), 311–327.
[97] H. Zheng, J. Yang, and W. Zhao. 2016. Probabilistic QoS Aggregations for Service Composition. *ACM Transactions on the Web* 10, 2, Article 12 (May 2016), 12:1–12:36 pages.