Future Internet **2012**, *4*, 92-109; doi:10.3390/fi4010092

OPEN ACCESS

future internet

ISSN 1999-5903

www.mdpi.com/journal/futureinternet

Article

Web Service Assurance: The Notion and the Issues

Marco Anisetti [1], Claudio A. Ardagna [1], Ernesto Damiani [1], Fulvio Frati [1,⋆], Hausi A. Müller [2] and Atousa Pahlevan [2]

[1] Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, via Bramante 65–26013 Crema (CR), Italy; E-Mails: marco.anisetti@unimi.it (M.A.); claudio.ardagna@unimi.it (C.A.A.); ernesto.damiani@unimi.it (E.D.)

[2] Department of Computer Science, University of Victoria, STN CSC, Victoria, BC V8W 3P6, Canada; E-Mails: hausi@cs.uvic.ca (H.A.M.); atousa.p@gmail.com (A.P.)

⋆ Author to whom correspondence should be addressed; E-Mail: fulvio.frati@unimi.it; Tel.: +39-0373-898064; Fax: +39-0373-898010.

Received: 21 December 2011; in revised form: 16 January 2012 / Accepted: 3 February 2012 / Published: 14 February 2012

Abstract: Web service technology provides basic infrastructure for deploying collaborative business processes. Web Service security standards and protocols aim to provide secure communication and conversation between service providers and consumers. Still, for a client calling a Web service it is difficult to ascertain that a particular service instance satisfies—at execution time—specific non-functional properties. In this paper we introduce the notion of certified Web service assurance, characterizing how service consumers can specify the set of security properties that a service should satisfy. Also, we illustrate a mechanism to re-check non-functional properties when the execution context changes. To this end, we introduce the concept of *context-aware certificate*, and describe a dynamic, context-aware service discovery environment.

Keywords: assurance; context-aware service certification; service discovery; service security; web service certification

---

1. Introduction

The goal of software assurance is to provide to software purchasers and users the justifiable confidence that the software will consistently exhibit some desired non-functional properties, including security-related ones. The expanding portfolio of Web services available over the global network has the welcome effect of enabling on-the-fly service selection, but at the cost of increased security and reliability risks with respect to ordinary components—off-the-shelf or developed in-house. For this reason, a number of assurance tools have been proposed, aimed at monitoring the delivery of Web Services and remotely diagnosing and resolving Web Service issues. Generally speaking, assurance tools consolidate and correlate the measurements of some aspects of the Quality of Experience (QoE) perceived by a Web service user. Such measurements are then stored into a database for further processing, including data analysis and visualization. Today, most commercial Web service engines, a.k.a. *containers*, such as WSO2 (http://wso2.com), IBM WebSphere (http://www.ibm.com/software/websphere), OpenESB (http://java.net/projects/openesb), Apache AXIS2 (http://axis.apache.org/axis2), Glassfish Metro (http://metro.java.net) support Web service assurance in some way.

Security assurance, that is, assurance aimed at monitoring security and dependability properties, is a less established subject. Well-known Web security standards, such as the WS-* stack [1], implement secure transactions, conversation, and access control systems. However, these standards do not include any provisions for Web service assurance; rather, they are aimed at satisfying pre-defined security requirements that service designers identify at development time. Some well-known packages for service security, such as Apache Rampart (http://axis.apache.org/axis2/java/rampart/) and Java WSIT (http://wsit.java.net), provide basic facilities for monitoring Web service security mechanisms; however, few if any assurance techniques based on such facilities have been proposed in the literature.

This paper explores the idea that machine-readable certificates of security and privacy properties of Web services can provide a way to bound, transfer, and alleviate security risks at run time. In 2008, the Software Engineering Institute (SEI) published a requirements document on the service certification process for the U.S. Army's Chief Information Office/G-6 (CIO/G-6) organization that takes charge of the information management function of the Army to address security and provisioning concerns the Army foresees in its development of Service-Oriented Architecture (SOA) environments [2]. This document presents a methodology for certifying Web services to assure their compliance with stated non-functional requirements. This work represents a first step in defining methods and techniques to certify that the service satisfies specific security requirements, producing security assertions and documents that could be accepted as a proof of service assurance. It is important to note that certification of Web service security properties involves both the container and the code implementing the service itself. Even if a Web service container is certified to support, say, the confidentiality of data flowing through a service interface, no inference can be done on the internal implementation of the service, where information leaks could happen.

Damiani *et al.* introduced container- and service-level security properties [3]. Their work aims to support the definition and run-time handling of certificates stating that a set of security properties are satisfied by Web services deployed in a given container. However, handling context changes involving the container, the underlying hardware, or the service itself remains an open issue.

This paper is organized as follows: Section 2 gives an introduction on how to express security requirements. Section 3 discusses our approach to security certification of services and presents different types of certificates. Section 4 describes the concept of Adaptable Security Certification using a practical example and introducing a conceptual architecture for adaptable assurance-based service discovery. Section 5 presents related work and Section 6 concludes the paper.

2. Expressing Service Security Requirements

Any *assurance process* consists of a set of activities aimed at increasing the users' confidence that a given service will satisfy their functional and non-functional requirements. Functional requirements are already considered in the context of dynamic Web service search and discovery, while performance assurance can be addressed via suitable Service Level Agreements (SLAs). Other non-functional properties, such as security and reliability, are usually not addressed. Therefore, there is a need for a solution that (*i*) permits to specify security requirements that a given service must satisfy [4,5], and (*ii*) uses these security requirements in the context of Web service search and discovery.

There are two key aspects to the notion of *service security requirements*: (1) security requirements describe a set of security properties whose semantics are shared between the service supplier and the service user; and (2) security requirements specify the process and techniques (evidence) used to verify that these security properties hold for this service.

The first building block to support an assurance process for service security is the definition of the set of security properties that are of interest. The literature defines several classes of security properties including *authenticity*, *integrity*, and *confidentiality* [5]. In a Service-Oriented Architecture (SOA), security properties may concern message-level security (*i.e.*, security of data in transit) and service-level security evaluating the service implementation (*i.e.*, security of data at rest). In other words, we distinguish between security properties that can be proven to hold for a given service at container-level and those that need to be certified on the real service implementation. The process of evaluating a service at container-level considers the compliance of the container with Web service security standards (e.g., WS-Security [6] and WS-SecureConversation [7]) and the correct enforcement of security policies (e.g., WS-Policy [8]) defined by the service provider, to prove that the security properties are supported by the service deployed in the container. In contrast, service-level security considers security properties that need be proven by analyzing and evaluating the real service implementation. Service-level properties complement the container-level ones by providing an overview of the service that goes beyond the service interface, and considers its implementation and the developed operations.

Here, we distinguish between *abstract properties*, as for instance confidentiality and integrity, and *concrete properties* that enhance abstract ones by adding *class attributes* [9]. Indeed, no security property is entirely specified without an adversarial model, that is, a description of what an attacker can do to compromise the property. Therefore our class attributes represent (1) the threats against the property to hold (e.g., eavesdropping or replay attack), and (2) the security functions that ensure that the property holds (e.g., the access control system) [9].

Security properties are organized in an abstraction hierarchy (cf. Section 3.2). Each node in the hierarchy is a security property of the form $p = (\hat{p}, A)$, where $\hat{p}$ is an abstract property and A is the set of

class attributes in the form $a = v$, with a the attribute name and v its value. The first-level nodes of the hierarchy are represented by all security properties $p = (\hat{p}, -)$, namely abstract security properties with no class attributes specified. An ordering relation $\preceq$ between properties p_i and p_j ($p_i \preceq p_j$) indicates that p_j is a specification of p_i, if and only if $p_i \cdot \hat{p} = p_j \cdot \hat{p}$ and for each class attribute $a \in A$, $p_j \cdot a$ dominates $p_i \cdot a$.

Upon the specification of security properties, an important aspect of the assurance process is the definition of requirements over the mechanisms used to provide evidence for those properties. We can verify the service support for a given property using two different approaches: *a test-based approach* providing evidence that a test carried out on the software has given a certain result, and *a formal approach* providing evidence based on an abstract model of the service. In this paper, we focus on test-based evidence: each security property can be associated with one or more classes of tests, which in turn contain a set of test types (e.g., equivalence partitioning, fuzzy/mutation) used to generate the test cases providing the evidence that the property is supported by the service. As for security properties, test-based evidence (including the results of test execution) is stored in a certificate awarded to a service and can be compared with users' requirements, to search and discover services on the basis of their non-functional characteristics.

Section 3.1 presents our approach to service security certification using a test-based mechanism for certifying that a service possesses a concrete security property that can be integrated into the lightweight Web service discovery technique presented in Section 4.

3. Security Certification of Services

Our approach relies on a model-based testing certification solution for services that produces security assurance metadata (*i.e.*, *test-based evidence*) supporting a given security property for the service. The test-based solution used in this paper requires the generation and execution of suitable test cases on the service, starting from a formal model of the service itself. The test-based mechanisms used to certify a service and the related results compose the certificate evidence.

3.1. Model-Based Testing

Model-based testing targets the representation and certification of complex Web Services. Services can be modeled as finite state automata and transition systems for the automatic generation of test cases, and the evaluation of correctness of tested services [10,11]. Keum *et al.* [10], authors proposed Symbolic Transition System (STS) [12] as a suitable solution for the certification of complex Web services that involves communications, allowing the specification of typed variables, guards (*i.e.*, constraints on state transitions), and actions (*i.e.*, function calls).

Anisetti *et al.* [9] proposed an extended version of STS suitable for generating tests based on container and service behavior. Specifically, Web Service Description Language (WSDL) specification is used for the creation of STS models. The definition of service interface, describing operations and input and output data models can be exploited to build a specific model describing three states for each described operation:

- *initial state*, no inputs have been received;
- *intermediate state*, the inputs have been received but the outputs have not been produced yet;
- *final state*, the outputs have been generated and returned to the counterpart.

The modeling can also be extended to the Web Services Conversation Language (WSCL) specification, where WS-Conversation policies and constraints are defined and applied. In particular, WSCL regulates the communications between users and services, which could be modeled in STS to take into consideration data exchanges [9]. Also, the WSCL specification can be extended to integrate implementation details describing the internal functioning of the services.

Finally, the model can handle container certifications; STS service models can be extended to include interactions between the customer and the service, which are outside the service implementation but part of the security specification flow (e.g., key exchange for data integrity).

A more lightweight approach was introduced by Pahlevan *et al.* [13]. They model the behavior of the Web services and their communications employing constraints to automate and evaluate the correctness of the architecture and ultimately the reliability of the designed system (e.g., data validity and correctness). This approach can be extended to support a wide class of non-functional properties including security properties.

3.2. Security Assurance Certificates

When test-based certification is used, test cases and related evidence can be attached to Web service interfaces in order to evaluate the assurance level. According to Damiani *et al.*, three different certificate types can be identified, each one characterized by specific metadata and a certification process [14].

Self-certificate (SC): SCs are characterized by a four-way interaction between service provider (SP) and consumer (C). First of all, C sends a request to SP specifying the list of security properties to be certified on the service. SP already mapped to each service consistent test suites, including functional and QoS-based test cases; those tests are then used to build the reply, sending to C the test cases related to the specified set of properties. If the reply is satisfactory, C can directly execute the test cases on the service and analyze the results. It is important to note that this mechanism does not require a trust relationship between C and SP; however, it reduces the number of actors involved in the certification process and, therefore, the certification time.

Lightweight Certificate (LC): This type of certificate introduces a new actor in the certification process, namely a Third-Party Certifier ($3PC$). In this case, all tests are executed by the $3PC$. A consumer C can send a request to $3PC$ specifying the requested security properties, and a list of candidate providers. Then, $3PC$ can contact each specific provider, supply the test cases, and apply them to services. $3PC$ can also play the role of certificate repository, storing available certificates for future usage.

Collaborative Certificate (CCert): Certificates are generated and stored independently from C requests. An extension of the UDDI protocol could support CCert, as well as the storing and managing of test suites. Test suites are signed by SP during the service registration phase; then

$3PC$ can access them periodically to verify the test results, to generate missing certifications, and to invoke one or more tests as needed to reconfirm or strengthen service quality.

In the remainder of this paper, we further develop the CCert scenario to enable classifying services based on their level of assurance. We adopt this well-established approach [14] to provide users with a subset of certified services. Then, we show how the CCert scenario can be implemented, using our approach so that real, practical implementations can properly meet user needs.

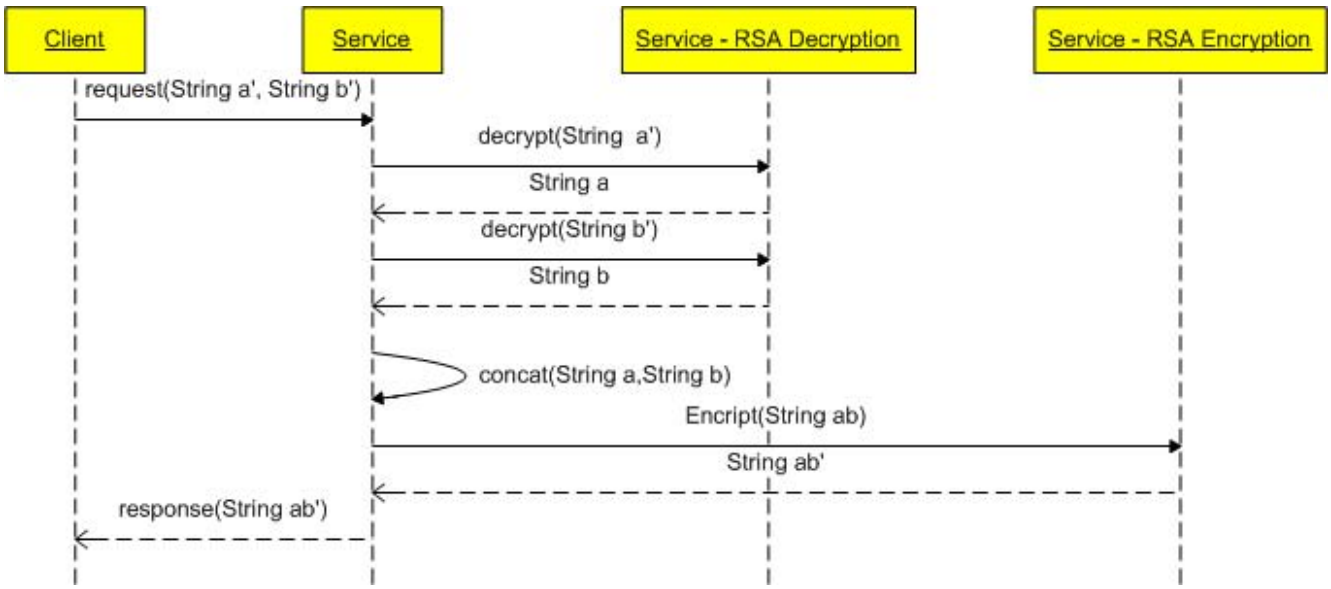
4. Selection of Services

We now focus on the practical challenges arising when implementing a Web Service Discovery (WSD) system that considers certificates for security assurance. When security properties of individual Web services are certified, changes in the context may require re-certifying them at run-time in a context-agnostic manner. In this section, we illustrate the challenges arising in our context with a simple example and propose a first conceptual architecture for adaptable assurance-based service discovery.

4.1. An Example of Service Discovery and Dynamic Certification

Let us assume that a web farm is supplying a secure storage service based on an *ad hoc* security framework. The service is tested in a given context (*i.e.*, on a physical machine), and CCerts are issued (*i.e.*, static certification), containing evidence that all the tests have passed successfully (cf. Section 3). For the sake of simplicity, we consider a simple service implementing the concatenation of two strings, whose sequence diagram is presented in Figure 1.

Figure 1. String concatenation service sequence diagram.



The service first receives as input the two RSA encrypted strings (*String a'* and *String b'*), then decrypts the strings (*String a* and *String b*) concatenating them (*String ab*), and finally encrypts the concatenated response using the RSA algorithm (*String ab'*). Such encryption is performed by the service itself and not by the container. The example can be easily reformulated in the case where the

container performs SOAP encryption according to WS-Security. In this scenario, it is the container that, using an ad-hoc component (e.g., Apache Rampart), is responsible for encrypting and decrypting the input strings and their concatenated version. In the following we consider the first scenario in which encryption and decryption are managed by the service implementation.

Test cases in the service certificate assert the confidentiality of the response data by providing the encrypted response string, the key, and the input parameters used to compute the response. Suppose that the test suite includes some critical input parameters that may generate a service response too big for the local RSA implementation to encrypt. In fact, in the case of very long strings, their concatenation could exceed the number of characters accepted by the RSA implementation; furthermore, different systems could manage very long strings differently, having unpredictable results at client-side. In particular, those tests are to guarantee that no data leakage happens during the execution on the server, and that the information is secure at the application level during the entire exchange of messages between the clients and the service. Each time a user searches for a secure storage server certified for the "Confidentiality" property having the class attribute *MsgEncryption* set to "RSA", the repository will return the certificate along with the property and the evidence of the tests applied to the service interface (*i.e.*, triples <*encryptedtext*, *key*, *plaintext*>).

The above discussion assumes static certification. However, there are cases in which changes in the service context and environment invalidate the certificates awarded to the service. As an example, consider a service provider rationalizing its software infrastructure, moving its storage service to the cloud. The service is instanced in a dedicated virtual machine and, from that moment on, the certification is no longer valid. In fact, moving the architecture to a virtualized system has added a new virtualization layer to it, and the encryption/decryption takes place on the same physical server hosting other containers and services. In this situation when a user asks for a string concatenation service with the confidentiality property on parameters, the original certificate can no longer be considered since the context has changed. The provider can dynamically re-create the certificate re-running the test suite enclosed in the certificate evidence, and obtain a certificate for the new context. Alternatively, re-certification can be carried out by the service user, executing the test suite in the old certificate in the new context. In principle, server-side re-certification may be preferable for two main reasons: (*i*) the re-certification process could be too heavy for common client infrastructures; (*ii*) online re-certification can represent a security threat for the service.

The same idea can be applied in cases where services are dynamically moved from a server to another, or aggregated on the same virtual machine to optimize resources. Notably, not only changes in the hosting infrastructure require service re-certification. In fact, users can change their infrastructure thus introducing the need of a new selection process or at least of service re-certification. For instance, doing online banking in a different workspace such as mobile, owned, or shared computers may change the required authentication method and encryption algorithm.

In summary, to improve service certification and selection at run time, it is important to consider the context and environment in which the services are deployed and to handle context-aware re-certification for WSD. This idea is complementary to the approach by Anisetti *et al.* which mainly deals with static certification [9].

4.2. Adaptable WSD Security Certification

The above example is a good starting point towards dynamic and adaptable WSD based on security certification. Adaptable service discovery aims to recognize which security properties need to be certified to support the dynamic selection of services in a context-agnostic manner. Moreover, it provides a mechanism that enables service consumers to define their preferences in terms of certified properties, evidence, and tests, and automatically matches them against the certificates awarded to a service at discovery time.

The service security properties that are used for an adaptable WSD security certification can be further broken down into the following groups.

- *Abstract Security Property (ASP)*: An abstract security property represents a generic security requirement for the service, such as, confidentiality, integrity, and authentication. It can also be referred as a concrete security property with no class attributes.

- *Concrete Security Property (CSP)*: An ASP enhanced with class attributes. Given two instances of CSP, p_i and p_j, based on the same abstract property $\hat{p}$, p_j is a specialization of p_i, if a certificate proving p_j always proves p_i. For instance, given the abstract property *integrity*, and two concrete properties p_1 = (*integrity*,$\{algorithm = RSA, |key| = 1024bits\}$) and p_2 = (*integrity*,$\{algorithm = RSA, |key| = 2048bits\}$), a certificate proving p_2 always proves p_1. The relation between p_1 and p_2 is called intra-property relation, because it involves properties with the same ASP and only considers the class attributes.

- *Semantic Security Property (SSP)*: An SSP is a concrete security property. The only difference between CSPs and SSPs is that the latter refers to order and equivalence relations, called inter-property relation, involving different abstract properties. Inter-property relations are defined based on expert knowledge. As an example, given the two properties p_1 = ($authenticity$,$\{\}$) and p_2 = ($non\text{-}repudiation$,$\{\}$), p_2 implies p_1 meaning that each certificate for property p_2 also applies to property p_1. ASP, CSP, and SSP form a hierarchy of security properties.

- *Domain Security Property (DSP)*: A domain-aware security property specification. Since a hierarchy characterization of a security property could be different in different domains (both intra- and inter-property relations), a DSP enables an accurate evaluation of the security properties relevant for a given domain. This alleviates the problem of having a complex hierarchy of security properties that is suitable for all domains. Fragments of the hierarchies for *Domain X and Y* are depicted in Figures 2 and 3, respectively.

After defining different categories of security properties, we need to integrate the certification process into the WSD process. To realize this, we first introduce the concept of Assurance-Level Agreement (ALA), as the part of the Service Level Agreement (SLA) that exposes the security properties supported by a given service in the form of machine-readable certificates. The user has then to define its requirements in a *policy*, that is, a machine-readable format of its security requirements. Finally, WSD matches them against certificates to provide a list of compatible services. Each certificate is an XML-based document that stores and manages security properties and test evidence. Certificates can

be attached to Web service interfaces in order to provide the assurance information (*i.e.*, artifacts and evidence) to be matched during the discovery process.

Figure 2. An example of a hierarchy of security properties for Domain X.

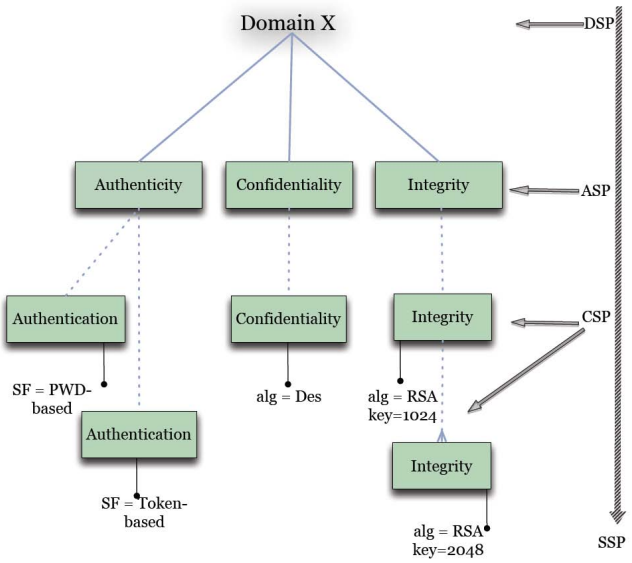
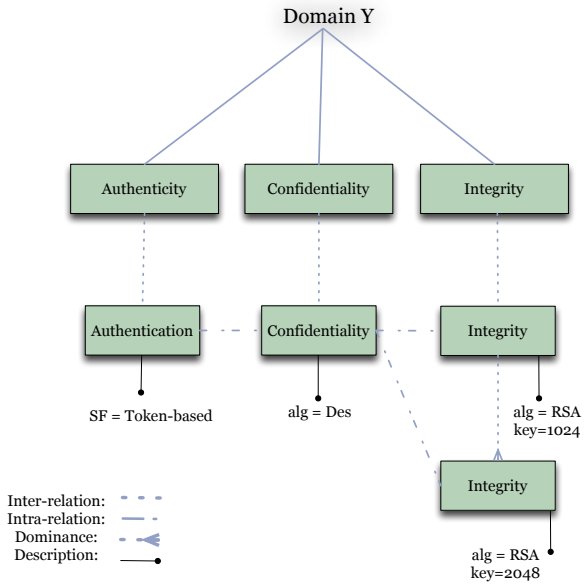


Figure 3. An example of a hierarchy of security properties for Domain Y.



The format of the service certificate can be specified using the XML-based schema shown in Figure 4. The service certificate includes the following main sections.

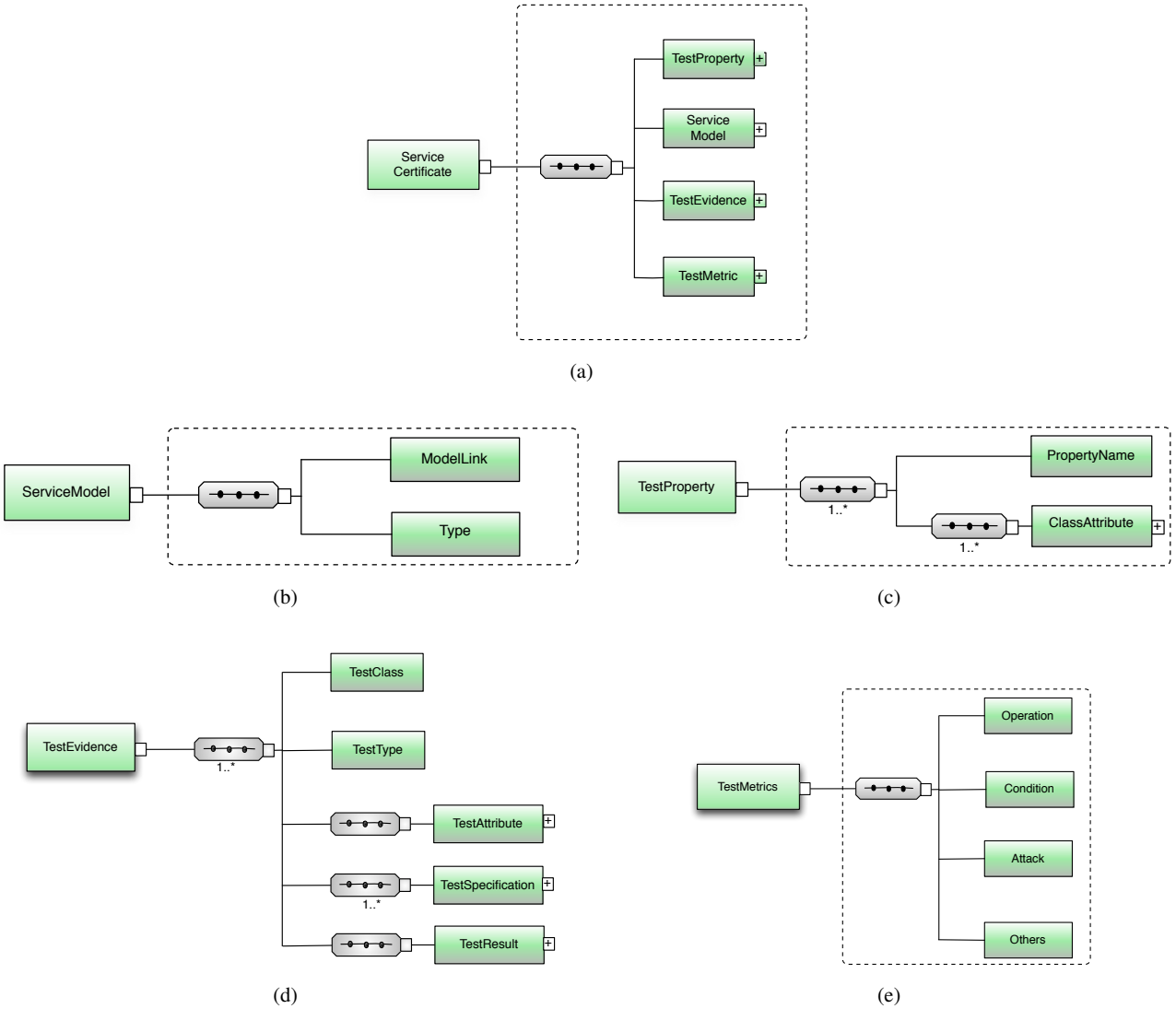
- *TestProperty*: Information about the certified security properties. Each property includes the *PropertyName* and a set of *ClassAttribute* fields.

- *ServiceModel*: A reference (*i.e.*, URI), named (*ModelLink*), to the location where the model of the service is stored, and the type of the service model (*i.e.*, WSDL-based, WSCL-based, implementation-based) is declared in the *Type* element.

- *TestEvidence*: All artifacts related to the test cases executed on the service for its certification. It includes test class (*TestClass*), type (*TestType*), attributes (*TestAttribute*), specifications (*TestSpecification*), and the result of test case execution (*TestResult*). *TestSpecification* is the specification about the real test case that is declared through test id, description, and a link to the test model that is used to generate the test case. *TestResult* is a set of pairs holding a reference to the test case and a pass-fail result.

- *TestMetrics*: A set of metrics representing the quality of the test cases executed on the service. These measurements are used in WSD matching to compare the services by representing the result of different tests on various services.

Figure 4. Certificate Schema.



The above-mentioned information is used by WSD to find the certified Web services that comply with the user policies.

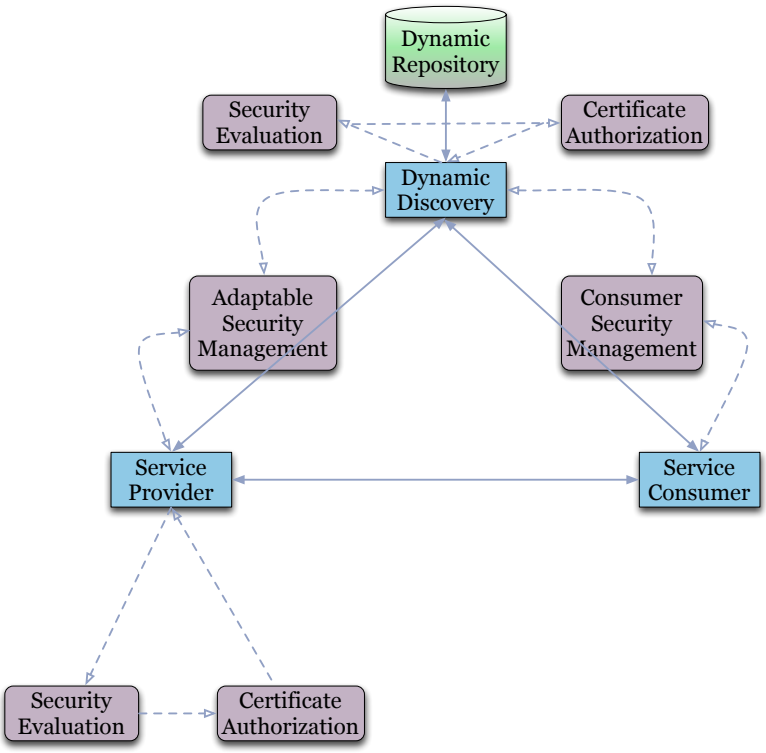
4.3. A Conceptual Architecture for an Adaptable Assurance-Based WSD

Our approach to dynamic certification of service security properties consists of two phases: *Static Certification (SC)* and *Dynamic Certification (DC)*.

Whereas the SC process involves certification of services at development time, independently from the context in which the services are deployed, the DC process supports run-time certification of services. Based on the generated service security certificates (*ServiceCertificate*), depicted in Figure 4(a), WSD provides an assurance-based selection process. Upon receiving a consumers' request with security preferences, it matches them against the *ServiceCertificate*. According to the matching results, WSD returns a set of services compatible with the users' preferences.

Figure 5 depicts the overall architecture of the adaptable assurance-based service selection, which is composed of the following components and processes.

Figure 5. Overall Architecture of Adaptable Assurance-based WSD.



Service Consumer: The party requesting access to, or integrating a remote service, according to users' preferences.

Service Provider: The party providing remote Web services that are accessed by service consumers.

Dynamic Repository: The component storing the Web services together with the security certificates awarded to them. Here, the certified services are registered and published, and periodically re-certified to assess their security properties.

Consumer Security Management: The component dealing with the consumer's security requirements and preferences. It allows service consumers to define their preferences in terms of certified properties, evidence, and tests.

Adaptable Security Management: The component enabling automated run-time service certification, beyond the security implementation and pre-deployment certification of the Web service. It monitors the properties that could hold at run-time and identifies new and old security properties that need (re-)certification. The module continuously evaluates the propriety of the security properties claimed by the service.

The run-time Web service assurance process is executed in three distinct phases as follows.

Security Evaluation: An accredited process that executes test cases for service evaluation. It generates new test cases, if needed, according to the security requirements given by the "Adaptable Security Management" component and the service security specification. If the required test evidence is not available in the service certificate, a new set of test cases is generated and executed on the service (run-time evaluation). As a result, new evidence is generated and used in the assurance process.

Certificate Authorization: Services are certified using the evidence provided by the Security Evaluation phase. It generates an evidence-based certificate guaranteeing that a set of test cases is executed on the service, or on an entire business process in a service container.

Dynamic Discovery: The conformance of the selected services with the consumer's security preferences is evaluated by means of a matching process. The latter measures the degree of compliance between users' preferences and service certificates.

In summary, run-time certificate matching allows a service consumer to ascertain that the assurance level provided by the service certificate complies with its own preferences. This solution increases consumer confidence because their assurance requirements have been met at service execution time and in the context of certification.

5. Related Work

Previously researchers have approached the problem of service verification and certification by applying two families of techniques to prove that a service possesses a given property: *test-based techniques*, which require the application of suitable tests to the service, and *formal-based techniques*, where an abstract model of the service (e.g., a set of logic formulas, or a formal computational model such as a finite state automaton) is generated and used for validation [15].

Test-based certification of software components is a time-honored software engineering problem; for instance, assurance levels from 1 to 4 of the well-known Common Criteria standard [16] are themselves test-based [15]. Collecting evidence supporting security properties differs greatly from standard software testing procedures, making existing security certification schemes not applicable in the service ecosystem. The loosely coupled nature of Web Services (as compared to traditional component-based software systems) poses strict limits on the way testers can interact with the services, making the usual stub-based testing techniques hardly applicable to Web service security testing. A major problem is that

Web service source code is usually inaccessible, and remote execution may involve a cost. In addition, existing certification techniques have been provided for static and monolithic software [16,17], where certificates are usually human-readable statements signed by a trusted certification authority, and do not suit the service requirements in terms of run-time evaluation of certificates during the discovery process.

There are two major approaches to testing a Web service [18,19]: (*i*) consider Web Services as independent software components to which traditional interface testing methods can be adapted, by considering the released interfaces and protocol bindings; (*ii*) consider Web Services as composite elements, where integration testing methodologies should be adapted. Web Services can be tested from the perspective of the different stakeholders (*i.e.*, developer, provider, integrator, certifier, and user) and there is the need to specify who can perform a test, which types of tests are needed, and what are the challenges in carrying out the tests and documenting their results. Others focused on the automatic generation of test cases starting from WSDL released by the service provider or general service specification [20–25]. For example, Noikajana and Suwannasart present a new methodology in which the Web service specification is used to generate test cases based on a decision table [23]. Moreover, an approach to testing Web services using fault-coverage to check the conformance of the Web services to their WSDL specification has been proposed by Wen-Li Dong *et al.* with the goal of automating testing [25].

Model-based certification of Web services has also been used for service representation and certification [14]. Certificates based on formal proofs deal with verifying the properties of the models of Web services [26]. A milestone is the definition of Web Services Business Process Execution Language (WSBPEL) [27], an industry standard for specifying workflows. Other industry driven modeling approaches [28,29] are based on the Unified Modeling Language (UML) [30]. A UML extension also exists and enables, with some limitations, the modeling of security properties to be provided by the system [31,32].

The problem of service verification has been extensively studied and is the topic of several research projects, highlighting its intrinsic importance in the context of the future Internet. An interesting approach is the one taken by the AVISPA (Automated Validation of Internet Security Protocols and Applications) research project [33], which defines a High Level Protocol Specification Language (HLPSL) for modeling communication and security protocols [34]. AVANTSSAR (Automated VAlidatioN of Trust and Security of Service-oriented ARchitectures) introduces a platform supporting the validation of trust and security aspects of service-oriented architectures and automated techniques to reason about services composition security [35]. One of the proposed tools, the SAT-based Model Checker [36,37], was recently used to identify a security flaw in the SAML-based Single-Sign-On protocol for Google Applications [38]. The SPaCIoS (Secure Provision and Consumption in the Internet of Services) project concentrates on the problem of providing security in a complex service ecosystem [39]. It aims to provide a solution and new generation analyzers for automated security validation of services not only at production time, but also at deployment and consumption times. SPaCIoS tries to combine technologies for penetration testing, security testing, model checking, and automatic learning. Moreover, the project ANIKETOS (Ensuring Trustworthiness and Security in Service Composition) provides service developers and providers with a secure service development framework [40]. Such a framework includes methods, tools, and security services that support the

design-time creation and runtime composition of secure services in environments where both services and threats are evolving. Finally, the project ASSERT4SOA (Advanced Security Service cERTificate for SOA) aims to define and develop a certification infrastructure dealing with both test-based and model-based certification to provide a certificate-aware SOA [41]. Gürgens and Rudolph proposed another interesting approach to find security flaws in a number of key exchange, authentication, and non-repudiation protocols [42–44]. Their approach is supported by the Simple Homomorphism Verification Tool [45].

Another important area of research analyzes the computation of test results at service invocation time and the definition of enhanced UDDI supporting QoS. Tsai *et al.* propose an enhanced UDDI server specification to manage check-in and check-out testing for services [46]. In particular, the check-in test is performed when the service is first registered in the system, while the check-out test is done when the service receives a request from a user. Ran presented a new service discovery model taking into account both functional and non-functional requirements and proposes a QoS certifier responsible to check the claims made by the service provider [47]. Serhani *et al.* illustrate an architecture based on a QoS broker for efficient Web services selection [48]. After the verification of the service by the broker, the client can select services on the basis of its QoS requirements. In this paper, we focused on security certification and on the definition of a discovery process that considers the certified properties and the way in which these properties are proven to hold.

6. Conclusions

The rapid worldwide deployment of Web services on enterprise IT infrastructure is the enabler of a new generation of applications. Web service security standards are now firmly in place, but the provision of a security architecture that can ensure that key security and privacy properties actually hold at service execution time is still an open question.

Current solutions to this problem are based on compliance to standards in a static execution context and do not address security requirements of service-oriented applications when execution contexts can change and evolve rapidly.

We posit that our context-aware certificates will enable administrators to specify their requirements in terms of security properties to be re-checked when the execution context changes. In this paper, we discussed the notion of context-aware certificates and gave a preliminary description of a dynamic, context-aware Web service discovery architecture aiming to fulfill the security requirements of enterprise applications. We believe that this dynamic certification approach can be easily adapted to the needs of specific domains such as telecommunications and healthcare. Future works will include practical experimentation of our approach and a generalization of the whole framework able to certificate Web services with respect to generic properties.

Acknowledgments

This work was funded in part by the European Commission under the project ASSERT4SOA (contract n. FP7-257351), and by the National Sciences and Engineering Research Council (NSERC) of

Canada (CRDPJ 320529-04 and CRDPJ 356154-07), IBM Corporation via the CSER Consortium, and University of Victoria, British Columbia, Canada.

References

1. Galbraith, B.; Hankinson, W.; Hiotis, A.; Janakiraman, M.; Prasad, D.V.; Trivedi, R.; Whitney, D. *Professional Web Services Security*; Wrox Press Ltd.: Birmingham, UK, 2002.
2. Software Engineering Institute. Securing Web Services for Army SOA. Available online: http://www.sei.cmu.edu/solutions/softwaredev/securing-web-services.cfm (accessed on 6 February 2012).
3. Damiani, E.; Maña, A. Toward WS-Certificate. In *Proceedings of the ACM Workshop on Secure Web Services*, Chicago, IL, USA, 13 November 2009; pp. 1–2.
4. Han, J.; Kowalczyk, R.; Khan, K. Security-oriented service composition and evolution. In *Proceedings of the 13th Asia Pacific Software Engineering Conference*, Bangalore, India, 6–8 December 2006; pp. 71–78.
5. Kim, A.; Luo, J.; Kang, M. Security ontology for annotating resources. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*; Springer: Berlin, Germany, 2005; Volume 3761, pp. 1483–1499.
6. Nadalin, A.; Kaler, C.; Monzillo, R.; Hallam-Baker, P. Web Services Security: SOAP Message Security 1.1. Available online: http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf (accessed on 6 February 2012).
7. Nadalin, A.; Goodner, M.; Gudgin, M.; Barbir, A.; Granqvist, H. WS-SecureConversation 1.3. Available online: http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.html (accessed on 6 February 2012).
8. Vedamuthu, A.; Orchard, D.; Hirsch, F.; Hondo, M.; Yendluri, P.; Boubez, T.; Yalcinalp, U. Web Services Policy 1.5 - Framework. Available online: http://www.w3.org/TR/ws-policy/ (accessed on 6 February 2012).
9. Anisetti, M.; Ardagna, C.; Damiani, E. Fine-grained modeling of web services for test-based security certification. In *Proceedings of the 8th IEEE International Conference on Services Computing*, Washington, DC, USA, 5–10 July 2011; pp. 456–463.
10. Frantzen, L.; Tretmans, J.; d. Vries, R. Towards model-based testing of web services. In *Proceedings of the International Workshop on Web Services—Modeling and Testing*, Palermo, Italy, 6 June, 2006; pp. 67–82.
11. Keum, C.; Kang, S.; Ko, I.Y.; Baik, J.; Choi, Y.I. Generating test cases for web services using extended finite dtate machine. In *Testing of Communicating Systems*; Springer: Berlin, Germany, 2006; Volume 3964, pp.103–117.
12. Frantzen, L.; Tretmans, J.; Willemse, T. Test generation based on symbolic specifications. In *Proceedings of the 4th International Workshop on Formal Approaches to Software Testing*; Springer-Verlag: Linz, Austria, 2004; Volume 3395, pp. 1–15.

13. Pahlevan, A.; Müller, H.A.; Cheng, M. A dynamic framework for quality web service discovery. In *Proceedings of the 4th International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems*; Carnegie Mellon University: Pittsburgh, PA, USA, 2010; pp. 73–89.
14. Damiani, E.; El Ioini, N.; Sillitti, A.; Succi, G. WS-Certificate. In *Proceedings of the IEEE Congress on Services, Part I*, Los Angeles, CA, USA, 6–10 July 2009; pp. 637–644.
15. Damiani, E.; Ardagna, C.; Ioini, N.E. *Open Source Systems Security Certification*; Springer: New York, NY, USA, 2009.
16. Herrmann, D. *Using the common criteria for IT security evaluation*; Boca Raton, FL, USA, 2002.
17. US Department of Defence. Department of Defense Trusted Computer System Evaluation Criteria. Available online: http://csrc.nist.gov/publications/secpubs/rainbow/std001.txt (accessed on 6 February 2012).
18. Canfora, G.; Penta, M.D. Testing services and service-centric systems: Challenges and opportunities. *IT Prof.* **2006**, *8*, 10–17.
19. Bloomberg, J. The Rational Edge Ezine for the Rational Community: Testing web services today and tomorrow. Available online: http://www.p2080.co.il/go/p2080h/files/4989377677.pdf (accessed on 6 February 2012).
20. Hanna, S.; Munro, M. An approach for specification-based test case generation for web services. In *Proceedings of the IEEE/ACS International Conference on Computer Systems and Applications*; IEEE CS: Amman, Jordan, 2007; pp. 16–23.
21. Jokhio, M.; Dobbie, G.; Sun, J. Towards specification based testing for semantic web services. In *Proceedings of the 20th Australian Software Engineering Conference*; IEEE CS: Gold Coast, Australia, 2009; pp. 54–63.
22. Mao, C. Towards a hierarchical testing and evaluation strategy for web services system. In *Proceedings of the 7th ACIS International Conference on Software Engineering Research, Management and Applications*; IEEE CS: Haikou, China, 2009; pp. 245–252.
23. Noikajana, S.; Suwannasart, T. Web service test case generation based on decision table. In *Proceedings of International Conference on Quality Software*; IEEE CS: Oxford, UK, 2009; pp. 321–326.
24. Bai, X.; Dong, W.; Tsai, W.T.; Chen, Y. WSDL-based automatic test case generation for web services testing. In *Proceedings of the IEEE International Conference on Service-Oriented System Engineering*; IEEE CS: Beijing, China, 2005; pp. 207–212.
25. Dong, W.L.; Yu, H. Web service testing method based on fault-coverage. In *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference Workshops*; IEEE CS: Hong Kong, China, 2006; pp. 43–50.
26. Grefen, P.; Aberer, K.; Hoffner, Y.; Ludwig, H. CrossFlow: Cross-organizational workflow management in dynamic virtual enterprises. *Int. J. Comput. Syst. Sci. Eng.* **2000**, *15*, 277–290.
27. Alves, A.; Arkin, A.; Askary, S.; Barreto, C.; Bloch, B.; Curbera, F.; Ford, M.; Goland, Y.; Guizar, A.; Kartha, N.; *et al.* Web services business process execution language version 2.0. Available online: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf (accessed on 6 February 2012).

28. Skogan, D.; Gronmo, R.; Solheim, I. Web service composition in UML. In *Proceedings of the IEEE International Enterprise Distributed Object Computing Conference*; IEEE CS: Monterey, CA, USA, 2004; pp. 47–57.
29. Kramler, G.; Kapsammer, E.; Kappel, G.; Retschitzegger, W. Towards using UML2 for modeling web service collaboration protocols. In *Interoperability of Enterprise Software and Applications*; Springer: London, UK, 2005; pp. 227–238.
30. Rumbaugh, J.; Jacobson, I.; Booch, G. *The Unified Modeling Language Reference Manual*; Addison-Wesley Professional: Indianapolis, IN, USA, 2004.
31. Jürjens, J. UMLsec: Extending UML for secure systems development. In *Proceedings of the 5th International Conference on The Unified Modeling Language*; Springer Verlag: Dresden, Germany, 2002; pp. 412–425.
32. Lodderstedt, T.; Basin, D.; Doser, J. SecureUML: A UML-based modeling language for model-driven security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*; Springer Verlag: Dresden, Germany, 2002; pp. 426–441.
33. Automated Validation of Internet Security Protocols and Applications (AVISPA). Available online: http://www.avispa-project.org/ (accessed on 6 February 2012).
34. Chevalier, Y.; Compagna, L.; Cuellar, J.; Drieslma, P.H.; Mantovani, J.; Mdersheim, S.; Vigneron, L. A high level protocol specification language for industrial security-sensitive protocols. In *Proceedings of Workshop on Specification and Automated Processing of Security Requirements*; Austrian Computer Society: Linz, Austria, 2004; pp. 193–205.
35. Automated Validation of Trust and Security of Service-oriented Architectures (AVANTSSAR). Available online: http://www.avantssar.eu/ (accessed on 6 February 2012).
36. Armando, A.; Compagna, L. SATMC: A SAT-based Model Checker for Security Protocols. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence*; Springer: London, UK, 2004; Volume 3229, pp. 730–733.
37. Compagna, L. SAT-based model-checking of security protocols. PhD thesis, Università degli Studi di Genova, Genova, Itay; the University of Edinburgh, Edinburgh, UK, September 2005.
38. Armando, A.; Carbone, R.; Compagna, L.; Cuellar, J.; Tobarra, L. Formal analysis of SAML 2.0 web browser single sign-on: Breaking the SAML-based single sign-on for Google apps. In *Proceedings of the 6th ACM workshop on Formal methods in security engineering*; ACM: Alexandria, VA, USA, 2008; pp. 1–10.
39. Secure Provision and Consumption in the Internet of Services (SPaCIoS). Available online: http://www.spacios.eu/ (accessed on 6 February 2012).
40. Ensuring Trustworthiness and Security in Service Composition (ANIKETOS). Available online: http://aniketos.eu/ (accessed on 6 February 2012).
41. Advanced Security Service cERTificate for SOA (ASSERT4SOA). Available online: http://www.assert4soa.eu/ (accessed on 6 February 2012).
42. Gürgens, S.; Ochsenschläger, P.; Rudolph, C. Role based specification and security analysis of cryptographic protocols using asynchronous product automata. In *Proceedings of IEEE International Workshop on Trust and Privacy in Digital Business*; IEEE CS: Aix-en-Provence, France, 2002; pp. 473–482.

43. Gürgens, S.; Rudolph, C. Security Analysis of (Un-)Fair Non-repudiation Protocols. *Lect. Notes Comput. Sci.* **2003**, *2629/2003*, 229–232.
44. Gürgens, S.; Rudolph, C.; Scheuermann, D.; Atts, M.; Plaga, R. Security Evaluation of Scenarios based on the TCG's TPM Specification. *Lect. Notes Comput. Sci.* **2007**, *4734/2007*, 438–453.
45. Fraunhofer Institute for Secure Information Technology SIT, D. Simple Homomorphism Verification Tool—Manual. Available online: http://publica.fraunhofer.de/starweb/servlet.starweb?path=pub0.web&search=N-47349 (accessed on 6 February 2012).
46. Tsai, W.; Paul, R.; Cao, Z.; Yu, L.; Saimi, A.; Xiao, B. Verification of Web services using an enhanced UDDI server. In *8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003)*; IEEE CS: Guadalajara, Mexico, 2003; pp. 131–138.
47. Ran, S. A model for web services discovery with QoS. In *ACM SIGecom Exch.* **2003**, *4*, 1–10.
48. Serhani, M.; Dssouli, R.; Hafid, A.; Sahraoui, H. A QoS broker based architecture for efficient Web services selection. In *Proceedings of the IEEE International Conference on Web Services*; IEEE CS: Orlando, FL, USA, 2005; pp. 113–120.