

# Research Report

## WS-Attestation: Enabling Trusted Computing on Web Services

Sachiko Yoshihama, Tim Ebringer, Megumi Nakamura, Seiji Munetoh, Takuya Mishina, Hiroshi Maruyama

IBM Research, Tokyo Research Laboratory  
IBM Japan, Ltd.  
1623-14 Shimotsuruma, Yamato  
Kanagawa 242-8502, Japan



**Research Division**  
**Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich**

### **Limited Distribution Notice**

This report has been submitted for publication outside of IBM and will be probably copyrighted if accepted. It has been issued as a Research Report for early dissemination of its contents. In view of the expected transfer of copyright to an outside publisher, its distribution outside IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or copies of the article legally obtained (for example, by payment of royalties).

# WS-Attestation: Enabling Trusted Computing on Web Services

Sachiko Yoshihama<sup>†</sup>, Tim Ebringer<sup>‡</sup>, Megumi Nakamura<sup>†</sup>,  
Seiji Munetoh<sup>†</sup>, Takuya Mishina<sup>†</sup>, Hiroshi Maruyama<sup>†</sup>

<sup>†</sup>IBM Tokyo Research Laboratory, 1623-14 Shimotsuruma, Yamato-shi, Kanagawa, Japan

<sup>†</sup>{sachikoy, nakamegu, munetoh, tmishina, maruyama}@jp.ibm.com

<sup>‡</sup>CA Labs, 658 Church St., Richmond, Victoria, Australia

<sup>‡</sup>tim.ebringer@ca.com

## ABSTRACT

This paper proposes WS-Attestation, an attestation architecture based upon a Web Services framework. The increasing prevalence of security breaches caused by malicious software shows that the conventional identity based trust model is insufficient as a protection mechanism. It is unfortunately common for a computing platform in the care of a trustworthy owner to behave maliciously<sup>1</sup>.

Specifications created by the Trusted Computing Group (TCG) [1][2] introduced the concept of *platform integrity attestation*, by which a computing platform can prove its current configuration state to a remote verifier in a reliable manner. WS-Attestation allows Web Services providers and consumers to leverage this technology in order to make better informed business decisions based on the security of the other party.

Current TCG specifications define only a primitive attestation mechanism that has several shortcomings for use in real-world scenarios: attestation information is coarse grained; dynamic system states are not captured; integrity metrics are difficult to validate; platform state as of an attestation is not well bound to the platform state as of interaction, and; platform configuration information is not protected from attackers. We aim to provide a software oriented, dynamic and fine-grained attestation mechanism which leverages TCG and WS-Security technologies to increase trust and confidence in integrity reporting. In addition, the architecture allows binding of attestation with application context, privacy protection and infrastructural support for attestation validation.

## KEY WORDS:

*Trusted Computing, TCG, Attestation, WS-Security, WS-Trust*

## 1. Introduction

The current and planned Web Services Specifications describe a set of functionality for providing distributed services in a heterogeneous computing environment. A service describes its functional interface in WSDL (Web Services Description Language) [7] and advertises itself through a centralized UDDI registry [8] or ad-hoc metadata exchange [9]. The messages sent

---

<sup>1</sup> Zombie computers used to send spam being a common example.

between services are in SOAP envelopes [10] and transported over various protocols, the most common of which is HTTP.

In order to secure interactions between Web services, various Web Services Specifications are defined or being defined as proposed in the Web Services Security Roadmap [11]. These specifications are defined as building blocks allowing various specifications integrated together to enable security, reliable messages, policy and transactions [12].

At the simplest level the messages can be protected against malicious parties who would eavesdrop on sensitive data or alter messages. The WS-Security specification [13] defines mechanisms for protecting message integrity and confidentiality. WS-Trust [15] defines a generic framework for exchanging security tokens between endpoints to establish trust. WS Secure Conversation Language [14] utilizes WS-Trust and enables secure interaction context that will last for a series of message exchanges. The WS Policy Framework [16] and the WS Security Policy Language [17] defines syntax and vocabularies for exchanging security policies between endpoints. WS Federation Language [18] allows federating identities between multiple parties.

In the WS-Security framework, a trust relationship is established based on the identity of each entity who is participating in a transaction. For example, the sender of a message is identified by a digital signature, which is authenticated by the receiver using an X.509 certificate. The receiver checks the trustworthiness of the certificate and a trust decision is made, depending on the authenticity of the signature and the trust relationship to the Certification Authority (CA) who issued the certificate.

In the business world, an entity utilizes finer grained information to decide whether another entity is trustworthy enough to make a deal. For example, a customer may be interested in whether a service provider has ability to perform a certain service, with certain quality and quantity, by a certain deadline. A customer may also be interested in whether the service provider is honest, abides by the law, respects the privacy policy, has a good reputation, etc. On the other hand, a service provider is generally interested in the customer's financial credibility and his/her eligibility for receiving the service (e.g., does the customer have a license for buying controlled materials).

In addition, when the transaction takes place on the Internet, it is also important to be able to verify that a computer platform, acting on behalf of a service provider or a requester, does not betray the service requester or provider's will. For example, if a user trusts an on-line shopping service and submits his credit card number – even if the service provider company is actually honest and trustworthy – the server platform might be infected with a Trojan horse that surreptitiously sends the credit card number to a malicious remote attacker. In another case, the server software might have a vulnerability that would be attacked by an attacker, allowing the attacker to obtain the super user privilege and steal customer information and credit card numbers. Therefore, it is important to make sure that the service is running on a trustworthy platform; i.e., it is running on the hardware that it claims to be, and that the OS and software are not infected by malicious software and has no known vulnerabilities.

Remote attestation is one of the key functionalities of Trusted Computing which allows a remote challenger to verify not only the identity of the other party but also the integrity of the platform that represents the party. The platform integrity information allows the challenger to verify the configuration and the state of the system (e.g., what OS is running, which security patches are applied, what security policies are being observed or whether it is infected with viruses). It allows intelligent decisions to be made as to whether a service which runs on a trusted platform is fit to use. Conversely, the service provider may choose whether to accept a service request from a requester based on the trustworthiness of the requester. The trusted computing

allows establishing a trust relationship among potentially distrusted distributed parties, thus enabling new types of secure interaction.

There are several issues that need to be considered when enabling Trusted Computing on Web Services. First, it is obviously dangerous for a platform to advertise its precise configuration, because such information is very useful for an attacker in choosing the most effective attack technique. Therefore it is important that access to the precise configuration information is only seen by authorized parties. For distrusted parties, the platform needs to prove only its security properties without revealing configuration details. Second, exchanging and validating a platform attestation can be a relatively heavy process and it is important to adjust balance between fine granularity and effectiveness. Third, the architecture has to support validation of complicated computer platform that consists of various components.

This paper proposes WS-Attestation, attestation architecture built on top of the Web Services framework. WS-Attestation provides a software oriented, dynamic and fine-grained attestation mechanism which leverages TCG and WS-Security technologies to increase trust and confidence in integrity reporting. In addition, the architecture allows binding of attestation with application context, as well as infrastructural support for attestation validation.

The following sections are structured as follows: Section 2 discusses an overview of the trusted computing technology. Section 3 discusses design principles. Section 4 discusses WS-Attestation architecture of attestation support for Web Services framework. Section 5 and 6 discusses the WS-Attestation profile and token exchange model respectively. Section 7 describes prototype implementation. Section 8 shows observations on the architecture. Section 9 discusses related work. Section 10 concludes this paper.

## 2. Trusted Computing Technology

The Trusted Computing Group (TCG) [1][2] defines a set of industry standard specifications for hardware and software for enabling trusted computing among wide variety of computing platforms, such as PCs, servers, mobile phones, etc. The center of the TCG technology is a security module called the Trusted Platform Module (TPM) which is usually implemented as a tamper-resistant hardware module. In addition to serving as a cryptographic co-processor and a protected storage for secrets and keys, the TPM is used to *measure* and *report* platform integrity in a manner that cannot be compromised even by either platform owners or the software running on it. In this section, we review mechanisms of platform integrity measurement and platform integrity reporting (i.e., attestation) in more detail.

### 2.1. Platform Integrity Measurement

Platform integrity measurement consists of multiple phases of measuring and storing integrity of hardware and software components that constitute a platform. Integrity measurement can be categorized into two types: 1) TCG trusted bootstrap and 2) other integrity measurement built on top of trusted bootstrap.

TCG defines the trusted bootstrap process [1][2], that comprises an iterative process of “measurement” (cryptographic hashing), loading, and execution of software components. When the system is powered-on, the immutable initial bootstrap code (such as BIOS boot block) measures next component and stores the measurement in the TPM before transferring control to the next component. In subsequent steps, each software component recursively measures next

component and records the measurements in the TPM, until the operating system is loaded. The BIOS boot block and TPM are called Core Root of Trust for Measurement (CRTM), because they need to be trusted in order to trust measurement in trusted bootstrap.

Each measurement is taken as a SHA-1 (Secure Hash Algorithm 1) value of the binary image of a component, and stored into Platform Configuration Registers (PCRs). PCRs are special purpose registers within the TPM which record integrity measurements, and are protected from an arbitrary modification<sup>2</sup>).

The specification requires 16 PCRs at minimum on a TPM, but each PCR can store fingerprint of multiple components using a hash-chain mechanism. That is, a PCR supports only the *extend* operation to update its value; i.e., When recording a measurement value  $v$  into a PCR, the value  $v$  is *extended* into the PCR, which results a SHA-1 hash over concatenation of the current PCR value and the value  $v$ .

$$PCR_i^n = \text{SHA-1}(PCR_{i-1}^n \parallel v)$$

where the initial PCR value after power-on is  $PCR_0^n = 0$ ,  $n$  denotes the index of the PCR register ( $0 \leq n < 16$ ), and  $\parallel$  denotes a concatenation.

After the OS is loaded and initialized in a trusted bootstrap process, the PCRs will contain some predictable values, provided that the platform runs a known set of components that are intact. If any bits of measured components are modified from the original, the PCR value will be different, and such modification can be detected. The PCR values form a fingerprint of the exact software stack on a particular platform. Note that trusted bootstrap is different from secure bootstrap, in the sense that the system continues the bootstrap process even if there has been unauthorized modification of the components that are loaded, though this modification is recorded. In contrast, secure bootstrap terminates the bootstrap process when unauthorized code is detected.

Trusted bootstrap provides a basis for platform integrity measurement, but it does not prove all aspects of trustworthiness of a system. Various modules and application software which are loaded after the bootstrap process contribute to the trustworthiness of a platform. Similarly, even identical OS kernels can have different level of security if they have different configuration settings<sup>3</sup>, thus such configuration information needs to be measured. Measurement by TPM can be further extended into modules, applications and configurations as follows:

**Run-time measurement at OS.** While the system is running, various behavior, such as module loading or application execution, are monitored and measured by the operating system and recorded into PCRs. Integrity Measurement Architecture [20] realizes such measurement on the Linux kernel, and enhances the role of the TPM not only to measure static state of a system but also the dynamic state.

**Run-time measurement at Middleware.** Various forms of middleware constitute today's computing systems. However, it is not practical to extend OS to measure integrity of data that are used by middleware, because it requires rebuilding the OS each time to support new middleware or data. It is preferable that each middleware layer measures data that is loaded or used by itself. An example of the measurement at middleware is a Java™ Virtual Machine (JVM) that measures integrity of Java class files when each class is loaded.

---

<sup>2</sup> TPM Specification v1.2 supports a new operation to reset TPM to 0, in order to allow a virtualized operating system to leverage TCG without a hard reset.

<sup>3</sup> For example, the Security Enhanced Linux kernel supports an option "noenforce" which disables its mandatory access controls.

**Measurement by Agents.** SHA-1 has its limitations when used as an integrity metric for policy or configuration files as any semantically meaningless change (such as insertion of a white-space character or re-ordering configuration properties in a text-based configuration file) will change the PCR values, resulting in unnecessary complexity when validating PCR values. A solution is to allow a software agent to measure policy and configuration files. Such an agent might be a local daemon which reads system configuration files, and composes a structured message, in canonical form, that describes the properties of the configuration (e.g., network settings, minimum password length, etc.). Similar to the middleware level measurement, the chain of trust from the root-of-trust to agents must be maintained.

Care needs to be taken, though, that a chain of trust needs to be maintained from the CRTM to the component being measured. That is, in the case of measurement at middleware, 1) the integrity of the base code up to OS is measured in the trusted bootstrap sequence, 2) the integrity of a middleware is measured by OS, 3) and finally, the integrity of a file being loaded by the middleware is measured by the middleware. The record of measurements (stored in TPM) must prove that each component is measured by a component that is already measured, and the measurement record is not forgeable. Similarly, integrity of measurement agents need to be assured by OS (or any middleware which integrity is measured by OS) for such agent to be trustworthy.

## **2.2. Platform Integrity Reporting**

Platform Integrity Reporting, or attestation, is the mechanism defined in the TCG specification to report integrity measurements stored in the PCRs. In the attestation process, the TPM signs the PCR values and an external 160-bit challenge (such as a nonce from a challenger to avoid replay attacks) using an RSA private key, the confidentiality of which is protected by the TPM. The attestation is an atomic, protected operation and the attestation signature cannot be forged by malicious software. Therefore, if the TPM is properly designed and implemented to adhere to the TCG specifications, and the platform, including the initial bootstrap code, is properly integrated with the TPM, a remote verifier can have confidence in the integrity measurement reported by TPM.

Note that the platform may also send additional information in an attestation, and prove authenticity of such information using an attestation signature. For example, the list of modules and applications loaded on a platform is useful information to validate the state of a platform and can be included as part of the attestation process. In addition, an attestation signature over the PCRs which record the hash values of loaded modules and applications can prove which components have been actually measured and loaded, thus the challenger can verify authenticity of the modules listed by the PCR value.

## **3. Design Principles**

This section discusses principles that are taken into account in the design of the attestation support in Web Services.

### 3.1. Fine granular, dynamic, verifiable, and efficient attestation

Although TCG provides a hardware-based root of trust, platform integrity measurement and reporting, it conveys little information compared with the complex state of a running system. In WS-Attestation, we aim to complement TCG attestation with fine granularity, dynamicity, and verifiability.

**Fine granularity.** Trusted bootstrap, as defined in TCG, is designed to measure binary images of executables and components (e.g., BIOS configurations) during the bootstrap sequence. However, today's computing systems are complicated and include properties that cannot be meaningfully measured from their binary image. For example, behavior of Linux systems can significantly differ because of parameters specified in configuration files, even if they run on an identical OS kernel and the executable image is the same. It is not practical to measure configuration files with SHA1 hash values; as most of the Linux configuration files are text based, the system administrator can easily break the integrity of a configuration files by adding a white-space or a blank line, even though the semantics of the configuration file is unchanged. Therefore it is desirable that attestation can provide not only binary measurements but also semantic information, e.g., platform configuration retrieved by a software-based attestation agent.

**Dynamicity.** Trusted bootstrap measures integrity of executable components up to the operating system. However, various executables and data loaded on the operating system and on the application layer affect behavior of a running system [20]. It is desirable that the WS-Attestation supports rich semantic attestation information whilst leveraging the root-of-trust mechanism defined in the TCG specification.

**Verifiability.** TPM stores a measurement of components in PCRs in the form of composite hash values. Each composite hash value represents a list of components that are measured and 'extended' into a PCR. Since the PCR extension mechanism allows one PCR to record a list of measurements, each PCR is used to measure many components; e.g., TCG defines minimum 16 PCRs for PC platforms, and 8 of them are reserved for measuring BIOS, while the other 8 are used for measuring the OS and the application layer. As the number of components measured by a PCR becomes bigger, and as the number of possible revisions of each component becomes bigger, the number of permutations that constitute a PCR value becomes factorial; It becomes very difficult for a verifier to validate the platform integrity from a PCR value.

**Efficiency.** As the information conveyed and validated in the attestation process becomes more detailed, the attestation process can become overly expensive. On the other hand, it is not possible to simply separate attestation from the application context, because an entity sending an application message may not be in the same state as what was attested, thus may not be trusted anymore. It is desirable to increase efficiency while maintaining a cryptographic binding between attestation and application context.

### 3.2. Attestation Supporting Infrastructure

As a large number of vulnerabilities are found every day [3], software vendors release security patches frequently. A typical security patch consists of multiple files that replace vulnerable components on the system. Each patch may fix one or more vulnerabilities. Thus it becomes increasingly difficult to make educated decisions as to whether vulnerability is present in a particular file. A well organized infrastructural support is therefore essential to enable validation measurement of each component on the system.

Finally, each entity requesting attestation may not be capable of validating attestation information. We assume presence of trusted third party validation services that validate attestation on behalf of requesters. We aim at defining communication models between the attestation requester, responder, and the validation service.

### 3.3. Privacy Protection

There are two types of privacy need to be considered in attestation: identity and integrity of the platform being attested.

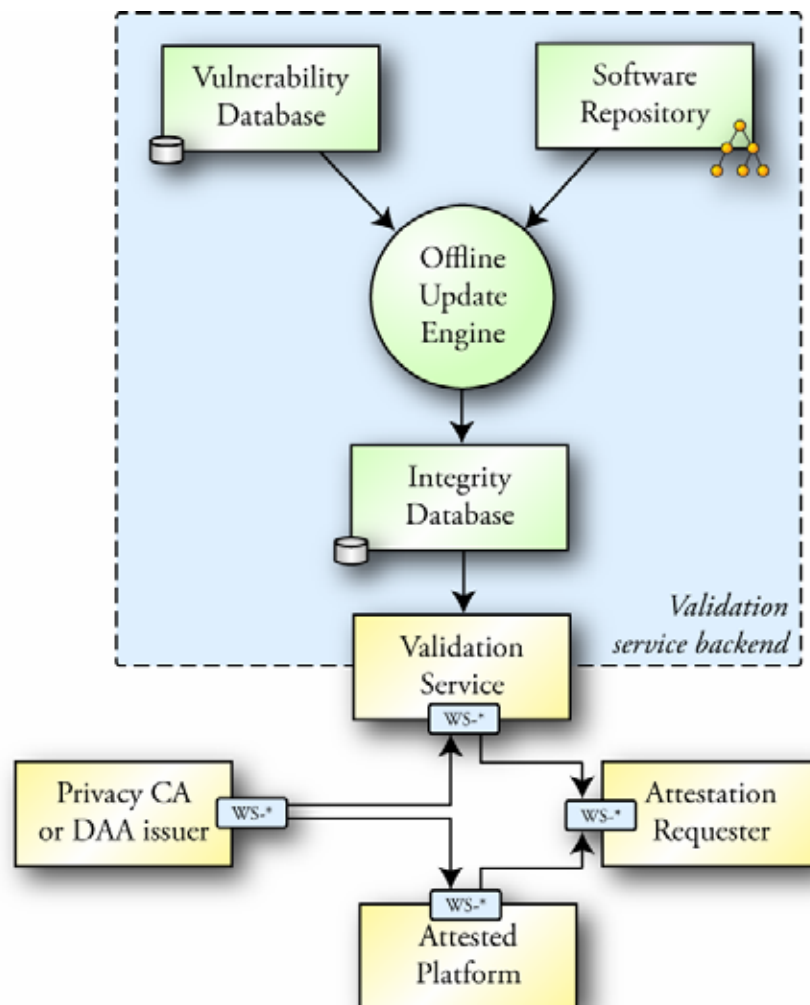
**Identity Privacy.** It is one of the key objectives of TCG-defined attestation to protect the privacy of the platform identity while establishing trust. TCG defines two mechanisms for identity privacy: the Privacy-CA and Direct Anonymous Attestation (DAA). Since current TCG specifications already address identity privacy issues, we do not focus on the identity privacy in this paper.

**Integrity Privacy.** The most unique aspect of attestation is that it proves not only the identity of the platform but also the integrity and state of the platform. Although it is useful information for a legitimate verifier to judge trustworthiness of a platform, it can also become a source of vulnerability if distrusted parties are allowed to perform an attestation. By investigating the OS and application versions, an attacker can quickly deduce the most effective attack technique. Therefore, it is important, especially in cross-organizational transactions, that a platform can prove its trustworthiness to anonymous challengers without disclosing its configuration details. This is addressed in Section 3.7.

## 4. WS-Attestation Architecture

In order for WS-Attestation to be widely adopted and interoperable, it is important that WS-Attestation matches the model and framework of the existing WS-Security standards. Therefore, rather than invent a new protocol for attestation, we leverage existing Web Services standards and define a profile for supporting attestation on top of these standards.

Figure 1 shows architecture of attestation support on Web Services. The attestation requester is an entity who initiates attestation request. The validation service is a trusted third party authority that validates (or sometimes performs) attestation on behalf of the requester. The validation service refers to the integrity database for validating integrity of each component measurement. The Privacy CA or the DAA issuer is responsible for certification of Attestation Identity Keys (AIKs) generated on attested platforms.



**Figure 1 WS-Attestation Architecture and Data Flow**

**Attested Platform (AP).** The platform being attested implements various forms of integrity measurements and is capable of responding to an attestation request. It is also assumed that the attested platform implements appropriate security mechanisms and policies that is to be required by the attestation requester, and presence of such implementation can be measured and reported in the attestation process. Typically, an attested platform is an entity which provides services through a Web service interface.

**Attestation Requester (AR).** A party who is interested in platform integrity of the attested platform. An attestation requester may not be capable of validating integrity measurement by itself, and relies on a validation service to make judgments. Typically, an attested platform is an entity who is attempting to use a service provided by AP.

**Validation Service (VS).** A trusted third party service which issue attestation credentials each of which asserts a set of security properties that represents the state of a platform. Typically, VS verifies integrity measurements of an AP, derives a set of properties, and issues an attestation credential that asserts the properties. The VS needs to be trusted by both AP and the AR, in a

sense that 1) AP has to trust VS that VS does not disclose configuration information of AP, and 2) AR has to trust VS that VS correctly performs validation based on the criteria that is understood by both AR and VS.

**The Integrity Database** is a repository of black-list and white-list of metrics of known components, and used by VS in validating the platform integrity measurements. The Integrity Database leverages external information sources, such as software repository and vulnerability database, to consolidate various information sources into a representation useful for VS.

**Identity Credential Issuer (Privacy CA or DAA Issuer).** Identity credential issuer certifies identity of each AP by issuing a credential (e.g., an X.509 certificate) on an attestation identity key (AIK) of AP. In TCG specifications, it is important to prove that AP is a genuine trusted platform, which has a genuine TPM and implements CRTM, because trust on integrity measurement builds on the trust on the implementation of the platform.

## 5. WS-Security Attestation Profile

This section defines detailed WS-Attestation protocol as a profile on existing WS-Security standards.

### 5.1. Attestation Signature

WS-Security defines a flexible framework for protecting message integrity and confidentiality using various cryptographic algorithms. We define a new signature algorithm profile that represents TCG attestation signature.

The TCG attestation signature is an RSA signature value generated by an AIK over the concatenation of the target data and PCR values. The signature operation is an atomic operation performed by the TPM. The values in the PCRs and the use of the AIK are also protected by the TPM. Therefore, a TCG attestation signature proves that the signed PCR values are not compromised, and represents the state of the attested platform at the time of signing. The signature value is considered a special form of the RSA signature. In order to handle the attestation signature in WS-Security as an XML digital signature, we define a new signature method that is identified by the URI and specified in the Algorithm attribute of the SignatureMethod element of the XML digital signature.

In order to verify an attestation signature, the verifier needs to be informed of the TPM\_QUOTE\_INFO structure that is being signed. The TPM\_QUOTE\_INFO includes a 160-bit challenge and a composite hash of selected PCR values. In the proposed signature method, this structure is concatenated to the signature value that is included in the SignatureValue element as

$$\text{TPM\_QUOTE\_INFO} \parallel [\text{TPM\_QUOTE\_INFO}]_{\text{AIK}}$$

(where  $\parallel$  denotes concatenation and  $[x]_K$  denotes a signature over  $x$  with the key  $K$ ).

In the XML Digital Signature, which defines extensible XML schema, it is also possible to add the TPM\_QUOTE\_INFO structure as a separate XML element. However, adding this structure to the signed value has two advantages. First, the same signature method can be used in protocols other than WS-Security where messages have no or little extensibility to include an additional element of information. Secondly, a provider-model crypto API such as Java Cryptographic Extension (JCE) supports different crypto algorithms under the same generic API.

Such generic API cannot be extended to add an extra parameter without losing advantage of plugability. By including the TPM\_QUOTE\_INFO in the signature value, the crypto provider can receive the necessary information for verification of an attestation signature through a generic API.

## 5.2. Platform Measurement Description

The Platform Measurement Description (PMD) is structured data that describes the state of the platform in a fine-grained and semantic manner. A PMD includes the log of measurements which are recorded during the trusted bootstrap and run-time, and describes the components that have been measured by PCRs. Such a log allows the verifier to validate the integrity of each component running on the system. The verifier can also verify that the hash of all components in the log matches the PCR values in the attestation signature. Since the PCR values in the attestation signature are not forged, provided that the TPM is genuine and not in direct contact with an attacker who performs hardware-level attacks, we can use these values to verify integrity of the PMD in such a way that malicious software cannot cause false positive validation.

In addition, a PMD may include non-TPM measurements such as semantic configuration parameters and properties of the platform which are tested or read by an *attestation agent* running on the platform. The integrity of such agent can be measured and reported in TPM-based measurement, providing a chain from the root-of-trust (CRTM) to the software based measurements.

## 5.3. Attestation Credentials

As PMDs become richer, validating the PMD at each transaction may become a bottleneck. To make attestation more efficient, we propose the notion of attestation credentials. An attestation credential has properties that are asserted by an authority, and may have expiration period. A typical attestation credential is issued by a trusted authority that asserts some properties (e.g., `hasKnownVulnerability='false'` or the level of trustworthiness such as `trustLevel={1,...N}`) about an attested platform. An attestation credential may bind a particular set of PCR values to the properties. Upon a challenge by an attestation requester, the attested platform may present the attestation credential along with the attestation signature signed over the challenge (See Section 6 for more details). By verifying the challenge, the PCR values and attestation credentials, the attestation requester can verify, without knowing the details of measurement description, that the attested platform's current state is represented by the PCR values in the attestation signature, and the PCR values represent the properties that are asserted in the attestation credential. The attestation credential also protects integrity privacy of the attested platform from potentially distrusted attestation requesters, especially by utilizing PCR obfuscation technique described in Section 3.7.

An attestation credential can be represented in various forms; e.g., an X.509 attribute certificate and a SAML Assertion are well-standardized formats for this purpose.

## 6. WS-Attestation Token Exchange Model

In WS-Attestation, we propose exchanging attestation information in the form of security tokens, based on WS-Trust, a standard token exchange protocol [15]. By transforming attestation information into security tokens and exchanging them, we can communicate the state of the platform integrity efficiently, rather than performing the PMD-based attestation process for each message.

This section describes how we map attestation token exchange into WS-Trust, and then review logical token exchange patterns with observation on pros and cons of each pattern. Next, we propose a method to protect platform integrity privacy from misuse, and show an example message exchange scenario that integrates all proposed technology elements. Finally, we discuss advantages and disadvantages of several different mechanisms that bind attestation and messaging context.

### 6.1. Mapping to WS-Trust

Rather than defining a proprietary protocol for attestation, we leverage WS-Trust (WS-Trust, 2005). In a WS-Trust message, a requester may request a particular type of a security token, with an optional challenge. Upon a successful response, the responder returns the requested security token. The challenge in the request should be returned back to the requester with a responder's signature over it, thus proving that the response is fresh and is not replayed from past records.

WS-Trust defines generic framework for exchanging security tokens on Web services. The basic message structures in WS-Trust is RequestSecurityToken and RequestSecurityTokenResponse, that represents token request and response respectively. The basic structure of these messages are as follows:

RequestSecurityToken:	TokenType [, Supporting] [, SignChallenge]
RequestSecurityTokenResponse:	TokenType, RequestedSecurityToken, [SignChallengeResponse]

(Note that WS-Trust supports more flexible message structure, but we discuss only minimal elements that are relevant to our proposal, for the sake of simplicity)

Here, the *TokenType* in RequestSecurityToken message specifies URI that represents particular token type, which the requester demands. An optional *Supporting* element provides a security token and provides additional claim information. An optional *SignChallenge* element contains a value, for which requester demands the responder to return a signature.

The *TokenType* in the RequestSecurityTokenResponse message is the type of actual returning token. The RequestedSecurityToken element contains the returning token, and *SignChallengeResponse* contains the responder's signature over the Challenge.

In WS-Attestation we have three kinds of token request/response interactions that are illustrated below in an informal syntax:

- 1) Requesting full attestation information in PMD. This implies that an entity (i.e., usually an attestation requester) may ask for a PMD from an attested platform.

RequestSecurityToken:	TokenType=PMD_TokenType, SignChallenge=nonce
-----------------------	---

RequestSecurityTokenResponse: TokenType=PMD\_TokenType,  
RequestedSecurityToken=PMD,  
SignChallengeResponse=[nonce]Sig

- 2) Requesting issuance of an attestation credential by presenting a PMD. This implies that an entity (i.e., an attested platform or attestation requester) may request a validation service to evaluate a PMD and to issue an attestation credential which represents the result of validation.

RequestSecurityToken: TokenType=AC\_TokenType,  
Supporting=PMD

RequestSecurityTokenResponse: TokenType=AC\_TokenType,  
RequestedSecurityToken=AttestationCredential

- 3) Requesting an attestation credential. This implies that an attestation requester may request for an attestation credential to the attested platform.

RequestSecurityToken: TokenType=AC\_TokenType,  
SignChallenge=nonce

RequestSecurityTokenResponse: TokenType=AC\_TokenType,  
RequestedSecurityToken=AttestationCredential,  
SignChallengeResponse=[nonce]Sig

Note that the responder in each interaction may be a trusted party itself, or a validation service who issues attestation credential for the attested platform.

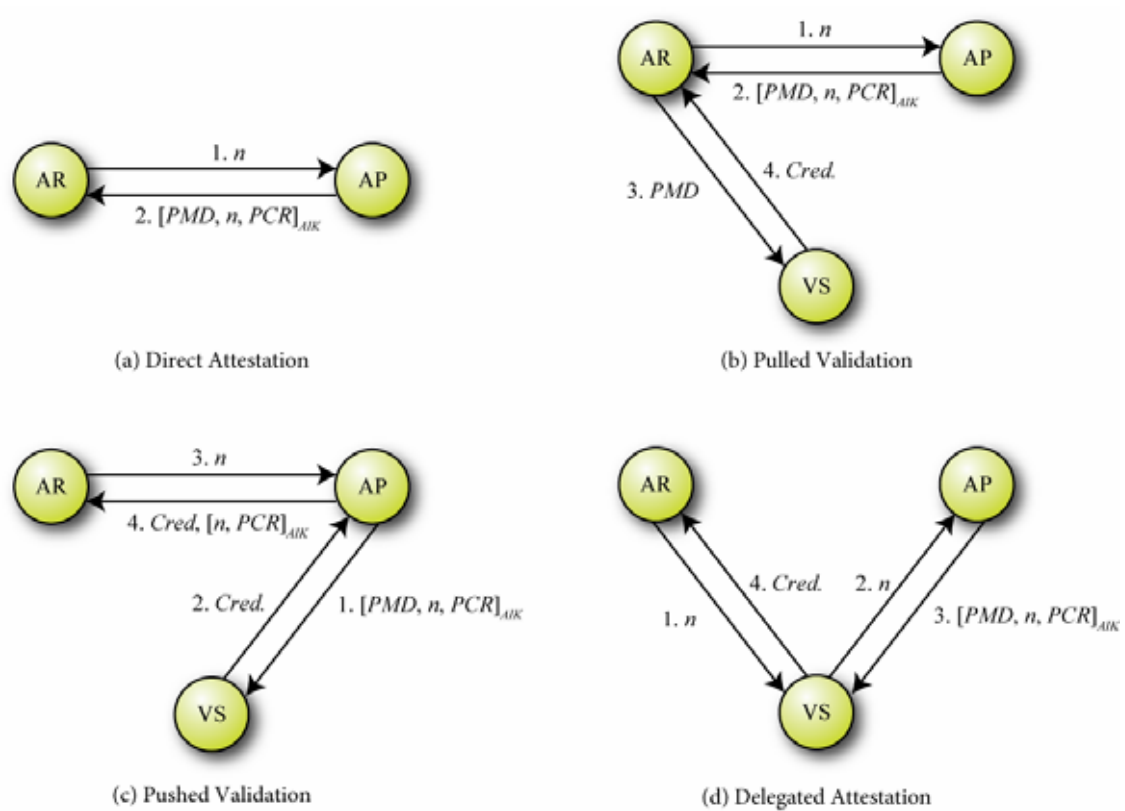
In the following subsection, we illustrate four communication patterns using WS-Trust messages above.

## 6.2. Attestation Token Exchange Patterns

An attestation requester (AR), an attested platform (AP), and a verification service (VS) play central roles in an attestation, especially in verification of integrity of the attested platform. This section discusses four attestation models each of which is built on a different trust model, and has advantages and disadvantages.

**Direct Attestation.** Figure 2a shows the Direct Attestation Model in which an attestation requester challenges the attested platform, which then returns the measurements back to the requester. The attestation requester validates information by itself, which has the advantage of not requiring that any other party need be trusted. This model has two notable disadvantages. 1) The attestation requester has to be capable of validating the attestation response; 2) the attested platform has to disclose all of its integrity information to the requester, which violates its integrity privacy to potentially distrusted attestation requesters.

**Attestation with Pulled Validation.** The second model (Figure 2b) is similar to the Direct Attestation, except that the attestation requester consults the validation service to validate the PMD, and does not have to be capable of validating attestation. Integrity privacy of the attested platform is not protected in this model. An additional disadvantage is that this model may suffer from the performance bottleneck of the validation service, because for every attestation the validation service needs to be contacted.



**Figure 2 Attestation Model**

**Attestation with Pushed Validation.** In the attestation with pushed validation model (Figure 2c), the attested platform pushes the attestation to the validation service, to request an attestation credential. Upon a challenge from the attestation requester, the attested platform sends the attestation credential along with the attestation signature over the challenge, thus allowing the attestation requester to verify that the attested platform has the properties asserted in the credential. The advantages of this model are that 1) the attested platform does not have to disclose integrity information to the attestation requester; 2) the attestation requester does not have to be capable of validating attestation, 3) the performance bottleneck at the validation service is of less concern, because once an attestation credential is issued by the validation service, the attested platform can re-use the credential for subsequent transactions. Finally, the attested platform can choose which validation service to disclose its integrity information to, thus helping maintain the privacy of platform.

**Delegated Attestation.** In the delegated attestation model (Figure 2d), the attestation requester requests a validation service to perform attestation on behalf of the requester, and then sends only the validation result in the form of a credential. The advantages of this model are that 1) the integrity privacy of the attested platform is protected; 2) the attestation requester does not have to be capable of validating attestation.

### 6.3. Privacy Protection: PCR Obfuscation

As we discussed in Section 3.3, attestation must address two types of privacy issue: identity and integrity. This paper focuses on privacy of integrity information. One problem with attestation is that it provides detailed configuration information useful to an attacker, since they may use this to choose which attack tools will be effective against the platform, or when a platform has changed its configuration. The solution is to extend each PCR register with a random value at random times, yet at the same time recording these random values in the log. The resulting PCR value is unpredictable; provides no information about configuration details to the attacker. However, using the log of all PCR measurements, a legitimate verifier can still verify integrity of all other components.

In attestation models with a third-party validation service, the validation service may issue a credential to the measurement log including random extensions, and the credential asserts that some properties are true only when the attested platform has a particular set of PCR values. The attestation requester who receives the credential and the current PCR values cannot derive detailed configuration information from PCR values, but can verify that the current PCR values prove the properties asserted in the attestation credential.

Random extension of PCRs may be performed any number of times, provided the log of the extensions is maintained. Especially important is that the extension is performed more frequently than release of security patches components that run on the system. If a patch that fixes vulnerability is released, by observing PCRs before and after the patch release, an attacker can infer whether the patch is applied to the platform and will be able to make an educated decision on attack tactics.

### 6.4. Binding to Secure Conversational Context

Several approaches are possible to bind the state of an attested platform to an application context.

First, the attestation requester and the attested platform may establish a secure communication channel before attestation. WS-SecureConversation [14] defines a key exchange protocol to exchange a shared secret, which enables the binding between attestation and subsequent transactions by adding Hashed Message-Authentication Code (HMAC) to the messages. Care needs to be taken to protect the shared secret from being bound to distrusted attested platforms; not only must the attestation requester discard the shared secret when the attestation fails, but the attested platform must also discard the shared secret when its state changes. When the application is terminated, or the system is rebooted, the attested platform must exchange a new shared secret and start the attestation process again; it must be verified before a key exchange that the attested platform and its applications are implemented to relinquish shared secrets at termination. However, if the state of the attested platform changes without terminating the application, e.g., as a result of additional kernel module being loaded, this change is difficult to detect at the application layer. To prevent the use of a shared secret in a context that is not expected, the secret should expire and be renewed in a short window of time. This has the obvious side-effect of reducing performance.

Second, the attested platform can sign each application message with the attestation signature. The PCR composite hash value included in the attestation signature proves the state of the attested platform at the time of signing, and therefore that the properties asserted in the credential are still in effect. The freshness of the attestation signature has to be verifiable; e.g., by having a signature over the timestamp and the application message body. If the attestation

signature is performed on a SOAP response message, the entire application protocol should include a challenge-and-response scheme. Although performing attestation signature on each message requires extra processing power on each party, this mechanism allows verifying the latest state of the attested platform without needing to maintain shared secret keys between peers. An attestation credential should be sent to the attestation requester when it needs a new credential for verifying the attestation signature, but the credential can be re-used until it expires or is revoked. An attestation credential may be valid even for multiple attested platforms as long as they have identical integrity measurements.

## 6.5. Example: Pushed Validation with PCR Obfuscation

WS-\* specifications are defined to be building blocks that can be composed together to meet specific requirements. Likewise, WS-Attestation profile is intended to be a flexible set of building blocks. However, in order to illustrate its usage, we review an example scenario with one particular combination of the profile and a model.

In this scenario, the attested platform (AP) wants to prove its platform integrity to the attestation requester (AR), but does not trust AR in a sense that AR might misuse AP's configuration details. On the other hand, the AR requires verifying platform integrity of AP on each message exchange, to make sure that the state of AP has not been changed into unexpected state as of the time of message exchange. Presumably, attestation and validation are heavy processes and should be performed as few as possible.

In order to meet these requirements, we employ the following steps as shown in Figure 3. (Note that for the sake of simplicity, we use only one PCR in this example while actual attestation may deal with multiple PCRs):

- 0) In the integrity measurement phase, AP uses PCR obfuscation technique to randomize the values in PCR. After trusted bootstrap finishes, the original PCR value is  $PCR_i$ , where  $i$  is the number of components that have been measured by the PCR. An attacker may infer exact platform configuration from  $PCR_i$ . Then AP *extends*  $PCR_i$  with a random value *salt*, and also records the salt value into the measurement log. Resulting  $PCR_{i+1}$  is randomized and no detailed configuration can be inferred from it.
- 1) AP requests VS to issue an attestation credential for the PMD (including the measurement log) of the AP. AP's intention is to present the attestation credential to any party to prove its security properties, without showing detailed configuration information.
- 2) When VS receives a PMD from AP, it first verifies integrity and trustworthiness of each component in the measurement log. The random value extended into PCR will be ignored in this phase, as it is marked as *salt* in PMD. Second, VS verifies that the hash chain derived from the measurement log, including the salt, matches the PCR value  $PCR_{i+1}$  in the attestation signature; i.e., the measurement log is not altered. Third, VS validates any additional properties measured by agents and recorded into PMD. Finally, VS generate an attestation credential which asserts validation result as particular security properties, and those properties are to be associated with particular set of PCR values, and then send the credential back to AP. Note that the PCR value in the attestation credential is obfuscated with salt, thus nobody can infer the precise platform configuration from this value.

- 3) AP sends the attestation credential to AR to presents its security properties and the PCR values that the properties are associated with. When sending application level messages, AP signs each message with the attestation signature with  $PCR_{n+1}$ , to prove the PCR values as of the signature.
- 4) Upon reception of each application level message, AR verifies the signature as if it is an ordinary digital signature over the message (which might also include a challenge from the AR), with the specific *attestation* signature algorithm as defined in this paper. Then AR verifies that  $PCR_{i+1}$  in the signature matches the PCR values in the attestation credentials, thus the security properties asserted in the credential is still effective as of the message generation.

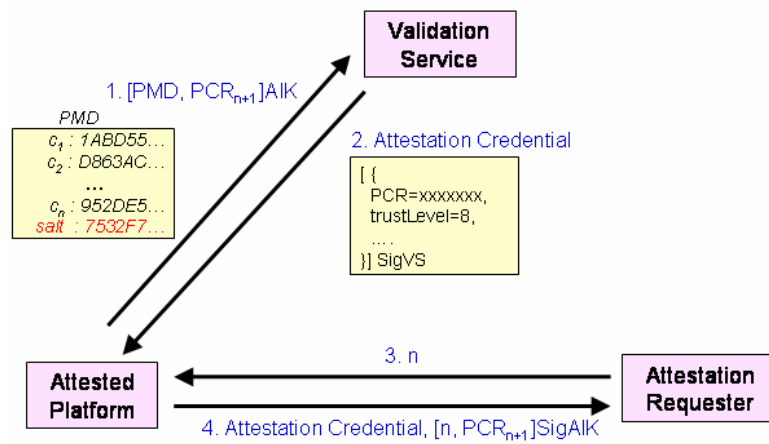
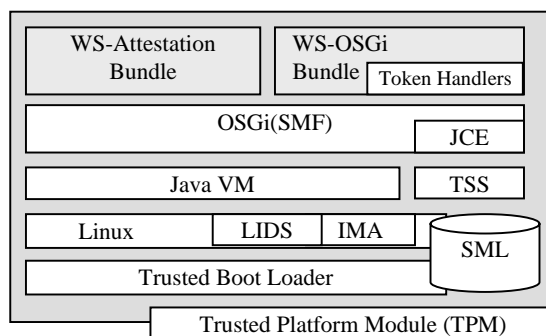


Figure 3 Example: Pushed Validation with PCR Obfuscation

## 7. Prototype

Figure 4 shows the architecture of the attested platform.

The integrity of the Linux OS is measured by the modified boot loader, and loadable modules and executables are measured by IMA[20]. The measurements (SHA1 hash values of files) are stored in PCRs as well as in the kernel-held store measurement log (SML).



**Figure 4 Prototype Attested Platform**

Linux Intrusion Detection System (LIDS)[29] is used to improve the OS level security. LIDS consists of a kernel patch and administration tools which enhances the OS security by enforcing Mandatory Access Control (MAC) policies on operating system resources.

The prototype service is implemented in Java, and runs on the OSGi (Open Service Gateway initiative) [30] platform, which is an open-standard framework for Java based applications and services. We extended IBM Service Management Framework (SMF) [31], one of the OSGi implementations, to measure each bundle JAR file when it is loaded and record the measurement into PCR and the log.

We also extended the WS-OSGi, a light-weight SOAP/WS-Security engine for OSGi platforms, to support the attestation signature and tokens described in Section 3. The attestation signature and verification operation are implemented as a crypto provider of Java Cryptography Extension (JCE) [33] and communicates with TPM via TCG Software Stack (TSS)[32]. The WS-Security engine can switch between an ordinary RSA signatures and the attestation signature simply by specifying the signature algorithm and the key storage as a set of options.

The attestation requester and the validation service are also implemented as services on the OSGi platform and communicate each other using the WS-Trust protocol. PMDs returned by the attested platform consists of stored measurement log in the XML format, while an attestation credentials are implemented as a SAML attribute assertion signed by the validation service.

See Appendix A. for sample SOAP messages.

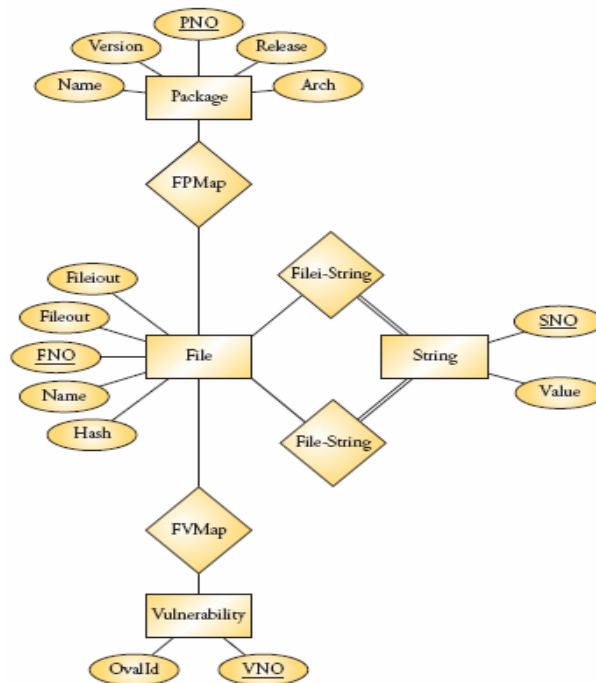
## 7.1. Validation Service and Integrity Database

The validation service implements a Web services interface which receives PMD and returns an attestation credential that contains the validation result of the PMD.

Our validation service prototype provides a database which is used for making informed decisions as to the integrity and quality of each component represented by an integrity metrics. This allows attestation verifiers to query the integrity and vulnerability of each measured component.

Many operating systems support mechanisms to distribute software components and patches in precompiled packages. For instance, RedHat's Package Manager (RPM) is the standard way of distributing and deploying components of RedHat's Linux distribution. When a different version is then distributed, the executable images in the package almost always have a new hash value.

Thus, the exact version of an RPM package can often be deduced from the hash values of its executable files.



**Figure 5 Integrity Database ER Diagram**

A relatively recent endeavor in platform security is the Online Vulnerability and Assessment Language (OVAL)[24], sponsored by MITRE and supported by various operating system vendors, including RedHat[34]. OVAL is a language for expressing the preconditions necessary for a vulnerability to exist. Although the exact semantics differ depending on the operating system platform, the RedHat variant references particular RPM packages.

A hash database of RPM packages was built by simply unpackaging RPMs and generating hash values of all ELF executables. By parsing OVAL vulnerability descriptions and correlating these with RPM package versions, we were then able to deduce which executable hash values would indicate the presence of vulnerabilities.

We found that a verifier with a database like the one could verify the RPM-providence of all the executable images loaded. Furthermore, by cross-referencing with OVAL vulnerabilities, they could determine the presence of vulnerabilities, merely from the hash values.

The integrity database prototype is built on DB2 and queried by the validation service by SQL over JDBC. The integrity database currently supports RPM packages only; data entries are generated from RPM package repository for RedHat Enterprise Linux 3 (REL3) and OVAL repository for this OS, thus capable of validating integrity of REL3 systems.

## 8. Discussion

In this section, we discuss how WS-Attestation contributes in achieving our design objectives.

### 8.1. Performance of Attestation

We refer to an attestation with full PMD a “full attestation” and an attestation with an attestation token a “token-based attestation”. Our proposal assumes that full attestation is required to fine granular validation of precise state of the attested platform, but it is expensive in terms of computational cost. So light-weight token-based attestations can replace full attestation in each application message. Therefore, performance difference between a full attestation and a token-based attestation signifies the performance improvement achieved by this proposal.

Time required for full attestation  $T_{fa} = msg_{comp} + pmd_{comp} + as_{gen} + tr + ps + as_{verif} + pmd_{verif}$ , where  $msg_{comp}$  and  $pmd_{comp}$  are the time required for composition of SOAP message and PMD respectively,  $as_{gen}$  is time required for generating an attestation signature,  $tr$  is time required for transmission,  $ps$  is time required for message parsing,  $as_{verif}$  is time required for verifying attestation signature,  $pmd_{verif}$  is time required for validating PMD.

Time required for token-based attestation  $T_{ta} = msg_{comp} + at_{comp} + as_{gen} + tr + ps + as_{verif} + at_{verif}$ , where  $pmd_{comp}$  is the time required for composition of an attestation token,  $at_{verif}$  and is time required for verifying an attestation token.

Therefore, performance difference between a full attestation and a token-based attestation is  $(pmd_{comp} - at_{comp}) + (pmd_{verif} - at_{verif})$ . The performance difference between  $pmd_{comp}$  and  $at_{comp}$  usually depends on how much overhead it costs to collect information for PMD. Since a PMD consists of measurement log and other information, it requires access to stored measurement log in files and BIOS memory. Caching or pre-collection mechanisms may be used to optimize composition of PMD. On the other hand, composition of an attestation token does not require much processing, since the same attestation token can be reused for every transaction.

Verification of a PMD takes significantly more time than verification of an attestation token, as it require accessing validation database for looking up semantics of measurement hash values. In our first prototype, verification of a PMD with approximately 100 hash values took more than a minute, because of the overhead of the 100 database queries. This was significantly reduced by using stored procedure to lookup all hash values in one query. Still, access to database takes about a second in our environment. Validation of an attestation token is quicker, as it does not require any network access but just parsing the token and checking the properties stored in it. Therefore, we can safely say that the token-based attestation is quicker than the full-attestation.

Although it can provide fine-granularity, token-based attestation does not provide much of performance benefit compared to the binary attestation of TCG, since most performance overhead is caused by the generation of an attestation signature, which is a 2048 bits RSA signature generated by TPM. Additional performance improvement, in a long-running transaction, can be achieved by adopting WS-SecureConversation [14] to bind the attestation token into the secure messaging context using a light-weight symmetric key signature algorithm (such as HMAC).

## 8.2. Vulnerability Detection

The architecture combining TCG trusted boot, IMA allows measuring all native applications running on the platform. In addition, supporting measurement at the OSGi framework allows each JAR file to be measured when being loaded. Therefore, proposed scheme allows fine-grained and dynamic measurement of the platform integrity, which enables detection of vulnerabilities or malicious software in timely manner. If malicious code runs after an attestation credential is obtained, the presence of the malicious code can be detected from the latest PCR value included in the attestation signature. In addition, the platform measurement description (PMD) in our architecture can be easily extended to include any finer-granular information measured by software agents. Examples of non-TPM measurement include the Java security policy, or system properties such as the minimum length and the maximum lifetime of the administrator's password.

However, still some problems exist in the current proposal.

First, it provides weak binding between integrity measurements and an entity involved in a transaction. I.e., when an attestation proves that the platform runs a genuine application and a (potentially) malicious application, it is difficult to assure which application is the originator of the messages. This leads to the "all-or-nothing" policy, in which all applications cannot be trusted when there is any potentially malicious components are present on the same platform. Strong isolation of execution environment for each application, and binding between attestation and the execution environment would be required to prevent such overly strict policy.

Second, the current architecture only deals with the trustworthiness of known components that are either in a black-list or in a white-list. However, in reality, many Web services are running custom-made applications which potentially have unknown vulnerabilities. One of the potential solutions for this problem is to have a distributed trusted computing base (TCB) among services. That is, all services run a common middleware (or one of a few common middleware) as a distributed TCG, which can confine application behavior with a given policy. Through integrity attestation, each service proves not only integrity of the TCB but also which policy is being enforced. Trusted Virtual Domains[27][28] is envisioning such architecture for establishing trust-based distributed coalition.

## 8.3. Attestation Supporting Infrastructure

The current prototype uses OVAL database as the source of vulnerability information. An OVAL document consists of a set of tests for detecting vulnerabilities. OVAL consists of two types of tests; whether particular components exist on the platform, and what are the configuration parameters of the platform. Combination of these two types of tests enables to make intelligent decision on vulnerability presence. We utilize OVAL only as a source of the vulnerability information, thus the information about configuration parameters is not utilized. This introduces false-positives, since some vulnerability may exist only when a particular version of the component runs with particular configuration parameters.

The other problem of OVAL is, that although it is a comprehensive vulnerability database which is based on unique numbering of vulnerability in Common Vulnerabilities and Exposures (CVE), the support of operating systems is limited. OVAL scheme consists of two parts; the core scheme which is generic among all platforms, and platform specific schemas. As of writing of this article, schemas for Cisco IOS, UNIX, HP-UX, Debian Linux, Red Hat Linux, Apple Macintosh, Sun Solaris, and Microsoft Windows are released. However, vulnerability definition

of only three platforms, i.e., Windows, Red Hat Linux and Sun Solaris, are released. Since timely update of maintenance of vulnerability information requires contribution of OS vendors, we still need to wait for emergence of the eco-system for the supporting infrastructure to mature.

## 9. Related Work

Related work includes previous efforts to secure Web services interactions, establish trust relationship between parties measuring, reporting and verifying system integrity.

As discussed earlier, various WS-\* specifications have been defined or being defined [11] for protecting interaction on Web services and communicating trust models. However, current specifications are concerned only with identity based trust model and do not deal with platform integrity based trust. WS-Attestation leverages flexible WS-Trust framework to communicate platform integrity metrics and assertions in an effective manner.

WS-Policy [16] is a generic framework for expressing policies of web services, and WS-SecurityPolicy [17] defines a set of vocabulary for expressing policies on how to protect Web services messages. They are concerned with messaging level security, but new set of vocabulary can be defined in future for expressing requirements on state of the platform which behaves on behalf of the service provider and the requester.

AEGIS system [19] provides secure bootstrapping architecture on PC system that maintains integrity chain from the lowest trustable layer of a system. Secure bootstrap is different from trusted bootstrap in a sense that its objective is not to allow remote verification of the system integrity; in the secure bootstrapping, the system aborts bootstrap process upon integrity check failure.

Sailer et al leverages TCG in Integrity Measurement Architecture (IMA) [20] , to enhance the role of the TPM not only to measure static state of a system but also dynamic state. IMA is implemented as a Linux Security Module to measure each executable, library, or kernel module upon loading and record the SHA1 hash values into TPM and the log. As mentioned earlier, we leverage TCG and IMA to build Linux based attested platforms.

More recent work of Sailer [21] utilizes the integrity measurements and attestation to protect remote access points, to enforce corporate security policies on remote clients in a seamless and scalable manner. Cisco and IBM have announced an enterprise network security solution based on their current products: Cisco's Network Admission Control (NAC) protects the network infrastructure by enforcing security policy compliance on all devices seeking to access network computing resources. The integrated security solution leverages IBM Tivoli Compliance Manager (TSCM) which inspects device configurations, thus denies network access to the devices that are not compliant to the corporate security policies. The compliance checks are based on software agents (e.g., whether anti-virus software is up to date, or the OS is running the latest software patches), but NAC's extensible architecture would allow incorporating further attestation mechanisms in the future.

Terra [26] realizes isolated trusted platforms on top of a virtual machine monitor, and allows attestation by a binary image of each virtual machine, e.g., virtual disks, virtual BIOS, PROM, and VM descriptions.

Recent efforts on mitigating drawbacks of TCG attestation include a proposal [22], which leverages language-based security and virtual machines to enable semantic attestation, e.g., attestation of dynamic, arbitrary, and system properties as well as behavior of the portable code.

Property-based Attestation [23] proposes an attestation model with a trusted third party that translates low-level integrity information into a set of properties.

This paper leverages prior work and define substantial mapping to the WS-Security framework. Our contribution includes 1) extension of integrity measurement architecture [20] into upper layers, especially Java applications, 2) definition of WS-Attestation profile that works on top of existing WS-Security standards with ability to attest fine-grained system configuration while protecting integrity privacy, and 3) a proposal for attestation supporting infrastructure.

## 10. Conclusion and Future Direction

This paper presents our proposal on WS-Security support for attestation. Although attestation is a generic technique to allow remote verification of platform integrity, our proposal is based on TCG, which is the most promising and available technology at time of writing. This paper shows a set of profiles that seamlessly works on existing WS-Security standards. On top of the WS-Trust protocol, attestation information can be exchanged as two forms of tokens; i.e., the platform measurement description that conveys fine granular and semantic information, and the attestation credential that binds low-level integrity measurement into high-level property assertion. Using these tokens, the attested platform can prove its properties to requesters with little performance overhead. We also take privacy protection into account, so that the configuration privacy of the attested platform is protected from potentially distrusted challengers and allows only high-level properties to be reported. The integrity database, which incorporates information on binary measurements and vulnerabilities of deployed package software, provides base for efficient, accurate, and fine granular attestation validation.

## Acknowledgment

The authors wish to thank anonymous peer reviewers and many people of IBM Corporation for comments and insights on an earlier version of this paper. This work is partly supported by Japanese Ministry of Economics, Trade and Industry through contract of a New Generation Information Security Research and Development project.

## References

- [1] Trusted Computing Group, TPM Main Specification <http://www.trustedcomputinggroup.org/>
- [2] TCG Specification Architecture Overview, Revision 1.2 28 April 2004.
- [3] CERT/CC Statistics 1988-2005, [http://www.cert.org/stats/cert\\_stats.html](http://www.cert.org/stats/cert_stats.html)
- [4] Web Services Trust Language (WS-Trust), <http://www-106.ibm.com/developerworks/library/specification/ws-trust/>
- [5] Java Cryptographic Extension (JCE), <http://java.sun.com/products/jce/>
- [6] Assertion and Protocol for the OASIS Security Assertion Markup Language (SAML), <http://www.oasis-open.org/>
- [7] W3C Working Draft, Web Services Description Language (WSDL) 2.0, <http://www.w3.org/TR/2005/WD-wsdl20-primer-20050803/>, August 3, 2005.
- [8] UDDI Spec Technical Committee Draft, UDDI Version 3.02, <http://www.oasis-open.org/>, October 19, 2004.

- [9] IBM, BEA Systems, Microsoft, SAP AG, Computer Associates, Sun Microsystems, webMethods , Web Services Metadata Exchange (WS-MetadataExchange), <http://www-128.ibm.com/developerworks/library/specification/ws-mex/> , September 2004.
- [10] W3C Recommendation, SOAP Version 1.2, <http://www.w3.org/TR/soap/>, June 24, 2003.
- [11] IBM, Microsoft, Security in a Web Services World: A Proposed Architecture and Roadmap, <http://www-128.ibm.com/developerworks/library/specification/ws-secmap/>, April 01, 2002.
- [12] IBM, BEA Systems, Microsoft, Arjuna, Hitachi, IONA, Web Services Transactions specifications, November 1, 2004.
- [13] OASIS Standard 200401, Web Service Security: SOAP Messaging Security 1.0 (WS-Security 2004), March 2004.
- [14] IBM et al., Web Services Secure Conversation Language (WS-SecureConversation), February 2005.
- [15] IBM et al. Web Services Trust Language (WS-Trust), February 2005.
- [16] BM, BEA Systems, Microsoft, SAP AG, Sonic Software, VeriSign, Web Services Policy Framework (WS-Policy), September 2004.
- [17] IBM, Microsoft, RSA Security, VeriSign, Web Services Security Policy Language (WS-SecurityPolicy), July 2005.
- [18] IBM, Web Services Federation Language, (WS-Federation), July 8, 2003.
- [19] Arbaugh, W.A., Farber, J., Smith., J.M., A Secure and Reliable Bootstrap Architecture, in IEEE Computer Society Conference on Security and Privacy. IEEE, 1997, pp.65-71.
- [20] Sailer, R., Zhang, X., Jaeger, T., Van Doorn, L., Design and Implementation of a TCG-Based Integrity Measurement Architecture, proceedings of the 13<sup>th</sup> USENIX Security Symposium, , San Diego, California, August, 2004, pp.223-238.
- [21] Sailer, R., Jaeger, T., Zhang, X., Van Doorn, L., Attestation-based Policy Enforcement for Remote Access, in proceedings of the 11th ACM Conference on Computer and Communications Security, Washington DC, USA, October 2004, pp.308-317.
- [22] Haldar, V., Chandra, D., Franz, M., Semantic Remote Attestation – A Virtual Machine directed approach to Trusted Computing, in proceedings of the 3rd Virtual Machine Research and Technology Symposium, May 2004.
- [23] Sadeghi, A., Stübke, C., Property-based Attestation for Computing Platforms: Caring about properties, not mechanisms, in proceedings of the 2004 Workshop on New Security Paradigms (NSPW 2004), Nova Scotia, Canada, pp.67-77.
- [24] Open Vulnerability and Assessment Language, <http://oval.mitre.org/>
- [25] IBM and Cisco: White Paper, October 2004, <http://www-3.ibm.com/security/cisco/docs/wp-ibm-cisco-iisfcn.pdf>
- [26] Garfinkel, T., Pfaff, B., , Chow, J., Rosenblum, M., Boneh, D., Terra: A Virtual Machine-Based Platform for Trusted Computing, in Proceedings of the 19th ACM Symposium on Operating Systems Principles, 2003.
- [27] John L. Griffin, Trent Jaeger, Ronald Perez, Reiner Sailer, Leendert van Doorn, and Ram´on C´aceres. Trusted virtual domains: Toward secure distributed services. In Proceedings of the Workshop on Hot Topics in System Dependability, 2005.
- [28] Yuji Watanabe, Sachiko Yoshihama, Takuya Mishina, Michiharu Kudo, and Hiroshi Maruyama , Bridging the Gap between Inter-Communication Boundary and Inside Trusted Components, the 11th European Symposium on Research in Computer Security(ESORICS 2006), LNCS, Springer, 2006.
- [29] Linux Intrusion Detection System (LIDS), <http://www.lids.org/>.
- [30] OSGi Alliance, <http://www.osgi.org/>.
- [31] IBM Service Management Framework, <http://www-306.ibm.com/software/wireless/smf/>.
- [32] TCG Software Stack Specification Version 1.2, <http://www.trustedcomputing.org/specs/TSS>.
- [33] Java Cryptography Extension (JCE), <http://java.sun.com/products/jce/>.
- [34] RedHat Enterprise Linux, <http://www.redhat.com/>.
- [35] Direct Anonymous Attestation, <http://www.zurich.ibm.com/security/daa/>.

## ABOUT THE AUTHOR

Sachiko Yoshihama is a researcher in the Security & Privacy Department at IBM Tokyo Research Laboratory in Yamato, Kanagawa, Japan. Her research interests include Trusted

Computing, Web Services Security, and pervasive computing. She was an editor of the Trusted Mobile Platform specification (<http://trusted-mobile.org/>) which was a joint effort between IBM, Intel, and NTT DoCoMo, and aimed at defining Next-Gen reference architecture for secure mobile devices.

Tim Ebringer is a Research Staff Member at CA. Tim studied self-destructive software as part of his Ph.D., where software deliberately damages itself in order to accomplish certain security goals. He then spent a year at IBM Tokyo Research Laboratory, where he tampered with tamper-resistant hardware. Tim then tracked down fraud and corporate criminals at Deloitte Forensic, and was involved as part of a team of expert witnesses in the Kazaa case, a landmark copyright case to come before the Federal Court of Australia. He now works for CA in their research division, CA Labs. His aim is to become the most dangerous person in Computer Science. In his spare time, he studies mythological dogs.

Megumi Nakamura has received her B.E. and M.E. in engineering from Keio University, in 2002 and 2004, respectively. She is currently a researcher in the IBM Tokyo Research Laboratory. Her research interests include information security and privacy.

Seiji Munetoh has received his B.E. and M.E. degrees from Tohoku University, Sendai, Japan, in 1990 and 1992, respectively. He joined the IBM Tokyo Research Laboratory, Tokyo, Japan in 1992, and engaged in the research and development of VLSI, security system and trusted computing.

Takuya Mishina has received his B.E. and M.E. from the University of Tsukuba, in 2002 and 2004, respectively. He is currently a researcher in the IBM Tokyo Research Laboratory, and is interested in security technology and secure process design.

Hiroshi Maruyama is the director of IBM Tokyo Research Laboratory (TRL), leading approximately 180 researchers in the areas of computer science, service science, and certain areas of physical science. He joined IBM in 1983 at Japan Science Institute (later renamed to Tokyo Research Laboratory). Since then, he worked in various fields such as artificial intelligence, logic programming, natural language processing, machine translation, hand writing recognition, multi media, XML, Web services, and security. During 1996-1997, he spent one year in Software Group Headquarters in Somers, New York, where he evaluated emerging Internet technologies. XML was one of these technologies, and he started to work on XML technologies after he returned to TRL in 1997. His team developed XML Parser for Java, one of the first fully compliant XML processors at the time, which helped IBM to jumpstart XML enablement of their software products. He co-authored the book "XML and Java: Developing Web Applications" that have been sold more than 60,000 copies worldwide in 6 different languages. In 1997-2000, he served as an adjunct professor at Computer Science Department of Tokyo Institute of Technology, where he conducted research projects on Internet security. In 2003-2004, he was temporarily transferred to IBM's Business Consulting Services as a part of Research's On Demand Innovation Service (ODIS) program. Hiroshi received his Ph. D from Kyoto University in 1995. He had several management positions at TRL before becoming the director in Feb. 2006. His research interests include security & privacy, Web services, natural language processing, and computer science in general. He is an IBM Distinguished Engineer and a member of IBM Academy of Technology.

## **Appendix A**

Example 1: Message requesting for an Attestation Credential

```

<S:Envelope xmlns:S="..." xmlns=".../secext" xmlns:wsu=".../utility" xmlns:wst="...">
  <S:Header>
    <wss:Security>
      <wsse:BinarySecurityToken wsu:Id="targetAIK" ValueType="...X509v3">
        ... Binary encoded certificate of the AIK ...
      </wsse:BinarySecurityToken>
      <tcg:AttestationToken Id="myMeasurement">
        <tcg:Measurement ValueType="tcg:PCRComposite" Encoding="xsd:hexBinary">
          00020A010000003C770FDFE8CD1CA4AEE432B818DBE...
        </tcg:Measurement>
        <wsse:SecurityTokenReference>
          <wsse:Reference URI="#targetAIK"/>
        </wsse:SecurityTokenReference>
      </tcg:AttestationToken>
      <ds:Signature>
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
          <ds:SignatureMethod Algorithm="http://trustedcomputinggroup.org/2005/03/rsa_pcr" />
          <ds:Reference URI="#body">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <ds:Digest Value>LyLsF094hPi4wPU... </ds:Digest Value>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:Signature Value>MC0CFFrVLtRlk=... </ds:Signature Value>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#myMeasurement"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wss:Security>
    ...
  </S:Header>
  <S:Body>
    <wst:RequestSecurityToken>
      <wst:TokenType>saml:Assertion</wst:TokenType>
      <wst:SignChallenge> <wst:Challenge>Huehf...</wst:Challenge> </wst:SignChallenge>
      <wst:Supporting>
        <xxx:PlatformMeasurementDescription>
          ... platform measurement description ...
        </xxx:PlatformMeasurementDescription>
      </wst:Supporting>
    </wst:RequestSecurityToken>
  </S:Body>
</S:Envelope>

```

## Example 2: Response message for returning a SAML assertion as an Attestation Credential

```

<S:Envelope xmlns:S="..." xmlns=".../secext" xmlns:wsu=".../utility" Xmlns:wst="...">
  <S:Header>
    <wss:Security>

```

```

<wsse:BinarySecurityToken wsu:Id="vpKey" ValueType="...X509v3">
  ... Binary encoded certificate of the Verification Service's Key
</wsse:BinarySecurityToken>
<ds:Signature>
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#body">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>LyLsF094hPi4wPU... </ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue> MC0CFFrVLtRlk=... </ds:SignatureValue>
  <ds:KeyInfo>
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#vpKey"/>
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
</ds:Signature>
</wss:Security>
</S:Header>
<S:Body>
  <wst:RequestSecurityTokenResponse>
    <wst:TokenType>saml:Assertion</wst:TokenType>
    <wst:SignChallengeResponse>
      <wst:Challenge>Huehf...</wst:Challenge>
    </wst:SignChallengeResponse>
    <wst:RequestedSecurityToken>
      <saml:Assertion saml:AssertionID="..." saml:Issuer="..." saml:IssueInstant="...">
        <saml:AttributeStatement>
          <saml:Attribute saml:AttributeName="trusted" ...>
            <saml:AttributeValue>true</saml:AttributeValue>
          </saml:Attribute>
          <saml:Attribute saml:AttributeName="pcr" ...>
            <saml:AttributeValue>
              00020A010000003C770FDFE8CD1CA4AEE432B818DBE...
            </saml:AttributeValue>
          </saml:Attribute>
        </saml:AttributeStatement>
      </saml:Assertion>
    </wst:RequestedSecurityToken>
  </wst:RequestSecurityTokenResponse>
</S:Body>
</S:Envelope>

```